De la réutilisation à l'adaptabilité

Philippe Roose

Thèse d'Habilitation à Diriger les Recherches

Spécialité Informatique

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

De la réutilisation à l'adaptabilité

Par

Philippe ROOSE
LIUPPA/IUT de Bayonne
2, Allée du Parc Montaury
64600 Anglet

Mail : Philippe.Roose@iutbayonne.univ-pau.fr

Soutenue le 27/11/2008

devant le jury composé de :

Lapayre Jean-Christophe	Professeur au LIFC, Université de Franche-Comté	Rapporteur
Libourel Thérèse	Professeur au LIRMM, Université de Montpellier 2	Rapporteur
Rieu Dominique	Professeur au LIG, UMF, Grenoble	Rapporteur
Réveillère Laurent	Maitre de Conférences, Labri, ENSEIRB	Examinateur
Laforest Frédérique	Maitre de Conférences, LIRIS, INSA de Lyon	Examinateur
Pham Congduc	Professeur au LIUPPA, UPPA	Référent

Table des matières

Table des figures

Remerciements

C'est un peu comme la cérémonie des Césars, l'exercice des remerciements est particulier. Beaucoup de monde a contribué directement ou indirectement à ce travail, aussi et sans transition, les nominés pour les remerciements sont dans les lignes qui suivent.

En tout premier lieu, je remercie les rapporteurs de cette Habilitation à diriger les Recherches. Leur confiance accordée en s'associant à ce travail est pour beaucoup dans sa réussite. Leurs retours toujours constructifs ont permis son amélioration de manière très significative. Et, au-delà des critères purement « scientifiques », il y a le caractère humain qu'il est difficile de dissocier, merci pour leur gentillesse et leurs encouragements. J'ai le jury que je souhaitais !

On dit qu'une HdR est un travail personnel. Si cela est vrai, elle n'en est pas moins le résultat de travaux menés en commun, car n'oublions pas que la recherche n'est pas le travail d'une seule personne. Aussi, mes premiers remerciements s'adressent au collègue avec qui j'ai démarré la recherche et avec lequel je travaille depuis plus de 11 ans : Marc Dalmau. Ca fait un petit moment que l'on en parle, on y a passé quelques heures de discussions. Aussi, quand le moment opportun s'est présenté pour se lancer dans cette aventure, c'est naturellement qu'on y a plongé tous les deux, il était inconcevable de faire autrement. Il est des personnes rares dont la rencontre change un destin. Je le remercie de tant de choses que je ne peux les citer ici. Je pense que nous formons un beau et bon couple…de travail !

Je ne peux oublier les personnes qui ont activement participé à l'amélioration de ce mémoire. Sophie, pour ses remarques toujours pertinentes et son œil de lynx pour les coquilles. Frédérique Laforest, avec sa vision extérieure, pour sa relecture minutieuse et ses commentaires constructifs. Toutes deux ont contribué à ce travail. Je remercie également Laurent Réveilère d'avoir gentiment accepté d'être au jury. Merci également à Congduc pour m'avoir encouragé et soutenu tout au long de ce travail.

Je remercie également les personnes avec qui j'ai travaillé : doctorants, directeurs de thèses, ingénieurs. Merci donc à Congduc Pham, Franck Barbier, Franck Luthon, Sophie Laplace, Manu Bouix, Christine Louberry, Raphaël Michel, Cyril Cassagne. Ce travail n'aurait pu exister sans vous également.

Je ne peux décemment pas continuer ces pages ô combien importantes sans remercier ma femme, Valou, toujours présente, qui m'a fait confiance, m'a aidé du mieux possible durant ces quelques mois. Et puis bien évidemment, ma fille, Haize, qui ne s'est pas rendue compte de ce qui se passait, mais qui par ses sourires et éclats de rires a su décharger mon stress. Il ne faut pas oublier Lorea, qui a commencé sa vie en même temps que les premières pages du mémoire pour naître lors de la rédaction de la conclusion. Elle a participé indirectement à me booster dans ce travail en étant, sans le vouloir, l'échéance de la rédaction.

Merci à ma mère qui depuis toujours m'a suivi, soutenu, encouragé dans ma vie. Sa prudence a toujours été un frein nécessaire à mon éternel optimisme.

L'environnement professionnel a une influence sur le travail. C'est pour cela que je remercie l'ensemble des collègues de l'IUT pour participer à la très bonne ambiance qui règne ici. Un merci tout particulier à Jean-Marc et Pierre pour nos séances de footing/natation voire triathlon et VTT. Ça fait du bien, ça vide la tête pour mieux la regonfler ensuite. Le sport n'est pas tout, il faut aussi se détendre, merci donc aux Christophe, Manu, Lopis, Patchoc pour l'animation des soirées GASCUB et Fêtes de Bayonne…

Je termine par une pensée à Mané.

Philippe

Chapitre 1 Introduction

Qu'est-ce que le contexte ?

Depuis plusieurs années, l'évolution naturelle des applications vers la distribution a mis en évidence le besoin d'informations autre que celles uniquement nécessaires aux traitements. Traditionnellement, les applications étaient conçues sur le modèle d'entrée/sorties, à savoir, une entrée est donnée à l'application qui produit une sortie. Ce schéma trop restrictif est maintenant dépassé [Liebermann, 2000], les données ne sont plus forcément explicitement identifiées, les traitements ne dépendent plus des données fournies, mais également de l'heure, de la localisation, des préférences des utilisateurs, de l'historique des interactions, etc., c'est le contexte de l'application dont [David, 2005] donne une définition informelle mais assez représentative « *Le contexte d'exécution d'une application regroupe toutes les entités et situations externes qui influent sur la Qualité de service/Performances (qualitative et quantitative) telle que perçue par les utilisateurs* ».

C'est ainsi qu'au fil du temps, les concepteurs et développeurs ont dû intégrer à leurs applications des tâches d'acquisition de l'environnement d'exécution permettant ainsi aux applications dans un premier temps de prendre en compte le contexte, puis de devenir « sensibles au contexte » et d'adapter leurs traitements en conséquence puis, dans un second temps, d'évoluer par reconfigurations dynamiques de manière à répondre au mieux aux sollicitations. Il est évident que pour s'adapter au contexte, la première des choses à faire est d'en avoir une bonne connaissance, et, pour s'y adapter dynamiquement, d'avoir connaissance de ses évolutions.

D'un point de vue activité de recherche, le contexte s'immisce maintenant dans quasiment toutes les thématiques. Ainsi, de nombreux travaux portent sur sa mesure, sa modélisation, sa prise en compte, son évaluation, etc. et ce dans quasiment tous les domaines (Réingénierie, IHM, Grilles, Applications Distribuées, Informatique ubiquitaire, Systèmes d'Informations, Sécurité, etc.). Néanmoins, il faut reconnaître que la notion de contexte en informatique n'est pas une nouveauté. Dès le début des années 1990 le centre de recherche d'Olivetti avec l'ActiveBadge [Harter, 1994] et surtout le regretté Xerox PARC avec le PARCTab System [Want, 1995] sorti en 1993 ont posé les fondements des applications modernes sensibles au contexte.

Le principe architectural (Figure 1) permettant la prise en compte du contexte dans les applications est assez classique. On en trouvera un exemple dans [Tigli, 2006]. Il se « résume » à une superposition de couches correspondant à l'acquisition des informations contextuelles, la gestion du contexte puis l'application (et/ou son adaptation).

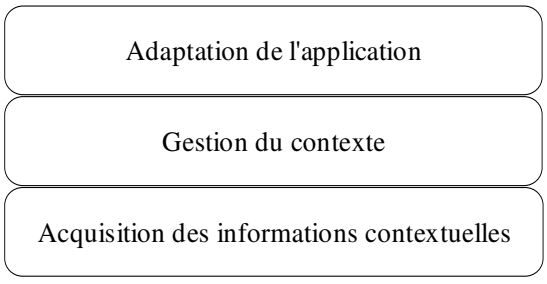


Figure 1 : Architecture en couche des applications sensibles au contexte [Tigli, 2006]

Les personnes à l'origine du terme « sensible au contexte » (*context-awareness*) sont Schilit et Theimer [Schilit, 1994] expliquant que c'est « *la capacité d'une application et/ou d'un utilisateur mobile de découvrir et réagir*

aux changements de sa situation ». De très nombreuses autres définitions ont été données [Pascoe, 2000] [Dey, 1998] mais aucune ne fait pour l'instant autorité.

Le contexte peut se diviser en trois couches. La première appelée ici « type de contexte » permet de réaliser l'acquisition du contexte selon 4 classes types [Lavirotte, 2006] [Chen, 2000] [David, 2005] et d'en proposer une représentation selon un formaliste adéquat.
Le premier type de contexte est dit **Environnemental** : c'est le contexte physique, il représente l'environnement externe dont les informations sont obtenues via des capteurs. Ces informations sont relatives à luminosité, la vitesse, la température, etc. Le second type, appelé **Utilisateur** donne une représentation du ou des utilisateurs interagissant dans les applications (langue du locuteur, localisation, présence, etc.). C'est le profil de l'utilisateur. Le troisième concerne le **Matériel.** C'est certainement le plus « classique », il informe sur les ressources disponibles (mémoire, charge processeur, connexions, bande passante, débit, etc.). On y trouve également les ressources disponibles comme l'affichage, le type d'hôte (PDA, PC, etc.), etc. Enfin, le contexte **Temporel** permet de conserver et de prendre en compte l'historique (date, heure, synchronisations, actions réalisées, etc.)

La deuxième couche, appelée « moyens de mise en œuvre de la gestion du contexte » est chargée de la gestion du contexte [Kjær, 2007] [Laforest, 2008] en se basant sur les représentations de la couche précédente. Elle met en évidence les **Services** spécifiant l'environnement logiciel de l'application (*intergiciel, système d'exploitation, etc.*), le **Stockage** des données relatives au contexte avec les services permettant de les récupérer, la **Qualité** donnant une mesure sur le service ou les données traitées et enfin la **Réflexivité** permettant de représenter l'application elle-même.

Enfin, la troisième couche offre les mécanismes permettant l'adaptation au contexte. On y trouve la **Composition** de composants selon des événements contextuels, la **Migration** permettant le déplacement des entités et enfin l'**Adaptation** assurant l'évolution de l'application selon le contexte et qui peut être transparente (le middleware s'en charge), selon un profil (stocké) ou selon des règles fournies par l'application ou les utilisateurs.

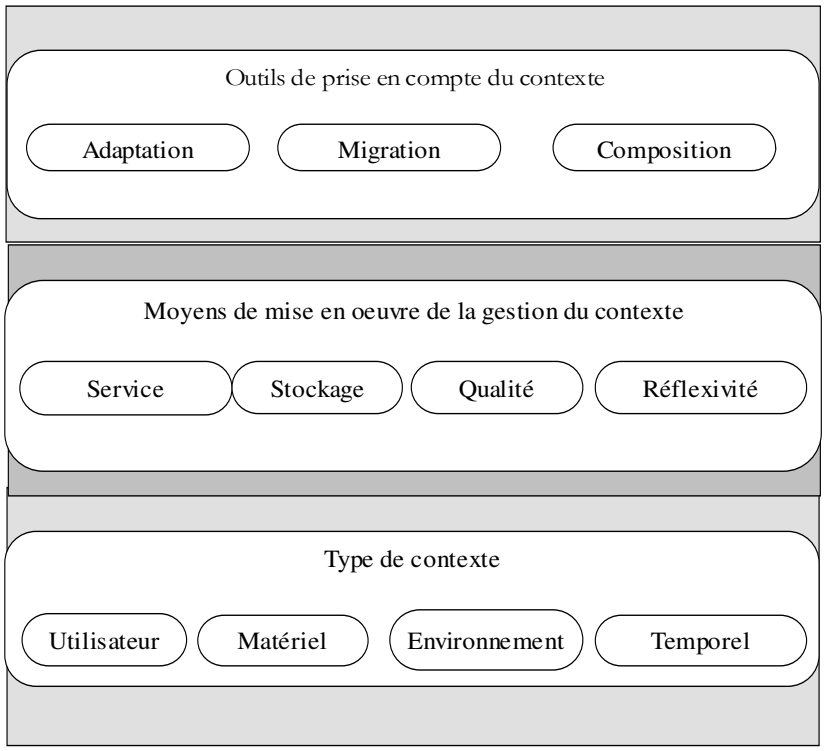


Figure 2 : Taxonomie du contexte

Afin d'éviter tout malentendu, je dois préciser que je n'ai pas travaillé sur le contexte en lui-même ni sur sa modélisation, mais sur des techniques, outils et méthodologies en lien avec le contexte. Ainsi, après la présentation de la taxonomie réalisée ci-dessus, je vais maintenant expliquer les différents travaux que j'ai menés pour lesquels le contexte a eu une influence. Ces travaux se scindent en deux parties. La première,

essentiellement présentée dans le premier chapitre utilise le contexte utilisateur et matériel afin de mettre en œuvre la réingénierie d'applications et la réutilisation de composants permettant ainsi ce qui peut s'appeler la reconfiguration évolutive [Ketfi, 2004], c'est-à-dire une réponse aux besoins de reconfiguration. Dans ce type de travaux, j'ai utilisé le contexte pour essentiellement intervenir au niveau système d'information, sans changer la structure applicative, et ceci dans un cadre de contexte figé. La seconde partie, exposée dans les chapitres 2 à 4, propose une approche permettant d'agir d'un point de vue fonctionnel et structurel sur l'application, toujours en se basant sur les informations contextuelles. Elle propose des outils méthodologiques et d'implémentation permettant la réalisation d'applications adaptables dans un contexte mouvant dont l'objectif est d'assurer une reconfiguration perfectible, c'est-à-dire permettant d'améliorer les performances d'une application même si son fonctionnement est cohérent.

Prise en compte, gestion et adaptation au contexte

J'ai été amené à travailler sur les trois couches identifiées sur le schéma précédent (Figure 2). Dans une première série de travaux (chapitre 1), je me suis intéressé aux domaines de la réingénierie et de la réutilisation d'applications à base de COTS Products (*COTS - Composants sur étagères)* en me focalisant sur la prise en compte du contexte (couches 1 du schéma précédent) et sur un travail au niveau du système d'information avec l'adaptation et migration de ces informations (couche 3) sans intrusion au niveau fonctionnel, c'est-à-dire, sans modification d'un point de vue fonctionnel des applications.

Fort du constat et des limites des approches non intrusives (sans modification fonctionnelle) , je propose trois modèles **Osagaia** (modèle de composant), **Korrontea** (modèle de connecteur) ainsi que le modèle de la plate-forme **Kalinahia** permettant la mise en œuvre des composants, des connections et leur supervision. Ces modèles sont des outils d'implémentation précieux. Leur conception a été guidée par la plate-forme intrusive **Kalinahia**. Ils permettent des reconfigurations de l'application pilotées par la plate-forme. L'objectif de ces reconfigurations est de permettre à l'application d'assurer une qualité de service suffisante aux utilisateurs.

Le modèle de composants Osagaia permet la connexion dynamique entre composants. L'architecture des composants permet de notifier des informations contextuelles locales sur leur état à la plate-forme et de recevoir en retour des commandes agissant sur leur cycle de vie.
Les connecteurs permettent de mettre en œuvre les communications entre ces composants. Ils ont été conçus selon le même principe que les composants et peuvent donc également donner des informations contextuelles relatives au matériel et au temps (couche 1). Ils peuvent également être supervisés par la plate-forme (couche 3) afin de modifier les connexions. Ce modèle permet également de mettre en œuvre différentes politiques de communication de données afin de répondre aux besoins.

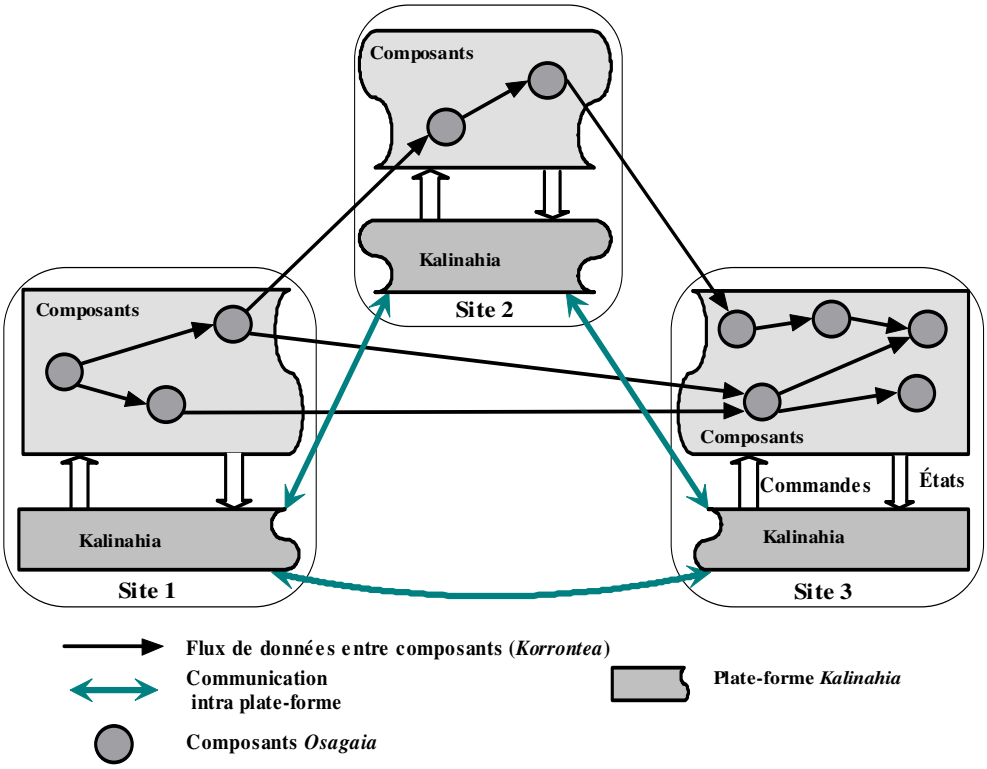


Figure 3 : Schéma général

Le principe de fonctionnement est le suivant : les informations contextuelles notifiées par les composants et connecteurs sont envoyées à la plate-forme, qui récupère les états représentant le contexte d'exécution local de chaque entité (de communication ou fonctionnelle) et réagit en conséquence en envoyant les commandes nécessaires à la reconfiguration de l'application. Cette dernière peut prendre différentes formes : un composant peut être remplacé par un autre, l'assemblage ou la composition des composants peut être modifié, les communications entre composants peuvent être réorganisées, tout en envisageant si nécessaire des migrations de composants sur différents hôtes.

Enfin, je propose une méthode spécifique allant de la conception à l'implémentation en passant par le déploiement des composants sur les différents hôtes. L'approche, certes complexe par la quantité de graphes, possède une unicité de représentation. Nous utilisons des graphes orientés pour représenter tant l'évaluation de la Qualité de Service relativement au contexte et à ses évolutions que l'application avec ses différentes configurations en fonction des contraintes (synchronisation).
Les différents graphes permettent d'identifier les assemblages de composants et de flux communs à plusieurs configurations et de réaliser l'évaluation de la qualité des services de l'application.

Plan du mémoire

Le premier chapitre intitulé « Processus de réingénierie » porte principalement sur la première phase d'acquisition des informations contextuelles. Je me suis plus particulièrement intéressé aux applications totalement existantes (*réingénierie*) ou partiellement existantes comme les COTS.
Dans un premier temps, l'objectif est de réaliser une analyse des informations produites par des modules logiciels afin de prendre en compte mais également de fabriquer (si nécessaire) automatiquement à l'aide de connecteurs - lorsque c'est possible - des informations contextuelles de plus haut niveau. L'application est ici vue comme un ensemble de boîtes noires (*code exécutable, documentation*) déployées et interconnectées dont seules les entrées/sorties sont prises en considération.
Dans un second temps, je me suis intéressé à l'intégration de Composants sur Etagères (ou *COTS Products*). L'objectif est ici une analyse des assemblages de COTS sélectionnés afin de s'assurer de la faisabilité du déploiement, et donc de la réalisation de l'application. Nous nous situons ici également dans le domaine de l'acquisition des informations contextuelles dans le sens où nous ne gérons que la phase d'acquisition du contexte et de production d'informations permettant des prises de décisions concernant la possibilité d'assemblage des COTS Products.

Si l'on retrouve les différentes granularités de visibilité (boîte noire, grise, blanche) dans le domaine des COTS Products, nous n'abordons dans ce chapitre les COTS Products qu'en terme de boîte noire, le niveau d'intervention pour leur intégration ne porte que sur le contexte et non sur le choix du produit ni sur son éventuelle adaptation.

Le schéma suivant présente, pour chacune des couches, les types de contexte sur lesquels nous intervenons. Ce schéma va être repris ici pour chaque chapitre afin de bien identifier son niveau d'intervention.

	Adaptation	Migration	Composition	
Adaptation de l'application	X	X		
	Service	Stockage	Qualité	Réflexivité
Gestion du contexte				
	Environnement	Utilisateur	Matériel	Temporel
Acquisition des informations contextuelles	X	X		

Figure 4 : Domaines d'interventions (chapitre 1)

Le deuxième chapitre intitulé « Outils d'adaptation permettant la prise en compte du contexte » est transversal aux trois couches de la Figure 1. Un modèle de plate-forme de supervision réflexive y est présenté afin de proposer les services nécessaires à la gestion du contexte. Un modèle de composant supervisable ainsi qu'un modèle de connecteur y est également proposé. Ces deux modèles offrent les mécanismes d'adaptation, de migration et de composition nécessaires à l'adaptation de l'application.

	Adaptation	Migration	Composition	
Adaptation de l'application	X	X	X	
	Service	Stockage	Qualité	Réflexivité
Gestion du contexte	X	X		X
	Environnement	Utilisateur	Matériel	Temporel
Acquisition des informations contextuelles			X	X

Figure 5 : Domaines d'interventions (chapitre 2)

Le troisième chapitre intitulé « Contexte et Qualité de service » présente comment la notion de qualité de service, intimement liée à celle de contexte, est intégrée dans nos travaux. En effet, fournir la qualité de service adéquate à un utilisateur ou, plus généralement à une application, demande d'avoir une connaissance à la fois du contexte de l'application (*qu'est-il possible de faire et comment ?*) mais aussi du contexte de l'utilisateur (*que souhaite-t-il ?*). Aussi, nous proposons un modèle formel de la qualité de service selon les deux critères intrinsèques (*fonctionnalité*) et contextuels (*dans et sous quelles conditions*). Ce modèle permet d'obtenir en permanence, selon le choix des composants, une mesure de la qualité de service prenant en compte les souhaits des utilisateurs mais également les possibilités de l'application en cours d'exécution. Si l'on resitue ce chapitre selon la Figure 1, nous nous plaçons clairement sur la deuxième couche, le modèle de qualité de service offre un mécanisme de gestion du contexte. Néanmoins, afin de correctement prendre en compte cette qualité, nous devons ajouter la prise en compte matérielle et temporelle qui n'avait pas été réalisée lors des travaux du premier chapitre, tout en continuant à prendre en compte le contexte environnemental et utilisateur.

	Adaptation	Migration	Composition	
Adaptation de l'application				
	Service	Stockage	Qualité	Réflexivité
Gestion du contexte			X	
	Environnement	Utilisateur	Matériel	Temporel
Acquisition des informations contextuelles	X	X	X	X

Figure 6 : Domaines d'interventions (chapitre 3)

Enfin, le quatrième chapitre « Représentation des applications et de leur qualité de service » a pour objectif de proposer une modélisation d'applications sensibles au contexte ayant pour but d'assurer une qualité de service aux utilisateurs. L'approche formelle utilisant des graphes orientés se base sur l'architecture des applications proposée au Chapitre 3 tout en prenant en compte la qualité de service telle qu'elle a été définie au chapitre précédent. Les différents graphes proposés vont du niveau conceptuel au niveau d'implantation permettant de générer directement les graphes de déploiement et d'implantation nécessaires qui seront ensuite utilisés par la plateforme pour la reconfiguration dynamique de l'application. L'objectif étant d'assurer une qualité de service acceptable, le contexte de qualité est également pris en compte tout au long de la démarche et surtout lors des étapes de reconfiguration puisque c'est une modification du contexte (au sens large) qui déclenchera la modification du déploiement de l'application. L'objectif est de continuer à assurer une qualité de service « acceptable » malgré le contexte mouvant en provoquant des reconfigurations dynamiques de l'application.

	Adaptation	Migration	Composition	
Adaptation de l'application	X	X	X	
	Service	Stockage	Qualité	Réflexivité
Gestion du contexte	X	X	X	X
	Environnement	Utilisateur	Matériel	Temporel
Acquisition des informations contextuelles	X	X	X	X

Figure 7 : Domaines d'interventions (chapitre 4)

A l'issue de ces quatre chapitres présentant les différents travaux menés au cours de ces 11 dernières années, une dernière partie présente un curriculum vitae détaillé. La première partie présente brièvement mon implication pédagogique avec les différentes charges assumées, et particulièrement celle de Responsable du Pôle Multimédia. La seconde partie plus détaillée présente mes activités de formation par la recherche avec les encadrements et co-encadrements de stages de DEA/Master Recherche/Ingénieur et Thèses. On y trouvera également une liste des projets dans lesquels j'ai été ou suis impliqué ainsi que des activités d'expertises et de participation à des comités (lecture, programme, organisation). Enfin, la liste des publications viendra clore de chapitre et ce mémoire.

.

Chapitre 2 Processus de réingénierie

La prise en compte du contexte impose aux applications d'être capables de le capturer mais également de réagir par quel que moyen que ce soit (*adaptation, migration, composition*) afin de s'adapter aux changements du contexte et de proposer **des adaptations fonctionnelles ou de données** (par transformation de contenu et/ou de transmission) [Laforest, 2008]. La gestion du contexte nécessite donc des architectures spécifiques capables de réaliser des captures du contexte et de réagir en conséquence.
Dans le domaine de la réingénierie, les informations capturées ne sont pas forcément directement exploitables pour des raisons techniques (format, localisation, etc.) ou pour des raisons sémantiques nécessitant ainsi de réaliser de la composition ou de l'extraction d'informations, etc. Ce travail a été abordé au cours des travaux sur ELKAR, une architecture et une méthode permettant la réingénierie d'applications par adaptation de données (adaptation, migration, composition).

1.1 ELKAR : Architecture logicielle pour la réingénierie

ELKAR part du constat que de plus en plus d'applications sont constituées de modules répartis (un module correspond à une cellule de production, un automate, un composant logiciel, un COTS, etc.) interconnectés par un réseau. A la fin des années 90, avec le déploiement de la technologie Internet, on a perçu de manière grandissante la nécessité de faire migrer des applications distribuées non coopératives (et souvent anciennes) vers la coopération. Le problème est que de telles applications ne savent pas se synchroniser ni échanger des informations ni surtout se constituer en groupes de travail (ensemble de modules ayant un but commun et formant une composante propre). Si l'on parvient généralement à établir des communications entre eux à l'aide d'opérateurs externes (humains ou machines dédiées de type automate ou autre – cf Figure 8), il est quasiment impossible d'obtenir une organisation en groupes de travail capable d'évoluer dans le temps grâce à l'intégration de nouveaux modules ou le départ de certains.

Au moment de ces travaux, il était très fréquent de rencontrer (et certainement malheureusement encore) des postes de travail dont tout ou partie de l'activité consiste en la (re-)saisie d'informations déjà disponibles ou que l'on aurait pu synthétiser à un moment donné mais que l'on n'a pas su ou pas pu récupérer au moment opportun. Il faut reprendre ces applications. Le problème est que les responsables informatiques hésitent, voire même refusent toute profonde modification de ces dernières qui tournent depuis longtemps, qui sont fiables et pour lesquelles le code n'est pas toujours modifiable pour diverses raisons : codes sources non maintenus ou perdus, compétences absentes, etc.

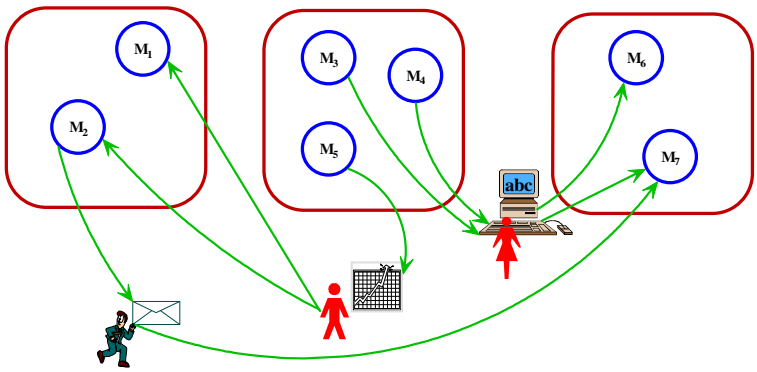


Figure 8 : Coopération par opérateurs externes

C'est ici que nous intervenons en tentant de faire disparaître tout ou partie de ces opérateurs externes (*humain, mécanisme ad hoc*) assurant la coopération, c'est-à-dire, en gérant automatiquement la circulation des informations et l'organisation en groupes de travail dynamiques [Aniorte, 2000a] [Aniorte, 2000b]. Nous proposons que les informations qui étaient auparavant véhiculées à l'aide d'opérateurs externes circulent maintenant de manière automatique par le réseau allant directement des producteurs aux consommateurs avec éventuellement des adaptations, de la composition et de la migration de données.

L'exploitation actuelle des systèmes de production nécessitent de la flexibilité [Martineau, 1996] [Martineau, 1997]. Ceci suppose que les divers modules puissent se constituer en groupes de travail évoluant au cours du temps (ordonnancement de production piloté par les commandes, insertion d'une production exceptionnelle dans une chaîne en cours, etc.). Une telle évolution du système ne se traduit pas par l'introduction de fonctionnalités nouvelles ou différentes dans les modules existants mais par la modification de leurs interactions.

Pour ce faire, nous avons mis en place une architecture logicielle [Medvidovic, 2000] à base de composants (appelés modules), de connecteurs entre composants métiers avec une glue associée aux connecteurs réalisée par les opérateurs externes. La configuration de l'architecture logicielle est représentée par les différents graphes représentant les différentes connections composants/connecteurs pouvant évoluer au cours du temps. Enfin, la notion de composite se retrouve dans la structuration en groupes de travail.

Une des originalités de notre approche réside dans sa partie connecteur. Là où traditionnellement les modèles d'interconnexion parlent de pipe & filters, RPC, etc. [Mehta, 2000], nous avons, dans le cas présent, des opérateurs externes (humains, automates, etc.) chargés de réaliser la glue et permettant de manière informelle les liaisons multiples dont l'expression est un problème récurrent [Magee, 1996]. Tout l'enjeu réside dans l'expression des interactions réalisées par ces opérateurs externes dans un langage informatique sans perdre pour autant d'informations.
Après analyse des informations manipulées et des actions réalisées par les opérateurs externes, nous identifions 3 types de communications : les communications directes à l'aide d'événements, les messages qui fonctionnent selon un mode différé tandis que les données représentent des informations persistantes.

1.1.1 Connecteurs dans ELKAR

Le problème des interactions multiples a été longuement étudié [Magee, 1996] [Ammour, 2004] avec des solutions proposées au cas par cas. Nous avons choisi une approche qui nous est apparue plus souple. Nous nous sommes inspirés du concept de règles ÉCA (*Événement-Condition-Action*) hérité des Bases de Données Actives (BDA) [Dayal, 1988].
Lors de la représentation de liaisons multiples, il manque généralement l'expression de la dynamique. Ainsi, au lieu de mentionner uniquement la somme des interactions, nous insérons une règle détectrice[1] [Roose, 2001] permettant de spécifier cette dynamique. Contrairement à l'architecture SAFRAN [David, 2005] qui utilise le même principe pour exprimer les politiques d'adaptation, nous avons étendu le 'C' de É**C**A pour ne pas « simplement » faire une évaluation booléenne (avec éventuellement des contraintes temporelles [Dayal, 1988] [Front, 1997]). Nous voulons que la condition puisse exprimer une condition algorithmique aussi complexe soit-elle.
Ainsi, la règle ÉCA permet de réaliser simplement le connecteur lié à l'adaptation alors que le couple « *règle détectrice-règle ÉCA* » permet de résoudre le problème de la composition de connecteurs et de lever les ambiguïtés précédemment mentionnées.

1.1.2 Démarche dans ELKAR

L'objectif était de proposer une approche globale allant de la modélisation jusqu'à l'implantation. Ce mémoire ne portant pas sur les aspects de mise en œuvre pratique, la description de l'implantation de plate-forme ne sera pas faite ici. Néanmoins, pour faciliter cette dernière, un langage de spécification a été défini pour réaliser une vérification formelle de l'analyse, puis dériver des règles actives (inspirées des règles ECA) qui seront intégrées à la plate-forme coopérative.

[1] Les règles détectrices permettent de passer outre les restrictions des conditions booléennes communément rencontrées et permettent d'obtenir des événements composés représentant des situations complexes algorithmiquement exprimables.

La méthode qui a été proposée permet une approche par décomposition hiérarchique en structurant dans un premier temps l'application en groupes de travail dynamiques et en mettant en évidence dans un second temps la circulation des informations, c'est-à-dire les interactions entre ces groupes.
Ce type d'approche a la particularité d'avoir une granularité de composition élevée favorisant la réutilisation. Or, la réingénierie sans aucune modification des modules est similaire à de la réutilisation. Ainsi, les modules répertoriés représentent des entités bien identifiées, plus proches de l'application et du processus que de la procédure.

Les échanges de données entre connecteurs sont réalisés à l'aide de trois types d'éléments de coopération :

- **Événements** : Anatole France disait « *Mais qu'est-ce qu'un événement ? Est-ce un fait quelconque ? Non pas ! me dites-vous, c'est un fait notable.* ». Ainsi, un événement peut se définir comme un fait important digne d'être remarqué. D'un point de vue purement informatique, un événement est un élément de notification suffisant en soi et possédant un caractère prioritaire. La sémantique d'un événement est exprimée par la simple occurrence de celui-ci, on dit qu'il se *suffit* à lui même.
- **Messages** : Les objets communiquent entre eux par le biais de messages. Dans la modélisation objet, le concept de message est l'unité de communication permettant de relier dynamiquement entre elles les différentes entités identifiées. Un message est une information éphémère (en opposition aux données ci-dessous) pourvue d'un contenu sémantique (contrairement aux événements ci-dessus dont la sémantique est leur propre occurrence).
- **Donnée :** Selon le dictionnaire, une donnée est une représentation conventionnelle d'une information (fait, notion, ordre d'exécution) sous une forme (analogique ou numérique) permettant d'en faire un traitement automatique. La notion de « *représentation conventionnelle* » permet d'exprimer l'aspect structuré d'une donnée. Dans le cadre de nos travaux, les données représentent les informations persistantes du système d'information (SI) qui existent déjà (on ne modifie pas le SI).

L'approche retenue est descendante. Elle part du niveau d'abstraction le plus élevé (constitution en groupes de travail - entités entièrement virtuelles) en allant vers le niveau le plus bas (modules composant les groupes de travail). Elle permet également de mettre en évidence la dynamique de ces groupes (leur composition est variable au cours du temps, les modules peuvent entrer/sortir). Puis nous mettrons en évidence à chaque étape les interactions nécessaires, c'est-à-dire, les événements clés de la dynamique et les informations (persistantes ou non) qui doivent être échangées et partagées entre et dans ces groupes de travail.

De manière spécifique à la réingénierie, nous identifions les informations manquantes (appelées éléments de coopération) qui étaient auparavant créées par la coopération externe : la glue, ajoutée par les connecteurs externes.

1.1.3 Présentation de la méthode de réingénierie

L'automatisation des coopérations externes, l'amélioration (quantitative mais aussi qualitative) des communications et les possibilités de flexibilité vont contribuer à l'amélioration de l'application. Dans la mesure où nous fondons la coopération sur la communication, la mise en évidence des informations produites par cette coopération et la mise en œuvre de groupes de travail constitués de modules, trouveront leur place dans chacune des étapes de la méthode que nous allons décrire.
Ces travaux ont été réalisés dans le cadre d'une collaboration avec l'entreprise de messagerie « Messageries de l'Adour», appartenant depuis aux transports « Alloin ». L'exemple qui sert d'illustration est directement issu de cette collaboration. Il porte sur les traitements et la gestion des colis à destination nationale ou internationale qui tient compte de la particularité de la messagerie qui est un nombre important de colis de petite taille traités avec une remontée quasi immédiate des informations (mise sur palette, chargement, livraison, en souffrance, litige, etc.).

1.1.3.1 Inventaire

L'inventaire regroupe deux étapes. La première permet l'identification de groupes de travail (entité logique propre regroupant un ensemble évoluant dans le temps de modules éventuellement distants travaillant à l'accomplissement d'une tâche commune) et des interfaces nécessaires et requises (Figure 9). Elle correspond souvent au découpage structurel (organisation en services) déjà en vigueur dans l'entreprise lorsqu'il existe. Chaque groupe de travail représente une entité à part entière, c'est à dire qu'il remplit un

rôle précis, nécessite un certain nombre d'informations et en produit d'autres. En résumé, un groupe de travail a un sens et une fonction au niveau de l'organisation. Dans un langage plus proche des architectures logicielles, un groupe de travail représente typiquement un composant composite pouvant être constitué par des composants locaux ou distants.

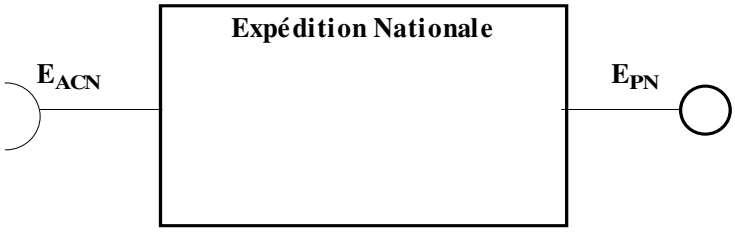


Figure 9 : Groupes de travail et interface

Les modules peuvent communiquer selon trois schémas différents : par événement, par message et par partage de données (stockées dans une base de données du système d'information).
A chaque expression de composite, il est nécessaire d'adjoindre un dictionnaire des données (informel) exprimant en langage naturel la sémantique et le type associé (pour ne pas surcharger les dessins/texte, nous ne montrons que l'aspect événementiel – les données et messages se gèrent de manière identique).
Dictionnaire :

- **E_{ACN} (événement)** : Un colis destiné à la zone nationale (N) est signalé. Cela correspond au passage d'un colis sous le portillon de l'un des tapis roulants d'acheminement.
- **E_{PN} (événement) :** Un service d'expédition a composé une palette et le signale. Cette palette quitte le quai pour être chargée dans un camion.

Ce schéma indique ici pour le composite *Expédition Nationale*, qu'il peut (qu'il peut ne veut pas dire qu'il va) réagir à l'événement E_{ACN} (arrivée de colis à destination nationale) et qu'il peut lever l'événement E_{PN} (palette à destination nationale prête) lorsqu'un certain nombre de colis ont été mis sur palette.
Une fois ce travail réalisé au niveau des composites, nous pouvons entrer dans le détail de chacun d'eux et étudier le niveau modules de la même manière, puis réaliser les liaisons en s'aidant du dictionnaire.
Pour ce faire, il est nécessaire de connaître l'ensemble des modules présents dans le système. Ils peuvent être de nature aussi bien logicielle (gestion des stocks, interfaces, etc.) que matérielle (cellule de production, automate programmable, etc.). Ensuite, nous identifions les composants composites auxquels ils sont susceptibles d'appartenir. Nous utilisons pour cela notre connaissance du rôle joué par chaque module. Nous allons ensuite répertorier les événements, les messages et les données produits par chacun de ces modules ainsi que ceux nécessaires à leur bon fonctionnement à l'instar de ce qui a été fait à la première étape pour les groupes.

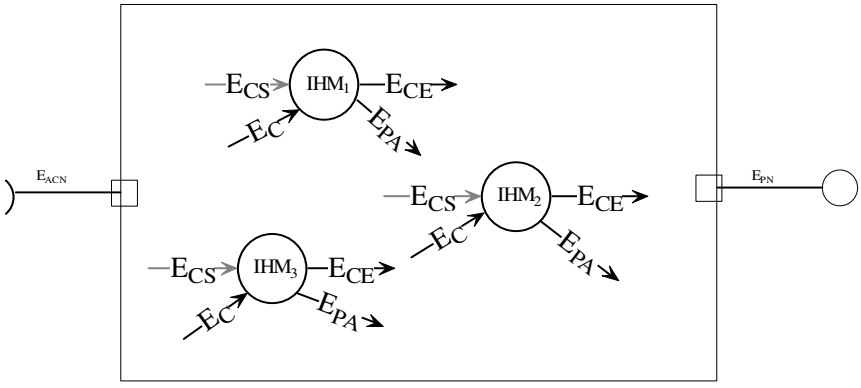


Figure 10 : Appartenance potentielle des modules aux groupes

Les modules IHM_1, IHM_2 et IHM_3 constituent le groupe d'*Expédition Nationale* et correspondent aux postes de travail des opérateurs chargés de préparer un plan de chargement. Ils permettent d'avoir une vue en temps réel de l'état des zones de stockage. Lorsqu'un colis est sélectionné pour être envoyé, il doit être

immédiatement enlevé des autres interfaces de manière à ne pas être choisi plusieurs fois (on évite ainsi aux manutentionnaires de chercher un colis qui n'est plus présent, ces problèmes sont une source importante de perte de temps dans ce type de manipulation).
Sans détailler tous les événements mis en évidence, voici ce qu'une entrée dans le dictionnaire pourrait être :

- E_C : Signale l'arrivée d'un nouveau colis à prendre en compte.

Même si pour des raisons évidentes de place nous ne le montrons pas, il est nécessaire de réaliser pour les messages et les données ce que nous réalisons pour les événements.

Notre approche de la méthode ressemble jusqu'ici à ce qui est fait dans Olan [Bellissard, 1995] c'est-à-dire que le premier travail est la séparation entre la description des modules (composants dans Olan) et la description des interactions entre eux. Elle y ajoute l'expression de la structure et la dynamique des groupes de travail.

1.1.3.2 Mise en œuvre des interactions

Nous établissons d'abord les jonctions possibles entre les informations correspondant à la coopération intergroupe. Afin de réaliser cette opération, nous nous appuyons sur le dictionnaire établi lors des deux précédentes étapes. Prenons un exemple pour illustrer leur utilité : dans le groupe *Réception*, nous avons l'événement E_{CN} *« colis à destination nationale prêt»* et dans *Expédition Nationale*, E_{ACN} *« colis à destination nationale en zone de stockage »*. Bien que ces informations ne portent pas le même nom, nous voyons bien qu'elles sont identiques.

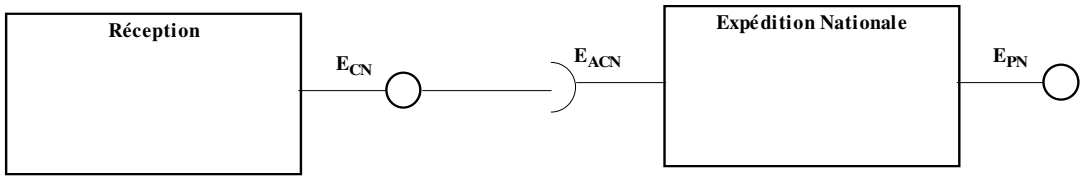


Figure 11 : Mise en place des connecteurs

Les connecteurs se placent entre composants composites, et entre modules (par soucis de clarté, nous ne montrerons pas ce dernier point ici) et ceci pour chaque type d'élément de coopération.

A l'étape précédente, nous avons relié entre eux les éléments de coopération lorsque ce lien était direct, mais il peut s'avérer que certaines informations nécessaires ne sont produites nulle part (le lien n'a pas encore été établi). En s'appuyant une fois de plus sur le dictionnaire, nous pouvons nous rendre compte que ces informations peuvent être construites par composition d'autres. Dans le cas où cela est impossible en l'état actuel de l'analyse, c'est que certains éléments de coopération ont été omis. Il est donc possible de revenir sur l'analyse afin de les ajouter. On peut aussi se trouver dans le cas où l'information nécessaire identifiée par l'analyse est une information théorique trop précise et irréalisable en pratique. Néanmoins, une reprise partielle de l'analyse devra nous permettre d'identifier (ou de construire) une nouvelle information peut être moins précise mais réalisable et suffisante.
Nous devons garder à l'esprit que nous travaillons dans le domaine de la réingénierie et que nous ne pouvons pas toujours obtenir ce que nous désirons. C'est l'une des contraintes et des limites de ce domaine. Si nous sommes si affirmatifs dans le fait que nous arriverons à produire les informations manquantes c'est qu'il ne faut pas oublier que nous intervenons sur des applications déjà existantes et donc que les informations identifiées sont déjà produites par un moyen ou un autre.

Les éléments de coopération directement disponibles ne permettent pas de réagir à des situations complexes. Ceci nous permettrait d'avoir un contenu sémantique plus riche et ainsi de ne réagir que dans certains cas bien déterminés augmentant d'autant l'efficacité. On se ramène au problème soulevé dans Darwin [Magee, 1996] avec l'expression des liaisons multiples. Il est nécessaire de prévoir un mécanisme permettant de détecter un schéma d'occurrences bien précis. Ce schéma peut être une suite d'événements distincts, une répétition dans un laps de temps précis, bref, toute situation envisageable à partir du moment où elle est algorithmiquement exprimable.

Nous proposons maintenant d'étendre ce principe et de le généraliser à la composition d'éléments de coopération [Koshel, 1998] [Lebastard, 1993]. Ceci nous permet de constituer des événements et des messages composés ainsi que des données synthétiques. Encore une fois, nous allons nous appuyer sur le dictionnaire établi par le concepteur de l'application au cours des étapes précédentes.
L'exemple suivant va illustrer la création d'événements composés. Par exemple, E_{SC} (en entrée du groupe *Service Commercial*) n'ayant pas encore été produit, nous allons examiner la manière de le créer. Cet événement sert à déclencher l'impression des factures pour les clients. Afin d'optimiser le traitement de ces factures, cet événement ne sera pas produit à chaque nouveau colis mais lorsqu'un nombre suffisant de colis à destination nationale ou régionale est à facturer. Cet événement pourra donc être constitué par une règle détectrice RD_1 [Aniorté, 1999] à partir des événements E_{CN} et E_{CR} produits par le groupe *Réception*.

Ci-joint un exemple de règle détectrice [Roose, 2000] :

```
DéfinirRègleDétectrice RD1 {
        // -- Début de l'algorithme de composition --
        String messageColisChercher = new String ( ) ;
        String evt = new String( ) ;
        while ( true ) { // une règle détectrice s'exécute à l'infinie
        evt = acceptRendezVous ( ) ;
        if (estÉvénement(evt, " ECN ","Réception") ) {
                produireÉlementComposé(Esc) ;
                }
        if (estMessage(evt, "Chercher Colis N", "Service Commercial") {
                messageColisChercher = lireBAL(« Chercher Colis N ») ; // Bien que l'on intervienne pas
sur le contenu du message, il est nécessaire de le retirer de la BAL.
                produireÉlementComposé(Esc) ;
                }
        } // Fin de la bouclie infinie
   } // -- Fin de l'algorithme de composition --
}
```

Par soucis de ne pas trop surcharger les explications, nous n'avons illustré la méthode qu'en utilisant des événements. Il va de soi qu'il est aussi possible de constituer de la même manière les autres types d'éléments de coopération manquants à partir de l'ensemble des informations disponibles.

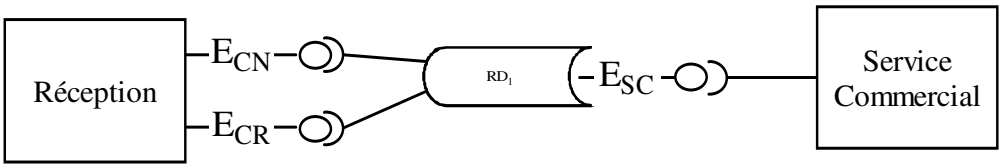


Figure 12 : Jonctions des éléments de coopération

Le schéma précédent (Figure 12) met en évidence un cas déjà identifié par [Magee, 1996] et le problème des liaisons multiples sur un seul port. La règle détectrice RD_1 (qui est un connecteur de première classe) se charge de la gestion des différents appels.

Si l'on se réfère à l'importante taxonomie des connecteurs [Mehta, 2000], la règle active joue ici le rôle d'un connecteur composite (*composition*) réalisant les rôles de :

- **Médiateur** (*arbitrator*) : la politique de gestion des informations est encapsulée dans ce rôle. Celui-ci rationalise la gestion des informations grâce à une connaissance globale de son environnement (ici, le composite).
- **Adaptateur** (*adaptor)* : bien que par simplicité, l'exemple ne mentionne que des événements, les messages et données peuvent également êtres inclus. Dans ce cas, nous sommes en présence du rôle d'adaptateur. En fonction des informations d'entrée éventuellement hétérogènes, il va réaliser les transformations nécessaires pour que l'information produite soit au format du port délégué.
- **Distributeur** (*distributor)* : les modules identifiés peuvent être localisés sur des sites distincts. Il est donc nécessaire de prendre en compte cet aspect de distribution.

1.1.3.3 Langage formel

La méthode permet la définition, la mise en correspondance et la création des éléments de coopération. A chaque étape, un langage graphique permet une représentation facilement compréhensible, mais ne permet aucune vérification. Aussi, un langage de spécification va permettre se s'assurer de la validité du résultat obtenu et ainsi vérifier que nous disposons de tout ce dont nous avons besoin par complétude des éléments de coopération.

En reprenant l'exemple simple de la Figure 11, nous avons :

- **DéfinirÉvénement** E_{ACN} **Pour Groupe** *Expédition Nationale* **De Groupe** *Réception* **comme** E_{CN}

Dans le cas où une règle active est utilisée (cf. Figure 12) nous avons :

- **DéfinirÉvénement** E_{CN} **De Groupe** *Réception* **Dans Groupe** *Expédition Nationale* **Pour Règle** RD_1
- **DéfinirÉvénement** E_{CR} **De Groupe** *Réception* **Dans Groupe** *Expédition Nationale* **Pour Règle** RD_1

Ces relations vont nous permettre de vérifier la complétude des éléments de coopération définis lors des étapes précédentes et donc de savoir si nous disposons de tous les éléments de coopération nécessaires. La preuve formelle du langage de spécification sur la base de relations d'ordre partiel est décrite dans [Roose, 2000].

1.1.4 La plate-forme coopérative en bref…

Comme déjà mentionné, je ne vais pas décrire ici la plate-forme qui ne traite pas directement du propos de ce mémoire, je vais simplement donner quelques idées quant à sa réalisation. Le lecteur pourra se référer à [Roose, 2000] pour obtenir une description complète.
La plate-forme doit permettre de faire circuler les éléments de coopération. Pour cela, il faut que le langage de spécification puisse nous donner un certain nombre d'indications concernant chacun des éléments de coopération. Ces informations nous permettront de pleinement qualifier un élément de coopération, c'est-à-dire de connaître pour chacun d'eux son type, sa source et sa ou ses destinations. On y trouvera en particulier :

- le groupe ou la règle qui émet l'information,
- le(s) groupe(s) ou le(s) modules ou le(s) règles destinataires de l'information,
- le type d'information : événement, message ou donnée,
- le nom de l'information.

Notre projet global va au delà de la simple proposition d'une méthode pour modéliser la coopération dans les applications distribuées. Nous gardons comme souci permanent que la méthode soit opérationnelle. Ainsi, une fois ces deux étapes réalisées, nous obtenons une dérivation du langage de spécification en règles ÉCA et détectrices. Celles-ci seront ensuite intégrées à une plate-forme coopérative qui permettra la mise en œuvre concrète de l'application.

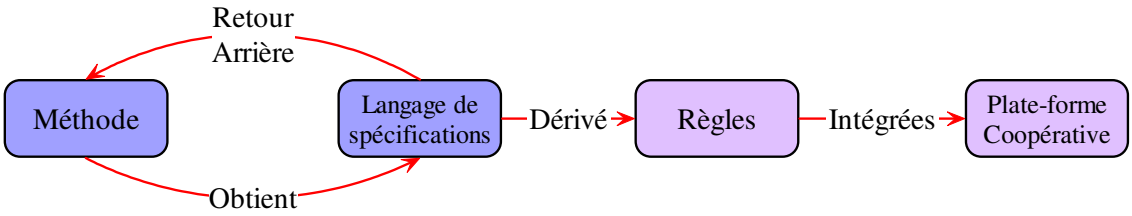


Figure 13 : Approche globale d'ELKAR

La Figure 13 résume la globalité des travaux liée à ELKAR qui propose une approche complète allant des aspects méthodologiques à l'implémentation en passant par la mise en œuvre et l'implémentation d'une plate-forme coopérative.

1.1.5 Synthèse d'ELKAR

Les travaux d'ELKAR ont apporté une contribution originale dans le domaine des méthodes de réingénierie et d'architectures logicielles (non détaillées ici).
ELKAR propose une méthode permettant d'étudier le système d'information de l'entreprise pour en sortir les éléments clés nécessaires à la mise en coopération, c'est-à-dire permettant la suppression de tout ou partie des opérateurs externes qui réalisaient cette opération par des copies, re-saisies, impressions, etc.
Cette étude situe ces travaux essentiellement au niveau de la première et de la troisième couche de la Figure 2, au niveau « acquisition des informations contextuelles », et « adaptation de l'application ». En effet, les étapes d'inventaire de la méthode agissent comme une phase d'acquisition du contexte. La seconde phase de la méthode chargée de la mise en œuvre des interactions agit également au niveau du système d'information en adaptant ou en composant les données ainsi qu'en réalisant les migrations vers les entités destinatrices. L'organisation en groupes de travail formés de modules a permis également, d'un point de vue conceptuel, la mise en œuvre d'une composition. La mise en œuvre des interactions a été confiée à des connecteurs appelées Règles Actives réalisant les 3 rôles précédemment cités. Notons bien que nous ne sommes pas ici dans une adaptation fonctionnelle.

1.2 Limites des travaux et perspectives

Contrairement à la plupart des approches, il n'existe pas dans ELKAR de connecteurs spécifiques à la distribution permettant de spécifier le rôle des connecteurs de distribution (RPC, Client/serveur, etc.).
Dans le domaine de la réingénierie, nous avons été confrontés à la manipulation de modules, vus et manipulés comme des boîtes noires, pouvant être de nature logicielle ou matérielle. La mise en coopération de ces derniers n'est pas forcément faisable dans tous les cas. Limité par le domaine de la réingénierie, nous ne sommes intervenus qu'en nous basant sur le système d'information des modules existant afin de capturer le contexte et de produire de nouvelles données manquantes, auparavant produites par les opérateurs externes.

C'est par une démarche analogue que nous nous sommes par la suite intéressés au domaine des **COTS Products**. La conception d'applications à base de COTS peut être vue comme une activité de réutilisation. En effet, l'objectif d'un assemblage est de partir de produits existants (avec leurs contraintes : environnement d'exécution, intergiciels, etc.) et de mettre en œuvre les relations (wrappers - *composants standards permettant l'intégration comme ODBC* - glue code, etc.) nécessaires à l'assemblage en vue de la réalisation d'une application complète.

1.3 Contexte environnemental : application aux COTS Products

Depuis plusieurs années, les développeurs informatiques sont tentés d'utiliser des composants logiciels déjà existants. Ce phénomène n'est pas nouveau, mais il prend de plus en plus d'ampleur. L'intégration de ces composants logiciels du commerce communément appelés COTS (*Components Off The Shelf*) dans une application globale nécessite l'utilisation d'un processus de développement mettant en œuvre une démarche permettant de les sélectionner, de les évaluer, de les comparer pour les départager mais également pour établir les moyens et les compétences nécessaires à leur intégration.
L'obtention des informations relatives à chaque COTS est souvent à l'origine des difficultés rencontrées pour la mise en œuvre d'un développement à base de COTS. La réalisation de prototypes doit permettre de les obtenir car elles sont nécessaires pour les évaluations et sélections dans le processus de développement [NCube, 1999]. Pourtant, leur réalisation nécessite elle-aussi des informations qui ne sont

pas facilement accessibles comme celles traitant des environnements d'utilisation des produits et de leur intégration ainsi que la connaissance des scénarios de dysfonctionnement. Toutes ces données sont des éléments aussi importants pour la réalisation du prototype que pour le processus de développement. C'est donc la base de plusieurs connaissances autour des COTS qui doit permettre de faciliter leurs utilisations dans les développements d'applications logicielles.

1.3.1 Qu'est-ce qu'un COTS ?

La définition admise par tous est qu'un COTS (*Commercial Off The Shelf)* est un composant issu du marché appelé composant sur étagère. Derrière cette définition très large se rattache un certain nombre d'attentes. Ainsi, un COTS est un produit logiciel existant en de multiples copies, dont le code source est disponible ou non, vendu, loué ou fourni gratuitement par un vendeur et ayant des mises à jour périodiques s'accompagnant d'un accroissement des fonctionnalités fournies et d'une obsolescence de certaines autres. Cette définition est suffisamment ouverte pour que beaucoup de produits logiciels puissent être considérés comme des COTS. Aussi, nous proposons une définition plus restrictive mais plus cohérente avec les attentes que l'on a d'un COTS Product : **un COTS Product est un produit ayant la capacité de proposer l'un de ses services (ou plus) comme partie indispensable au fonctionnement d'une autre application.**
Le terme générique et tronqué de COTS regroupe en réalisé cinq problématiques qui sont autant de domaines de recherche distincts : les *COTS System, COTS Based Product, COTS Software,* ***COTS Product***, (ce qui nous concerne), *COTS Component*. La terminologie des *COTS System* [Carney, 1997] [Morisio, 2000] considère les COTS comme un ensemble logiciel utilisant des COTS. Dans ce cas, ce n'est pas le produit de manière unitaire que l'on étudie mais le système dans son intégralité. Le *COTS Software* permet d'identifier (par sélection, comparaison) l'ensemble des produits potentiellement utiles à la réalisation d'une application à base de COTS. Le *COTS Software* est de notre point de vue, une vision plus restrictive que le *COTS System*. Si ce dernier correspond à une vision d'ensemble comprenant l'application à base de COTS, le *COTS Software* ne s'intéresse qu'aux différents COTS indépendamment de l'application à développer.
Une autre restriction dans la vision des COTS correspond à la troisième dénomination *COTS Product* qui s'attarde sur la définition d'un produit et les problématiques relatives à son intégration dans l'application. La relation, la comparaison et les contraintes entre produits ne sont pas abordées. Seules les fonctionnalités ou les questions techniques pour l'intégration sont discriminantes. C'est dans ce cadre que nous nous situons. Pour finir, les *COTS Components* rassemblent une catégorie de COTS autour des techniques de composants logiciels. C'est une vue particulière des *COTS Product* autour de composants prévus pour être intégrés dès leur conception.

1.3.2 Problèmes liés à l'utilisation des COTS

1.3.2.1 Évaluation

Le processus de développement des applications à base de COTS généralement admis (Figure 14) possède une phase critique appelée « component qualification » dont l'objectif est d'étudier les COTS précédemment sélectionnés.

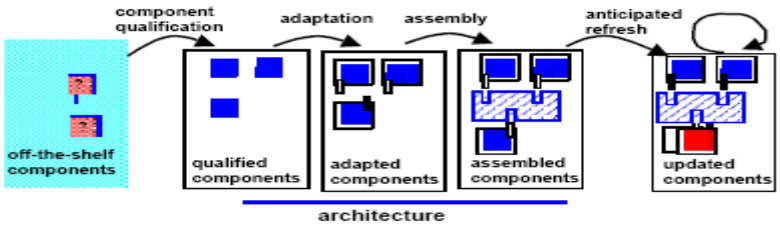


Figure 14 : Processus de développement à base de COTS [Carney, 1997]

A ce stade d'avancement du processus, plusieurs COTS sont déjà sélectionnés et identifiés comme concurrents. Ils doivent ici être départagés afin que quelques candidats potentiels soient éliminés. Le rôle de cette évaluation doit permettre d'identifier les fonctionnalités rendues, la faisabilité de l'intégration, les compatibilités entre produits, l'architecture à mettre en place, les compétences nécessaires pour l'intégration et les coûts (homme/mois) de l'application à base de COTS. En fait, toutes les informations permettant de réaliser le bon choix en réduisant les risques sur des sélections incorrectes doivent être récoltées.

Pour cela, l'évaluation des COTS passe bien souvent par la réalisation de bancs d'essais. Beaucoup de types d'évaluation exigent l'utilisation réelle des produits de COTS dans des prototypes, des démonstrateurs, ou des pilotes [Obendorf, 1997] [Obendorf, 2000] [Carney, 2003]. Ils sont essentiels pour estimer les risques et les coûts d'intégration.
L'objectif est de permettre le bon choix sur les COTS déjà sélectionnés dans une phase amont, mais aussi de réduire et/ou de minimiser les risques d'une mauvaise sélection. Nous avons présenté certains critères, mais bien sûr, cette liste n'est pas exhaustive. C'est surtout la quantité et la qualité des informations recueillies qui permettront une comparaison entre les produits concurrents et donc une sélection optimale des COTS.

1.3.2.2 Informations disponibles

Les informations mises à disposition des développeurs ne sont pas suffisantes pour réaliser correctement une évaluation. Leur disponibilité est d'ailleurs un critère de sélection pour le produit.
Lorsque quelqu'un veut sélectionner un COTS, les vendeurs de ces produits fournissent une liste classée par métier, domaine d'activité, plate-forme ou langage de programmation. Cette liste n'est pas exhaustive, mais montre que les informations fournies sont très génériques.
Parmi ces informations, nous pouvons trouver les services rendus par le COTS, mais aussi quelquefois des exemples de codes sources qui montrent comment l'intégrer. Généralement, le service principal du COTS est documenté, en revanche les services secondaires doivent être évalués par les développeurs en charge de l'intégration. En pratique, l'utilisateur de COTS doit évaluer les différents services et le contexte d'intégration avec les coûts et les risques associés.

1.3.3 Problèmes liés à la réalisation de l'intégration

Pour l'instant, le manque d'information sur les COTS et en particulier sur la réalisation d'un prototype permettant d'en faire une évaluation est le principal problème de leur intégration. Ceci augmente le temps nécessaire pour développer le prototype et de fait les coûts d'intégration, d'où l'importance de la veille technologique qui permet de récupérer les informations, de tester les composants avant leur utilisation dans le processus de développement. La connaissance de ces produits et de la manière de les intégrer devient un avantage sérieux lors du développement de l'application à base de COTS, en particulier pour la réalisation des prototypes.
Nous avons donc besoin d'un ensemble d'informations organisées, classifiées et pouvant être diffusées auprès des utilisateurs de COTS. Un Système d'Information (SI) peut relier les fournisseurs d'informations (développeurs de COTS) et les clients (développeurs d'applications à base de COTS). A travers ce système d'information, nous proposons d'aider ces derniers dans la réalisation de leurs prototypes en leur apportant des informations sur les moyens nécessaires à leurs développements.

1.3.4 Principes de la démarche

1.3.4.1 Informations disponibles

Les concepteurs et les distributeurs de COTS possèdent en toute logique les informations indispensables pour la réalisation de l'intégration par les développeurs. Ces deux acteurs sont des fournisseurs d'informations (regroupés par la suite sous le terme de « FourCo » - ***Four****nisseurs de* ***CO****TS*) et regroupent les concepteurs et les distributeurs de COTS. D'autres acteurs que nous appelons la « communauté des COTS » ont, du fait d'expériences passées dans leurs intégrations, des informations indispensables pour les développeurs de l'application à base de COTS (historiques sur des difficultés rencontrées ou les configurations de bon fonctionnement).
Si ces derniers (« FourCo » et « communauté de COTS ») sont plutôt fournisseurs d'informations, les développeurs d'applications à base de COTS sont plutôt les clients et les demandeurs d'informations comme nous le représentons sur la figure ci-dessous.

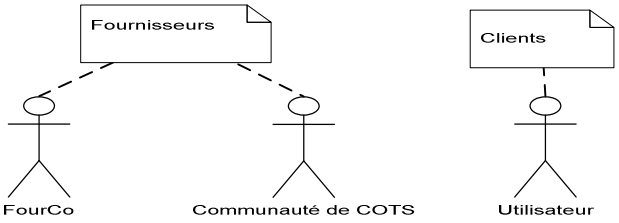


Figure 15 : Les acteurs de COTS

Cette relation a été la base de notre système. Elle établit un lien permettant l'échange d'informations entre les utilisateurs et les fournisseurs des COTS, se traduisant par la mise en œuvre de mécanismes d'échanges (Figure 16) :

- la diffusion d'informations grâce à l'enregistrement dans une base de données du Système d'Information,
- la recherche et la récupération des informations dans le Système d'Information,
- la demande d'informations complémentaires afin qu'elles soient ajoutées dans le Système d'Information,
- le traitement de ces demandes par les fournisseurs afin d'ajouter les données manquantes dans le Système d'Information.

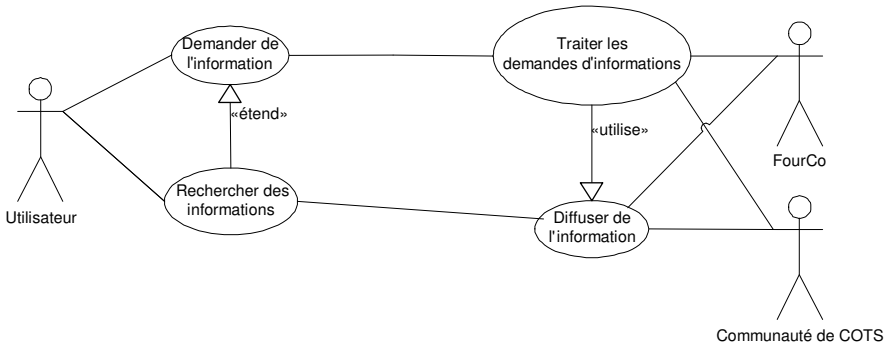


Figure 16 : Echange d'information entre acteurs

A travers ces échanges et le Système d'Information, il est possible d'obtenir une base de connaissance des COTS, avec la possibilité :

- d'extraire des données pour que les « utilisateurs de COTS » puissent réaliser des traitements comme par exemple des comparaisons entre produits ou pour l'utilisation d'outils de mesure de l'effort à fournir pour leur intégration comme COCOMO ou COCOTS [Abts, 1998],
- de l'enrichir par des informations complémentaires comme par exemple des scénarios de dysfonctionnement ou des résultats de tests d'évaluation.
- et surtout d'identifier les moyens nécessaires à l'intégration des COTS dans un prototype.

C'est à partir de ces informations que nous avons facilité la réalisation des prototypes que nous décrivons ci-dessous.

1.3.4.2 Un prototype pour un prototype

L'utilisation de COTS (au sens *COTS Product* dorénavant) dans un processus de développement implique de mettre en œuvre une relation client/fournisseur forte de manière à favoriser la publication des caractéristiques de chaque COTS qui seront utiles et exploitables par les clients.
Selon la norme ISO/IEC 9127 [ISO, 2004], il existe près de 600 conditions de risques ou combinaisons de paramètres d'entrée nécessaires à une entité logicielle comme un COTS. Ce volume important explique la difficulté du recueil d'informations. De plus, cette quantité importante et la faible structuration de ces dernières provoquent un éparpillement et donc un manque d'accessibilité.
Aussi, les développeurs à base de COTS se tournent fréquemment vers la réalisation de prototypes intégrant un ou plusieurs COTS permettant de :

- faire des expérimentations en testant la véracité des explications concernant le COTS au regard de ce qu'il produit/réalise ;

- recueillir des informations sur ce dernier en termes qualitatifs et quantitatifs, mais également en termes de compatibilité, d'environnement d'exécution, etc. ;
- valider ces informations, en s'assurant par des jeux d'essai de la validité des traitements réalisés.

Toutes ces informations obtenues permettent l'évaluation du COTS et sont exploitées afin de faciliter la phase de **sélection**. Néanmoins, la réalisation de prototypes se heurte à un certain nombre de difficultés. La première, et non la moindre, est d'obtenir les informations nécessaires à l'intégration des COTS. En effet, ces informations ne sont pas forcément disponibles.
La réalisation d'un prototype permet de connaître la méthode d'intégration spécifique à chaque COTS.
Si nous avons souhaité faciliter le recueil des informations précédemment citées, il a été nécessaire de mettre en œuvre une relation « Client / Fournisseur » qui permette au développeur d'accéder aux données nécessaires à la réalisation de son prototype.
Nous avons proposé d'organiser ces données dans un recueil en trois phases distinctes :

- **Utilisation des COTS** : informations permettant d'identifier l'ensemble des logiciels nécessaires au fonctionnement du COTS et du service identifié dans ce dernier. Par extension à l'ensemble des COTS, nous obtenons un graphe de dépendances. De la même manière, cette étape permet l'identification de l'ensemble des logiciels n'autorisant pas un fonctionnement optimal, identifiant ainsi les incompatibilités plus ou moins inévitables.
- **Intégration des COTS** : nous identifions ici l'ensemble des logiciels nécessaires au développement de l'intégration, c'est-à-dire qui permettent l'intégration des COTS dans l'application. Nous retrouvons les outils de développement, des logiciels annexes ou complémentaires ainsi que les bibliothèques. C'est également ici que sont identifiés les langages de programmation qu'il faudra manipuler et que sont référencés les exemples de codes permettant de faciliter l'intégration. A un second niveau, c'est à cette étape que sont mises en évidence les configurations ne permettant pas l'intégration de tel ou tel COTS.
- **Certification des informations** : cette dernière étape permet d'identifier l'auteur des informations. En effet, il est cohérent de penser que le concepteur du COTS lui-même fournit les informations les plus fiables. Ainsi, le Système d'Information proposé permettra de faciliter l'accès aux informations, la mise en relation des clients et des fournisseurs, les traitements automatisés et de vérifier la validité des données.

Afin d'éviter toute ambiguïté, il est nécessaire de préciser que nous n'avons pas proposé des processus sur l'évaluation et l'estimation des risques et des coûts. Nous n'avons pas non plus proposé un système de pilotage pour le traitement des choix de COTS ni pour le traitement des informations recueillies pendant les phases du processus de développement. Nous sommes partis de l'hypothèse que le choix des COTS a été réalisé en amont.

1.3.5 Base de donnée

La base de données a été établie à partir d'un certain nombre d'informations :

- un identifiant unique pour les COTS (au même titre que les références uniques des cartes réseau). Cet identifiant peut être l'identifiant international de l'entreprise fournissant le COTS auquel il faudrait ajouter un nom afin d'assurer l'unicité.
- éventuellement une référence à d'autres COTS ou applications (exemple d'un pré-requis ou d'une incompatibilité avec un autre produit).
- des informations sur l'utilisation et l'intégration du COTS.

Ces informations ont été regroupées pour définir de manière unique le COTS sous la forme d'une carte d'identité ICOTS (*Identity Card for COTS*) [Michel, 2005] permettant une caractérisation unique de ce dernier.

1.3.5.1 Environnements et scénarios

Nous avons proposé la notion d'environnement et de scénario. Pour décrire un prototype ou un COTS, nous avons identifié trois environnements : l'environnement d'exécution, l'environnement d'intégration, l'environnement de développement.

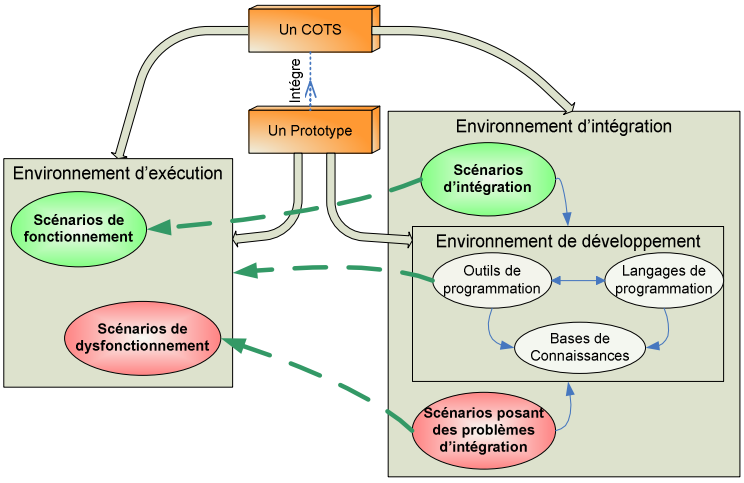


Figure 17 : Trois environnements

Ces informations sont fournies par les concepteurs et fournisseurs de COTS complétées par les acteurs appelés les clients durant le processus de développement d'une application à base de COTS. En fait, elles doivent toutes être rattachées aux produits et donc à la carte d'identité ICOTS. C'est à partir de cette carte d'identité que nous affectons les informations concernant les environnements d'intégration et d'exécution. De ce fait, lorsque les fournisseurs de COTS ont besoin de décrire un environnement pour leur produit ils peuvent faire référence à d'autres COTS et donc à une autre carte d'identité qui elle-même décrira son propre environnement. Ainsi, par références successives, il est possible d'obtenir la liste de tous les éléments logiciels nécessaires à un environnement donné et donc de détecter des incompatibilités. Par exemple si dans un prototype nous assemblons les COTS A et B, le premier a besoin pour son exécution d'un produit « Y » qui utilise un produit « Z » mais incompatible avec l'utilisation du produit « X ». Le second a besoin d'un produit « V » utilisant le produit « X ». Dans cet exemple nous pourrons conclure que l'utilisation simultanée du COTS A et B n'est pas possible : c'est donc une incompatibilité que le développeur devra résoudre. Pour cela, il peut choisir d'utiliser l'un ou l'autre des COTS et de développer les services non remplis par le COTS manquant, ou alors de remplacer l'un d'entre eux et revenir ainsi à la phase de sélection.

Un ensemble de rapports associés à chaque scénario est produit, aidant en cela le développeur à maîtriser les multiples informations issues des différents produits qui vont composer son application. Certaines l'aideront dans la réalisation de l'intégration (exemple du scénario d'intégration) alors que d'autres l'avertiront sur des configurations d'exécution ou de développement ayant déjà causé des problèmes à d'autres utilisateurs de COTS. Il est évident que cela ne peut que favoriser l'utilisation des prototypes pour évaluer, qualifier et estimer les coûts et les risques.

1.3.5.2 Environnement de saisie

L'obtention des informations est primordiale pour le bon déroulement du processus de développement de l'application basée COTS. Pour réduire les difficultés sur le sujet [Leung, 2002] propose de réaliser des prototypes qu'il juge indispensables à plusieurs phases du processus de développement pour le recueil et la validation des informations. Le prototype doit permettre d'étudier l'intégration d'un ou de plusieurs COTS cohabitant ensemble. De cette étude les développeurs vont obtenir et valider des informations indispensables pour chaque phase du processus de développement de l'application à base de COTS. L'utilisation de ces prototypes dans des expérimentations est une démarche qui garantit l'accès aux informations manquantes et difficilement accessibles. Indépendamment de savoir si cela représente le seul moyen d'obtenir ces informations, le fait est que l'expérimentation est souvent le meilleur moyen pour s'assurer de la validité des informations recueillies. L'application présentée ci-après va mettre en pratique les apports proposés précédemment.

1.3.5.3 Présentation

L'application SI-COTS [Michel, 2006] permet l'enregistrement de la carte d'identité avec les informations associées à la fois pour le COTS Product mais également pour l'application à base de COTS. Cet outil aide également à la relation client/fournisseur en mettant l'ICOTS à disposition et en permettant le téléchargement d'ICOTS supplémentaires.

SI-COTS permet également l'identification des services des COTS Products utilisés dans l'application à base de COTS, et notamment les services intégrables (ceux qui ne le sont pas ne sont d'aucune utilité). De plus, elle propose pour chaque COTS et service fournis par chacun, l'identification du contexte des environnements d'exécution et d'intégration, dans le but d'aider le développeur de prototypes, notamment pour la saisie des pré-requis, des outils de développement, des exemples de codes et des dysfonctionnements identifiés.
A titre d'illustration, nous allons prendre deux COTS Products bien connus, *MS Outlook* et *MS Project* afin d'utiliser respectivement les services de gestion des contacts et des tâches.
L'utilisation de SI-COTS se réalise en plusieurs étapes. La première est réalisée par le fournisseur de COTS, c'est la création de la fiche ICOTS (ici de MS Outlook) [Michel, 2004]. Les étapes suivantes sont faites par l'utilisateur et permettent de compléter la fiche ICOTS dans le prototype et de réaliser l'association des services. La dernière étape consiste à la production automatique des rapports permettant de connaître des types de contexte « matériel » et « environnement » identifiés au cours des scénarios d'intégration, d'exécution et de dysfonctionnement (Figure 19).

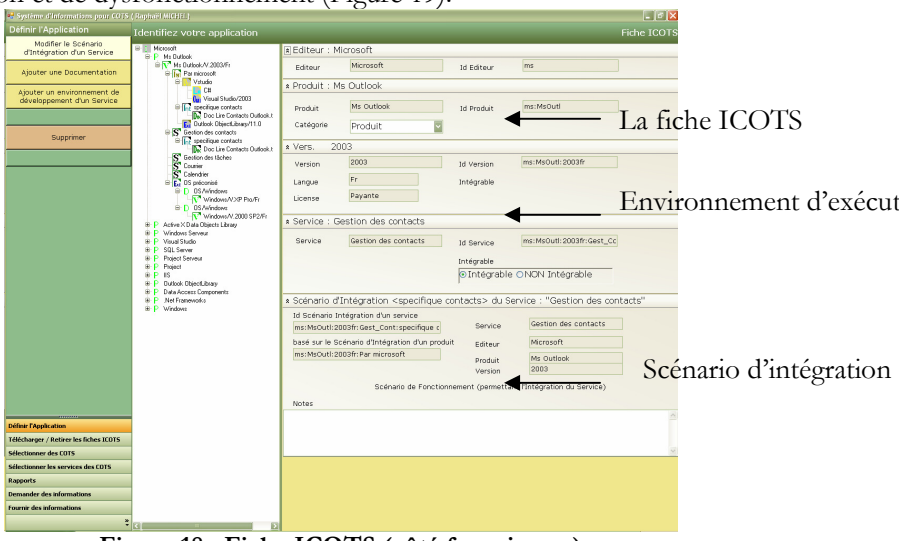


Figure 18 : Fiche ICOTS (côté fournisseur)

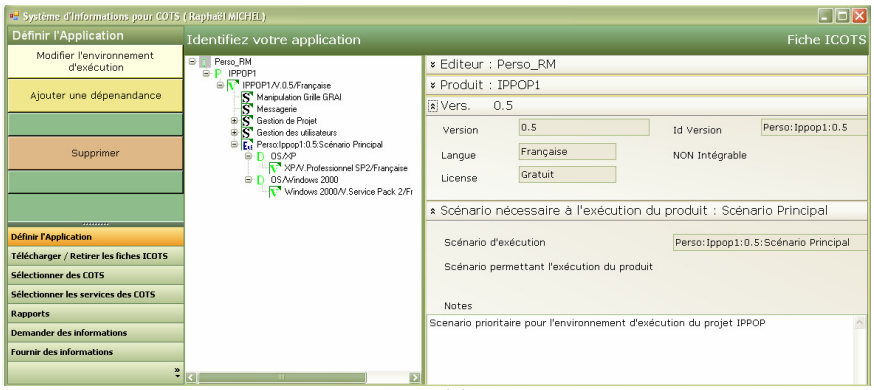


Figure 19 : Fiche ICOTS (côté utilisateur)

1.3.6 Synthèse

Nous nous sommes positionnés dans le processus de développement à la phase « component qualification » (Figure 14) et en particulier sur l'une des problématiques pour l'évaluation des COTS : la réalisation de prototype indispensable à cette phase [Leung, 2002]. La difficulté d'obtenir des informations pour sa réalisation aura une incidence sur la qualité des évaluations et donc sur les risques encourus lors de l'assemblage du produit.
Nous avons établi une relation client/fournisseur entre des développeurs d'application à base de COTS et des acteurs pouvant fournir des informations pour la réalisation des prototypes. Ces informations sont conservées dans le Système d'Information. Elles correspondent aux éléments nécessaires définissant le contexte matériel et environnemental du prototype et du COTS. Nous considérons dans ce dernier les

outils, langages et documentations permettant de réaliser l'intégration de chaque service du COTS dans un prototype.
A partir de ces informations contextuelles, le Système d'Information est en mesure de fournir des rapports de synthèse permettant de détecter des situations de conflit ou de dysfonctionnement. Dans les exemples que nous avons décrits, le développeur détecte rapidement ces problèmes avant même le démarrage des développements alors qu'ils sont aujourd'hui découverts lors de la réalisation du prototype ou pire encore pendant la phase d'assemblage. C'est donc un gain de temps non négligeable qui peut être mis à profit pour affiner les évaluations et réduire ainsi les risques et les coûts. De plus, nous traitons aussi l'une des difficultés qui était l'obtention des informations mises à disposition par le Système d'Information.

Le prototype développé en .NET pour plate-forme Windows est téléchargeable à l'URL suivante (http://www.iutbayonne.univ-pau.fr/~roose/pub/recherche/SICOTS). Il a donné lieu à une présentation complète de près d'une heure au cours du 3ème workshop de Donald Firesmith, chantre du COTS, lors de la conférence ICCBSS de 2008 à Madrid.
Il permet l'amélioration du recueil des informations contextuelles de chacun des COTS et facilite l'accès aux informations pour l'intégration ainsi qu'un gain de temps important. Il favorise les évaluations par la production de rapports et permet une capitalisation des informations via la communauté des utilisateurs de COTS Products. Le Système d'Information développé et implémenté dans l'application SI-COTS propose une aide aux développeurs de prototypes d'applications à base de COTS Products. Cette approche par l'intermédiaire d'un SI a abouti à la définition d'une carte d'identité (ICOTS) ainsi qu'à la production d'informations d'aide à l'intégration de COTS Products. Il est également important de capitaliser l'expérience acquise au gré des développements, ce que permet l'application développée. Néanmoins, l'approche possède ses limites en termes d'interopérabilité entre ICOTS qui nécessite une normalisation des échanges et des données. De plus, nous n'avons abordé l'approche des COTS Products que par l'intégration, ce qui ne couvre bien évidemment pas le spectre très large des phases amont de sélection, évaluation, qualification et d'estimation des risques et des coûts.

L'approche à partir d'applications et de composants existants possèdent un certain nombre de limites. En effet, n'ayant pas été conçus a priori pour donner des informations sur leur contexte, ni pour les prendre en compte, seule une approche statique au niveau du système d'information peut être utilisée. Par approche statique nous entendons le fait que nous ne modifions pas la structure d'une quelconque application ou COTS. Nous n'intervenons qu'au niveau du SI en capturant ou injectant des informations, voire en créant de nouvelles informations.

Nous souhaitons maintenant aller vers plus de souplesse et permettre non seulement de prendre en compte le contexte statique, mais également le contexte dynamique permettant aux applications de s'adapter de manière à offrir le meilleur service possible en fonction du contexte d'exécution.

Chapitre 3 Outils d'adaptation permettant la prise en compte du contexte

Jusqu'à présent, nous nous sommes exclusivement basés sur le système d'information afin de capturer le contexte et ainsi permettre une réaction de l'application en se basant exclusivement sur les données (prise en compte non fonctionnelle). Nous n'avons pas encore réellement abordé la troisième couche du modèle des applications sensibles au contexte permettant l'adaptation de l'application à ce dernier.
Afin d'illustrer nos propos, nous avons choisi le domaine des **applications multimédia distribuées sur Internet** qui offre un champ d'application intéressant de par les contraintes auxquelles elles sont soumises : contexte utilisateur mouvant (ajout/retrait d'utilisateurs, choix de fonctionnalités évolutif), contexte matériel également mouvant (débit, délais, gigue du réseau, affichage), tout comme le contexte temporel (respect de contraintes de synchronisation par exemple).
Dans ce chapitre nous proposons un modèle de composants permettant la réalisation d'assemblages dynamiques. Ce modèle de composant supervisable, appelé **Osagaia**, permet d'une part de notifier des informations contextuelles sur son état, mais également de recevoir des commandes agissant sur son cycle de vie.
Dans un second temps nous proposons également un modèle de connecteur de première classe (comme dans UML 2.0), appelé **Korrontea** (**Conduits)**, capable lui de renseigner sur l'état des communications entre composants tout en étant supervisable. Il permet en outre d'assurer différentes politiques de communication comme la synchronisation des données.
Enfin, nous décrivons une plate-forme d'exécution, appelée **Kalinahia**, qui possède une connaissance du déploiement des composants et permet ainsi, selon des heuristiques, d'assurer les déploiements et reconfigurations des composants et des connexions.

3.1 Plate-forme d'exécution

Il existe essentiellement deux approches permettant d'appréhender le contexte de manière dynamique. L'une est totalement autonome à savoir que chaque composant s'auto suffit et ne réagit au contexte que localement. Cette manière de faire, élégante sur le papier, possède néanmoins un défaut important à nos yeux, le manque de vision globale. Ainsi, la reconfiguration d'un composant peut engendrer des effets de bord désastreux. Nous avons préféré prendre une option plus centralisée en mettant en œuvre un monitoring local (des composants) et global (de l'application). Cette approche nécessite la mise en œuvre d'une plate-forme permettant à la fois la remontée d'informations locales capturées sur chaque composant, et d'informations contextuelles liées aux transferts de données puis assurant une supervision des composants lorsqu'un changement de contexte implique une reconfiguration de l'application. Associés à cette plate-forme, nous proposons également un modèle de composants et un modèle de connecteurs permettant de donner des informations sur leurs états et la supervision via l'envoi de commandes par la plate-forme.

Il est maintenant communément admis que le développement d'applications à base de composants logiciels permet de réduire à la fois les temps de réalisation mais également (et c'est une conséquence directe) les coûts. Ce temps est d'autant plus réduit si le concepteur peut se focaliser exclusivement sur les aspects fonctionnels. Jusqu'alors, dans le premier chapitre « Processus de réingénierie » nous n'avions pas la possibilité d'être intrusif au niveau des applications, nous étions liés à la contrainte de l'existant, ce qui explique en grande partie le fait de ne pouvoir adresser la troisième couche des applications sensibles au contexte Figure 1.
C'est ce qui nous guide dans l'utilisation d'une plate-forme logicielle servant de support d'exécution à l'application. L'objectif de cette partie est d'en proposer un modèle. La plate-forme est conçue comme un superviseur réparti sur les différents postes de l'application. Elle reçoit des états reflétant le

fonctionnement des composants, des connecteurs et de la structure de l'application et retourne aux composants et connecteurs des commandes (cf. Figure 20) afin d'optimiser la QdS grâce à une adaptation dynamique de l'application au contexte global en modifiant dynamiquement :

- le comportement des composants (pour ceux qui sont adaptables) ;
- la composition de l'application en supprimant, ajoutant ou remplaçant des composants ;
- les connections entre composants ;
- l'ordonnancement des composants lorsque cela est possible ;
- le déploiement des composants.

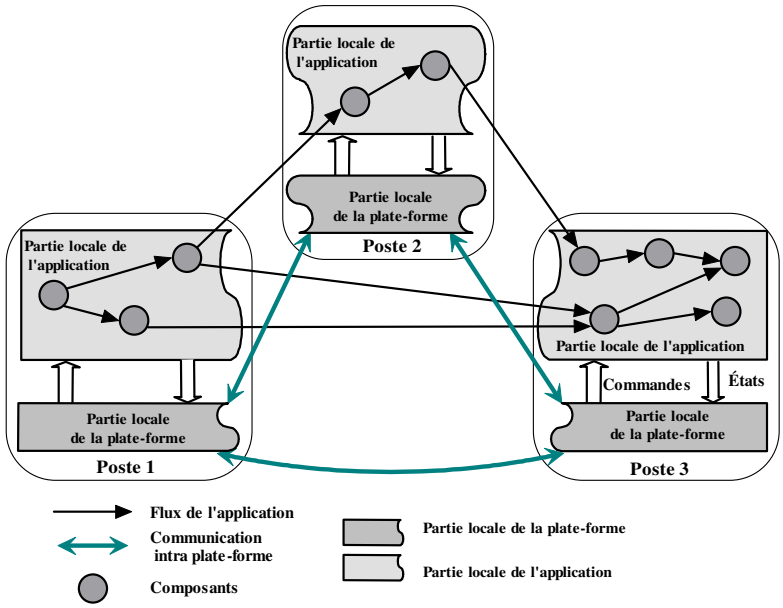


Figure 20 : Supervision de l'application par la plate-forme

Le rôle de la plate-forme est de récupérer les informations contextuelles temporelles, utilisateurs, matériel comme les connexions réseau, le type des terminaux informatiques, les préférences des utilisateurs, etc. Ces informations sont ensuite utilisées lors de l'évaluation du contexte afin d'assurer in-fine une qualité de service « suffisante » de l'application à l'utilisateur. Le résultat est la mise en œuvre de la politique d'adaptation (est-ce qu'il faut reconfigurer ? si, oui, comment ?) lorsque la qualité de service n'est pas (ou n'est plus) suffisante ou dans le cas contraire, s'il est possible de l'améliorer pour tendre vers les souhaits des utilisateurs.

3.2 Groupes et Sous-Groupes

La particularité des applications multimédia distribuées (ou AMD) est le point de départ de la structuration de nos applications. Dans le cas des AMD, nous avons montré dans [Dalmau, 2004] qu'une fonctionnalité ne peut être prise en compte que si elle peut être observée et évaluée par l'utilisateur.
Nous avons ainsi proposé une structuration de l'application en deux niveaux appelés Groupe et Sous-Groupes [Laplace, 2005].

3.2.1 Groupes, Sous-Groupes : définitions

Le **Groupe représente le service rendu à un utilisateur,** ou selon une terminologie plus commune, un cas d'utilisation pour un acteur**.** Pour une application donnée, le concepteur de l'application définit différents Groupes. Ainsi pour une application de vidéoconférence, deux Groupes différents décriront d'une part les intervenants de la conférence, *les locuteurs*, et d'autre part *les téléspectateurs*. Un Groupe spécifie ainsi un type d'utilisateur. Chacun bénéficie d'une instance du Groupe qui l'intéresse. Notre modèle permet donc à un acteur de rejoindre ou de quitter l'application en cours d'exécution grâce à une instanciation ou une suppression de son instance de Groupe.

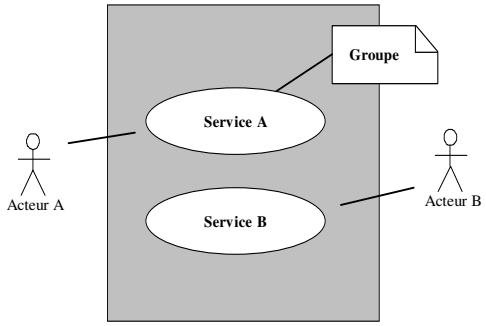


Figure 21 : Cas d'utilisation d'un Groupe

Le **Sous-Groupe représente une fonctionnalité**, et une seule, du service rendu à un utilisateur donc une fonctionnalité du Groupe de cet utilisateur. Un Groupe est donc composé de différents Sous-Groupes. Ainsi pour la vidéoconférence, le Groupe *téléspectateur* contient deux Sous-Groupes, l'un fournissant les images et l'autre fournissant le son. La composition d'une instance de Groupe en termes de Sous-Groupes n'est pas unique. En fonction du contexte, elle pourra varier de manière à proposer différentes qualités de service. Ainsi, il est possible de dégrader un service en enlevant des fonctionnalités pour continuer à assurer la QdS la plus satisfaisante possible. Dans notre exemple, le Groupe Locuteur peut être composé uniquement des Sous-Groupes *Son* et *Image* chargés respectivement de la gestion (acquisition, traitements, restitution) du son et des images ou fournir également une fonctionnalité de suivi automatique par la caméra permettant à l'orateur de se déplacer.

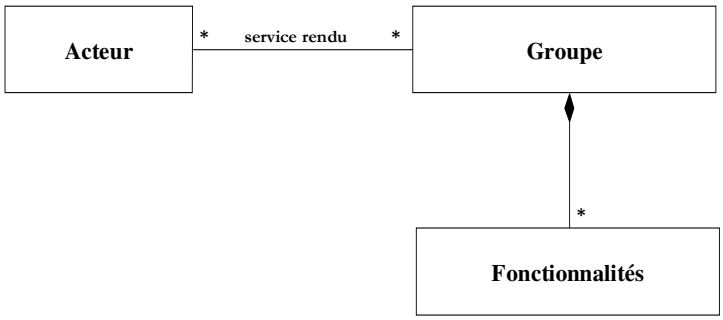


Figure 22 : Composition d'un Groupe

3.2.2 Propriétés

Un groupe est structurellement indépendant, il inclut l'ensemble des fonctionnalités nécessaire au rendu du service vis-à-vis de l'acteur. De manière pragmatique, il est évident qu'un groupe va se résumer à un ensemble de composants avec la particularité que même s'ils sont structurellement indépendants et n'échangent pas d'informations avec les autres, ils peuvent néanmoins avoir certains composants communs avec d'autres groupes et donc par extension un certain nombre de flux de données. C'est particulièrement vérifié avec des composants matériels qui doivent être partagés entre différents utilisateurs et qui seront donc dupliqués dans la modélisation pour apparaître dans différents Groupes bien qu'étant uniques d'un point de vue physique.

De façon analogue, un Sous-Groupe est chargé de fournir une fonctionnalité à un utilisateur. De manière similaire aux groupes, il ne partage par d'informations avec les autres Sous-Groupes d'un groupe tout en pouvant partager des composants et donc par extension des flux de données. Ainsi par exemple (Figure 23), le composant représentant une caméra dans le Groupe Locuteur chargée de la capture de la vidéo appartient également aux Sous-Groupes permettant la transmission de l'image pour que le locuteur surveille ce que les spectateurs voient. Ce composant appartient également au Sous-Groupe chargé du suivi automatique du locuteur (fonctionnalité appelée *tracking*).

En d'autres termes, les Groupes et Sous-Groupes ne sont pas des mini applications que l'on assemble pour former l'application finale mais constituent les différentes facettes de l'application en tant que fournisseur de services.

3.3 Composants

D'un point de vue pratique, les Groupes et Sous-Groupes sont réalisés à partir de composants. Néanmoins, par composant nous n'entendons pas uniquement des composants logiciels. Un **composant de l'application,** noté $\mathbf{C_i}$ est une entité logicielle, matérielle ou humaine qui participe à la réalisation d'une fonctionnalité. La granularité d'un composant dépend de son concepteur et de son utilisation : un téléphone portable est un composant permettant la visualisation des images sur un écran et la réception du son grâce à un haut-parleur incorporé. En revanche, s'il est muni d'un casque audio, le téléphone sera considéré comme un composant permettant la visualisation et le casque audio comme un autre composant réalisant la restitution du son. La qualité du son n'étant pas la même, les écouteurs offrent la stéréo, la QdS est différente et l'utilisation du téléphone dans les deux cas se rapportera à deux configurations différentes de l'application.

3.3.1 Définition des composants

Un composant est vu comme une entité logicielle conforme à un modèle, indépendante, déployable pouvant être assemblée avec d'autres [Heinman, 2001] [Szyperski, 2002]. Ainsi, de nombreux modèles sont issus des milieux académiques (comme Fractal ou les Bundle OSGi) et industriels (EJB, .NET). En dépit de ces modèles, le paradigme des composants logiciels ne fait pas, pour l'instant, l'objet d'un consensus. En effet, ils font l'objet de nombreuses visions et définitions sensiblement différentes ce qui se retrouve particulièrement avec leurs modes de communication ou de composition.

Ainsi, deux modes de composition se retrouvent :

- *La composition verticale* : 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. C'est ce qui est utilisé par les Enterprises Java Beans de Sun, les .NET de Microsoft] ou encore avec Fractal [Bruneton, 2003a/b] ou Darwin [Magee, 1995]. 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.
- *La composition horizontale* : si la composition verticale n'est pas indispensable, la composition horizontale elle, est capitale, si l'on veut offrir un moyen de développer des applications à l'aide d'interactions de composants. Elle propose de réaliser des assemblages de composants par interaction de ces derniers (la notion de connexion se cache derrière cet aspect compositionnel). Ces interactions peuvent se résumer comme suit :

Table 1 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

3.3.2 Nature des composants

Un Sous-Groupe s'obtient par un assemblage (appelé également composition fonctionnelle [Coupaye, 2001]) de composants logiciels ou matériels existants ou spécialement développés pour l'application. Comme mentionné précédemment, un composant peut être également de nature humaine, aussi, une personne ayant un rôle utile à une fonctionnalité (comme un traducteur) pourra également participer à l'assemblage du Sous-Groupe.

Nous traitons donc de la même manière les composants de l'application quelle que soit leur nature. L'avantage d'une telle approche a été souligné par [Franke, 1991] dans le cas de la co-conception matérielle/logicielle pour les composants logiciels et matériels « *Une approche plus féconde pour la conception des*

systèmes informatiques consiste à combiner les perspectives matérielles et logicielles dès les premières étapes pour exploiter la flexibilité de conception et l'allocation efficace de fonctions que cette approche offre. »

Ce traitement uniforme nous apporte ainsi une flexibilité de conception permettant une maintenabilité accrue pour un coût moindre. Cette flexibilité de conception va se répercuter sur l'implémentation et son adaptabilité (au sens « capacité [d'un système] à réagir selon le contexte, et selon les besoins et préférences des utilisateurs » [Bastien, 1993]).

3.3.3 Différents types de composants

Au-delà de la nature même du composant (logiciel, matériel, humain) chacun peut avoir des caractéristiques spécifiques. Ainsi, un composant logiciel peut éventuellement (ce n'est pas une règle absolue) être dupliqué, voire même dupliqué avec une QdS différente (dans le cas des composants) pour chacune de ses instances selon son contexte d'instanciation. Ces mêmes composants peuvent également être migrés selon le contexte si ce déplacement peut apporter quelque chose en terme de QdS.

Chaque Sous-Groupe rend un service observable à l'utilisateur car nous nous situons dans le cadre d'applications multimédia. Aussi, chaque Sous-Groupe se devra d'avoir un composant appelé **composant observable**[2] [Laplace, 2005] permettant à l'utilisateur de percevoir la partie de service qui lui est rendue. C'est de cette manière qu'il pourra juger de la QdS de la fonctionnalité. Il est à noter qu'un seul composant observable peut être présent. En effet, si le Sous-Groupe ne contient pas d'élément observable, il ne peut fournir de fonctionnalité à l'utilisateur et s'il en contient plusieurs, il fournit plusieurs fonctionnalités, ce qui est contraire à la définition que nous en avons faite.

3.3.4 Rôle des composants

Dans le cas des AMD, le composant représente la brique de base. Chaque composant (unitaire ou composite) possède un **rôle,** noté $\mathbf{r_i}$, comme étant sa fonction au sein de l'application. Le composant étant l'entité atomique, nous nommerons **rôle atomique**, noté $\mathbf{R_i}$, un rôle qui peut être réalisé par un seul composant de l'application. En effet, le principe même de la flexibilité et de l'adaptabilité est de permettre la réalisation de fonctions par différents moyens de manière à ce que l'application s'adapte à son contexte mouvant (fonctionnel ou environnemental).
De manière analogue au composant observable, nous définissions le **rôle observable** comme le rôle qui lui est associé. Un rôle observable est donc un rôle qui permet à l'utilisateur de percevoir la partie de service qui lui est rendue. Il existe donc un et un seul rôle observable par Sous-Groupe.

Les composants sont définis par leur rôle et 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 selon le contexte de l'application. Les flux de données sont des structures qui permettent de transporter (dans le cas présent, de manière synchrone) les médias dans les applications. Ils constituent un outil important de mesure du contexte de l'environnement 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.).

Nous avons défini l'ensemble de la structure de notre modèle d'application. Nous allons la représenter sur la base de l'exemple de vidéoconférence par la Figure 23 qui précise la constitution du Groupe Téléspectateur. On donne aux acteurs le rôle qu'ils ont vis-à-vis du système. Les noms de groupes qui devraient être, selon la terminologie adéquate, des substantifs, mais sont renommés pour des raisons de simplicité de lecture.

[2] Un **composant observable** permet à l'utilisateur de percevoir la partie de service qui lui est rendue et de juger de la QdS de la fonctionnalité : c'est donc un composant dont l'évaluation traduit la qualité du service fourni par le Sous-Groupe. Pour une fonctionnalité typiquement multimédia, comme la restitution des images, un composant observable est l'écran de visualisation.

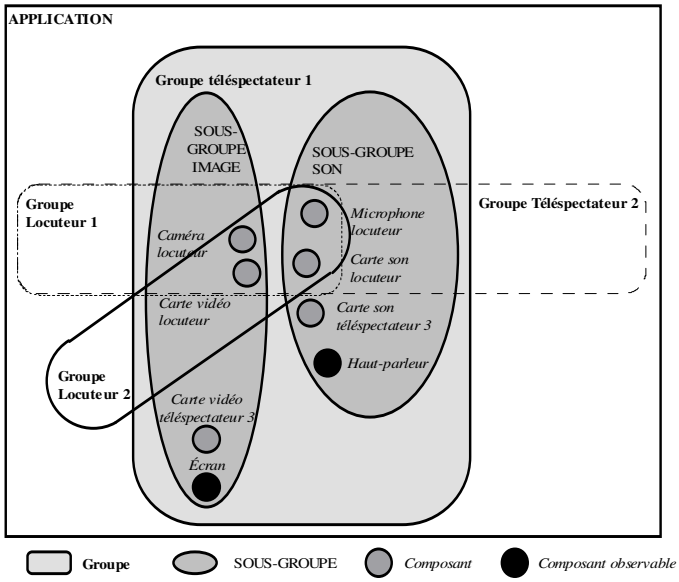


Figure 23 : Modèle d'une vidéoconférence

Il reste à modéliser les liens entre les composants. Selon l'application cible, les communications entre composants peuvent avoir différentes contraintes (temps réel, avec ou sans pertes, synchrones, etc.). Dans notre champ d'application, nous allons voir que la synchronisation est un problème important pour lequel nous proposons à la fois un modèle et des outils d'implémentation. Ces derniers sont adaptables en fonction du contexte de l'application et donc de la politique de communication à mettre en place.

3.4 Communication

Le contexte applicatif (manipulation de flux de données) du multimédia nécessite de s'intéresser aux communications entre composants et plus particulièrement à la synchronisation des flux de données. En effet, ces derniers flux transitant entre les différents composants subissent des traitements qui induisent des retards. Quelques explications s'imposent. Prenons l'exemple d'un flux vidéo synchrone avec un dialogue oral. Un traitement sur la vidéo pour en réduire la taille provoque un décalage. Si plusieurs traitements chaînés sont produits, ce sont autant de décalages produits. Lors de la restitution, le son sera donc en décalage (en avance) sur la vidéo qui deviendra de ce fait incompréhensible. Lors de la restitution la somme des traitements peut provoquer des décalages importants.
Nous proposerons un modèle de connecteur appelé **Korrontea** (signifiant « Conduit » en langue basque) présenté au paragraphe 3.8, page 42 et assurant le transport des flux de données devant rester synchrones. De manière plus générale, il permet de mettre en œuvre des politiques de communication (la synchronisation est une politique de communication) dans les Conduits assurant les échanges entre composants. Associé à ce modèle, nous proposerons le modèle de composant appelé **Osagaia** (« Composant » en langue basque) assurant, au niveau container, cette politique de synchronisation de sorte qu'elle devienne un aspect non fonctionnel.

3.5 Méta-Modèle des applications

Les applications sont construites à partir des différentes entités que nous avons définies : Groupe, Sous-Groupe, Composant (*Osagaia)*, Conduit (*Korrontea)*. Nous proposons un modèle structurel hiérarchisé des applications. Ainsi, un groupe est un agrégat de sous-groupes, eux même formés par un agrégat de composants (dont un observable) et de connecteurs permettant les échanges de flux via les conduits

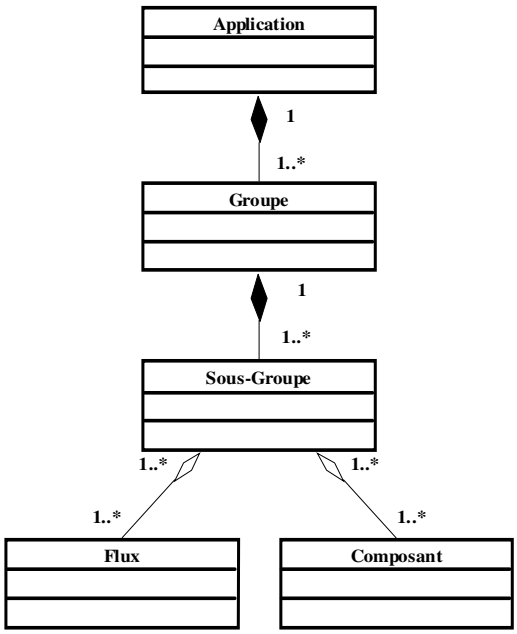


Figure 24: Modèle général des applications multimédia réparties

Ce modèle général propose une analyse de l'application qui se fait en terme de service rendu à l'utilisateur *i.e.* les Groupes. Cette vision de haut niveau permet de faire abstraction des contraintes matérielles et logicielles pour se concentrer sur la réalisation d'une application multimédia viable et utilisable. D'autre part, le concept de Sous-Groupe permet de traiter l'adaptation à l'environnement en faisant abstraction des contraintes de localisation puisque les composants le constituant sont répartis sur les différents hôtes de l'application. Ce modèle issu d'une analyse descendante permet donc de traiter en amont la gestion du contexte aux niveaux environnement, réflexivité, qualité, puis de s'intéresser aux fonctionnalités de bas niveau de l'application avant de procéder à l'implantation et mettre en œuvre les mécanismes d'adaptation, de migration et de composition.

3.6 Architecture des Applications Multimédia Distribuées

L'architecture globale d'une Application Multimédia Distribuée (AMD) se divise en deux parties décrites sur la Figure 25 [Dalmau, 2002], [Laplace, 2006] :

- Une partie applicative implémentant les fonctionnalités des AMD,
- Une plate-forme d'exécution supervisant et gérant dynamiquement l'exécution de la partie applicative en fonction du contexte utilisateur, matériel et temporel.

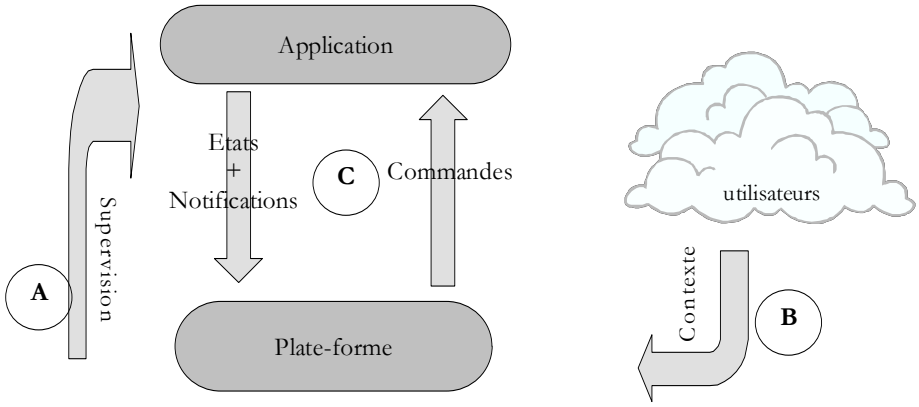


Figure 25 Architecture Globale des Applications Multimédia Distribuées

La plate-forme devant superviser et gérer les composants formant la partie applicative, ces derniers doivent fournir des états et des notifications sur leur fonctionnement (Figure 25 – **C**). Ces informations associées à celles de QdS fournies par les utilisateurs (Figure 25 – **B**) comme le nombre de couleurs préféré, la qualité sonore, la cadence souhaitée des images etc. permettent à la plate-forme d'exécution de reconfigurer dynamiquement par supervision (Figure 25 – **A**) les composants à l'aide de commandes (par exemple pour déconnecter un composant et en reconnecter un autre).

Afin d'avoir la connaissance du déploiement des composants en cours, la plate-forme se doit d'être réflexive [Laplace, 2006] [Riveill, 2000] dans le sens où elle nécessite d'avoir connaissance de son comportement et de sa structure. Comme nous adressons la problématique des AMD, il est nécessaire que la plate-forme soit également distribuée sur chacun des sites où des composants sont déployés.
Notre tâche consiste à définir les briques logicielles de la partie applicative dynamiquement manipulables, modifiables, éventuellement paramétrables et sensibles à leur environnement, ce qui implique de choisir un modèle de composant permettant une évolution simple de la structure d'une application par création, suppression, connexion, déconnexion et migration des composants.

3.7 La Plate-forme d'exécution

Notre objectif principal est de fournir une qualité de service acceptable aux utilisateurs, sans pour autant prétendre fournir la meilleure possible (ce problème se ramène à trouver un assemblage optimum, or ce problème est NP-Complet dans le cas général), et ce relativement aux capacités mouvantes de l'environnement d'exécution. Plusieurs solutions d'applications adaptables ont été proposées dans [Magee, 1996] [Campbell, 1992] [Hagimont, 2002] [Kon, 2001] [Layaida, 2004 [Segarra, 2002] [Singhoff, 1999]. Ces solutions proposent des modèles de composants et des frameworks, basés sur Fractal dans le cas de [Hagimont, 2002] et [Layaida, 2004] (projet *INRIA-Sardes*) et donc offrant la composition dynamique (également vrai sur ce point pour [Magee, 1996]) et hiérarchique. Les travaux cités précédemment offrent également des middlewares dont certains comme ceux proposés par [Kon, 2001] et [Campbell, 2002] offrent de plus la notion de réflexivité, impérative dans notre cas si l'on souhaite proposer une adaptation dans l'optique d'assurer une certaine qualité de service. Enfin, [Diot, 1995] propose une architecture permettant de prendre en compte le contexte réseau (délai, débits, gigues, etc.) et d'en proposer un retour afin d'adapter l'application et ainsi assurer une qualité de service suffisante pour améliorer les transferts de données.
Il existe également plusieurs projets INRIA traitant des plateformes d'exécution et de la batterie d'outils relatifs. Une synthèse de ces travaux a été réalisée dans le cadre de la proposition du projet **SyComor** (Systèmes Communicants Coopératifs – architecture logicielle et protocoles) en cours d'évaluation auprès de l'INRIA Bordeaux-Sud Ouest) et permettant de positionner nos travaux en cours.

Chaque projet peut être décomposé en trois grands axes :

- La partie architecture porte sur les aspects de la plateforme et sur l'auto-adaptation.
- La partie conception se focalise sur les modèles et langages.
- Enfin, la troisième traite du contexte et des composants logiciels utilisés.

Tableau 2 : Synthèse des projets INRIA

PROJET		Architecture		Conception		Implémentation	
		Plateforme	**Auto-adaptation**	**Modèles**	**Langages**	**Contexte**	**Composants**
						X	
ACES		X				X	X
ARES		X				X	X
ARLES			X				
JACQUARD					X		
PHOENIX					X		
POPS						X	X

Le tableau ci-dessus résume l'apport principal de chacun des projets INRIA dans un des domaines que j'aborde dans mes travaux.

La plate-forme que nous proposons est distribuée sur l'ensemble des sites physiques nécessitant un déploiement de composants. Cette distribution lui permet d'être à même de recevoir les états des composants et de leur envoyer des commandes (cf. Figure 25) le cas échéant afin de modifier sa configuration s'il propose plusieurs niveaux, ou bien de modifier dynamiquement la structure même de l'application par arrêt, démarrage, déplacement, etc. d'un ou de plusieurs composants. Ainsi, la plate-forme peut :

- modifier le comportement des composants logiciels (lorsqu'ils proposent différents niveaux de QdS) ;
- modifier la composition de l'application (par ajout, retrait, remplacement ou migration) intervenant sur l'un des points de modification introduits précédemment, à savoir au niveau des Groupes, Sous-Groupes, composants et par conséquent des flux de données.

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 plate-forme d'exécution se structure autour de cinq gestionnaires, chacun étant chargé de réaliser une tâche spécifique. La Figure 26 décrit le modèle structurel de la plate-forme :

- 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 du site en question.
- Un gestionnaire de communication est associé à chaque site de l'application, il assure la communication entre tous les sites où l'application est déployée et par conséquent la plate-forme. Bien que distribuée, il assure que la plate-forme se comporte comme une unité de supervision unique pour la totalité de l'AMD.
- Un gestionnaire d'utilisateur est associé à chaque utilisateur et donc à chaque groupe qui lui correspond, il est chargé de recueillir les vœux de celui-ci.
- Un gestionnaire de supervision est associé à chaque site de l'application, il gère les reconfigurations des composants et des flux de données localisés sur un site.

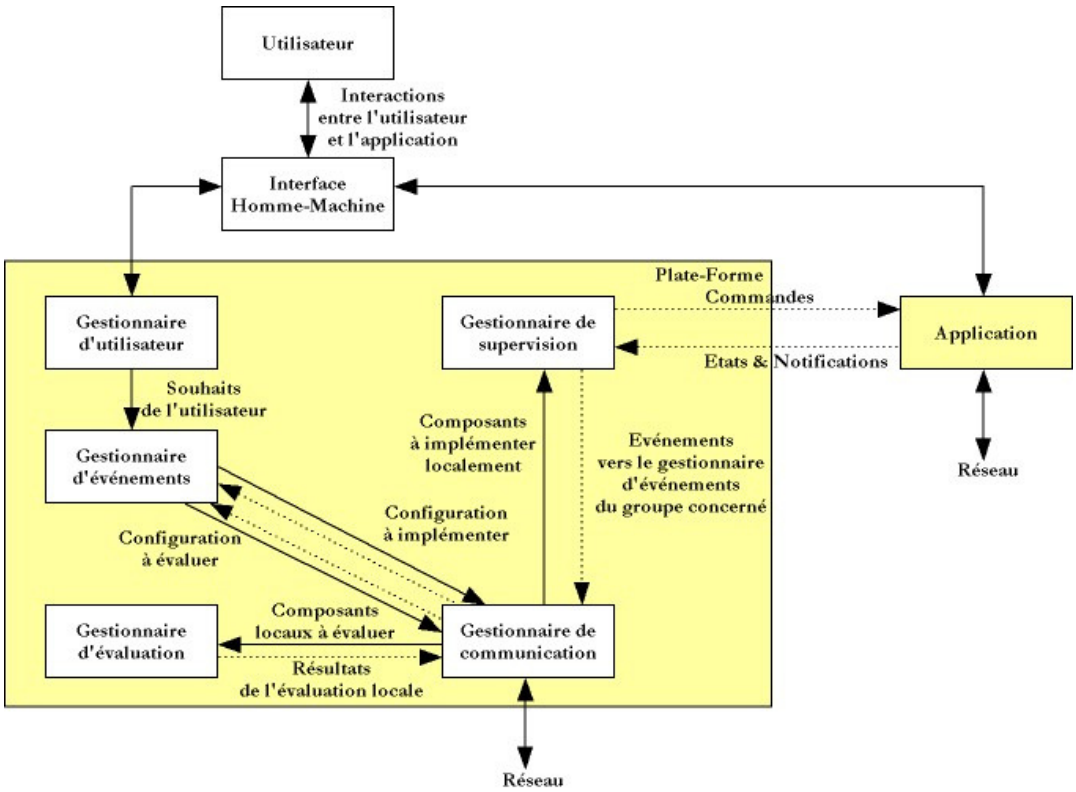


Figure 26 Modèle Structurel de la Plate-forme [Laplace, 2006]

Dans la mesure où nous définissons une architecture d'applications distribuées capables de prendre en compte le contexte, le modèle structurel complet de la plate-forme se trouve réparti sur chaque hôte supportant l'exécution de l'AMD.

3.8 Korrontea, un Modèle de Flux de Données

L'échange de flux de données dans les applications multimédia impose un certain nombre de particularités dont la principale est la synchronisation. Elle est de deux types : intra-flux (entre données d'un même flux) et inter-flux (entre différents flux).
Concernant les synchronisations intra-flux, nous avons identifié deux types de propriétés pour les médias : continu et discret. Les premiers possèdent une notion de temps et séquence entre chaque donnée alors que les seconds ne sont que des données ponctuelles. La manipulation de toutes les données échangées dans les applications que nous traitons étant réalisée sous forme de flux de données, cette distinction est importante car elle permet ainsi de les caractériser.

Les avantages d'une seule structure d'échange sont multiples :

- Manipulation uniformisée de l'ensemble des données,
- Intégration facilitée,
- Prise en compte des propriétés de séquence et de temps, si nécessaire.

3.8.1 Les Flux de Données

Toute production de données prend la forme d'un flux de données. Chacune d'elle est encapsulée dans des unités d'informations [Blakowski, 1996] comprenant un numéro de séquence traduisant l'ordre et produit par une horloge logique. Chaque unité d'information est ensuite encapsulée dans une tranche synchrone qui utilise une horloge physique afin d'obtenir des traductions en durées de mesures réelles.

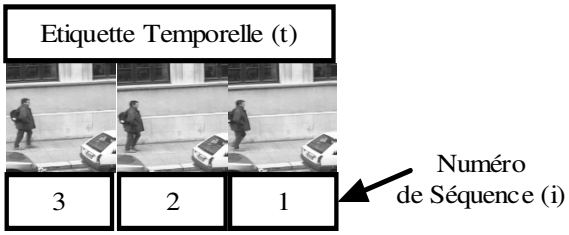


Figure 27: Tranche synchrone contenant un flux vidéo

Les tranches synchrones sont nécessaires à la production de flux de données synchrones (suite a priori infinie de tranches synchrones). Elles permettent d'exprimer les relations de synchronisation inter-flux. Il en existe deux types selon qu'il y a un seul flux de données (flux primitif) ou plusieurs (flux composé) pour lesquels il est nécessaire de conserver la synchronisation. Les flux synchrones permettent de prendre en compte les relations temporelles et de séquences de chaque flux entrant dans la composition. D'un point de vue formel, la relation de synchronisation définit une relation d'ordre strict entre les unités d'informations et les tranches synchrones [Bouix, 2008].

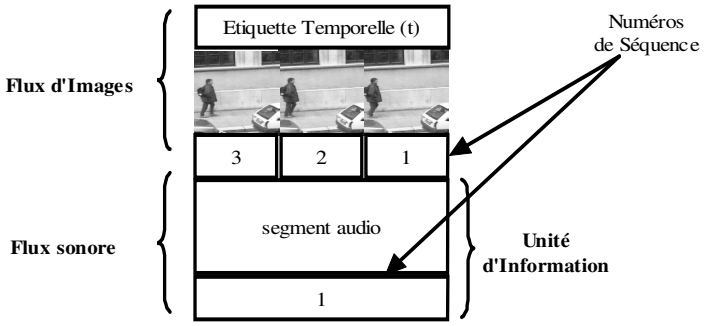


Figure 28: Tranche Synchrone issue d'un flux synchrone composé

3.8.2 Les contraintes temporelles

Les AMD manipulent deux types de données communément appelées médias continus et médias discrets. 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. Néanmoins, cette classification n'est pas suffisante. Prenons l'exemple d'un média composé d'une vidéo et de sous-titres. Nous avons ici l'exemple type d'un flux composé dont l'un des flux composants est continu et l'autre discret. Nous voyons dans ce cas que les contraintes temporelles entre les deux flux sont bien distinctes. Dans l'exemple précédent, si le flux vidéo est un flux « régulier » dans le temps, celui des sous-titres ne l'est pas. Néanmoins, la régularité ne peut pas être un critère pour caractériser un flux. Prenons toujours l'exemple d'un flux vidéo avec lequel on souhaite synchroniser une présentation de type « diapositives » dont chaque image apparaît toutes les minutes. Bien que ce média soit régulier, il n'en demeure pas moins que le comportement du flux est bien différent.

Ainsi, nous définissons les flux de données à contraintes temporelles fortes et faibles. Un flux est à contrainte forte si l'on est capable de définir une fenêtre dans laquelle on est sûr d'avoir au moins une unité d'information. Il est à contrainte temporelle faible lorsque l'on ne peut pas définir de façon certaine une fenêtre qui contiendra au moins une unité d'information. Il apparaît donc que les medias peuvent être classés selon leurs contraintes temporelles [Ghinea, 1998] [Steinmetz 1996]. Cette classification n'est pas universelle, et il n'est pas possible de définir strictement une valeur « générique » permettant de distinguer les flux. Ce travail sera fait lors de la conception et dépend en grande partie de ce que l'on souhaite faire de ces informations au niveau applicatif.

Nous avons défini un modèle de flux de données représenté ici à l'aide d'un diagramme de classes en UML (Figure 29).

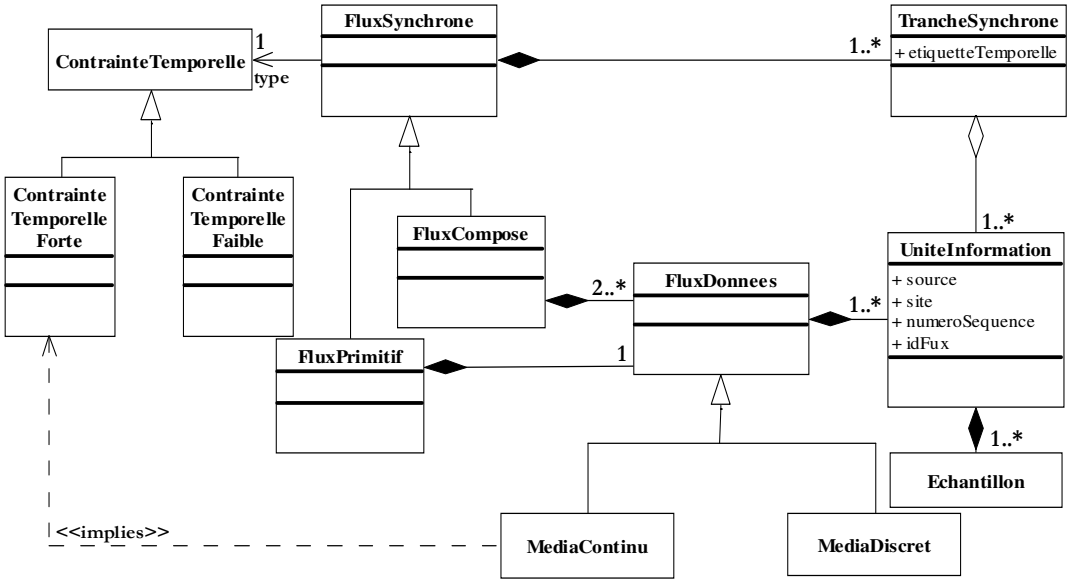


Figure 29: Modèle Conceptuel Korrontea

Les politiques de synchronisation fortes et faibles ont été démontrées formellement dans [Bouix, 2008]. Le lecteur pourra s'y référer afin d'obtenir la liste des propriétés mathématiques et leur démonstration.

3.8.3 Politiques de synchronisation

Comme nous l'avons exprimé, la synchronisation dans le cadre du multimédia est quelque chose d'essentiel au risque d'obtenir des informations incompréhensibles.

Synchronisation intra-flux

Les relations de synchronisation expriment les relations entre les données formant le flux (leur débit). Par exemple, une vidéo à 25 images/seconde implique l'affichage d'une image toutes les 40 millisecondes. Ces relations ne sont pas strictes et une certaine latence peut être acceptée [Ghinea, 1998] [Steinmetz, 1996]. Pour chaque flux synchrone, il est possible de définir la synchronisation intra-flux en utilisant le numéro de séquence de l'unité et l'étiquette temporelle des tranches synchrones. La relation d'ordre total strict [Bouix, 2007] permet d'ordonner les unités d'information de ces flux. Les contraintes temporelles de chacun d'eux déterminent les relations de synchronisation intra-flux en fonction d'une valeur θ définie au moment de la conception (il n'est pas possible de définir une valeur de θ a priori (cf. propriété 5 et 6 [Bouix, 2007]).

Le nombre de séquences, les étiquettes temporelles et les relations de synchronisation sont définis pour chaque flux lors de sa création par le composant adéquat appelée source localisée.

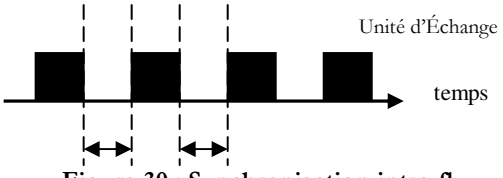


Figure 30 : Synchronisation intra-flux

Synchronisation inter-flux

La synchronisation inter-flux correspond aux relations de synchronisation qui peuvent exister entre les données de plusieurs flux. C'est le cas par exemple entre le flux d'images et celui correspondant à la bande son d'une vidéo. Nos travaux s'intéressent uniquement aux données crées/capturées sur le même site. En effet, l'objectif n'est pas de créer des relations de synchronisation mais bien de les conserver.

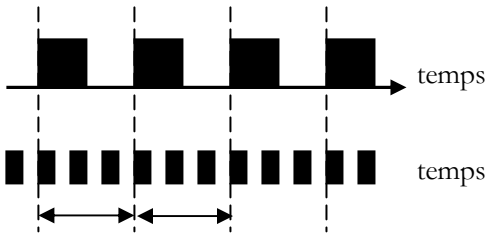


Figure 31 : Synchronisation inter-flux

Les flux ainsi liés par des relations de synchronisation inter-flux sont rassemblés selon des politiques de synchronisation dans des flux composés.

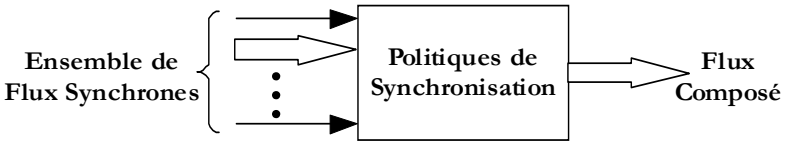


Figure 32 : Production de flux composés

Il existe trois types de politiques de synchronisation :

- **Politique Forte** : utilisée lorsque l'on veut conserver les relations inter-flux entre plusieurs flux synchrones à contrainte temporelle forte.
- **Politique Faible** : utilisée lorsque l'on veut conserver les relations inter-flux entre des flux synchrones à contrainte temporelle faible.
- **Politique Mixte** : utilisée lorsque l'on veut conserver les relations inter-flux entre des flux synchrones à contrainte forte et des flux synchrones à contrainte faible.

Lorsque plusieurs flux de données sont à contraintes fortes, la politique de synchronisation forte assure qu'une tranche synchrone contient au moins un échantillon de chacun. La politique faible réalise des fenêtres de longueur θ dès lors qu'un échantillon a été reçu sur l'un des flux, assurant qu'il y a au moins un échantillon dans chaque tranche. Enfin, la politique de synchronisation mixte assure qu'une tranche synchrone contiendra au moins un échantillon de chaque flux à contrainte forte. Le détail des politiques et de leur fonctionnement est expliqué dans [Bouix, 2007].

3.9 Connecteurs

Afin d'être complet, nous devons définir un moyen permettant la connexion entre ces composants afin de réaliser les assemblages. Nous utilisons pour ce faire un conteneur appelé « Conduit » dont le rôle est de permettre le transport de données (localement ou non) selon un mode particulier (nous avons présenté le transport synchrone de flux de données). Bien que le connecteur « Conduit » soit générique, son fonctionnement est fortement 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 contexte permettant de déceler des anomalies de fourniture de service mais aussi des possibilités d'amélioration.
Le Conduit se dote donc de deux unités qui sont l'unité d'échange et l'unité de contrôle. L'unité d'échange est dédiée à la gestion des connexions en entrées et en sorties du Conduit. Elle se compose de deux unités, chacune étant 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.

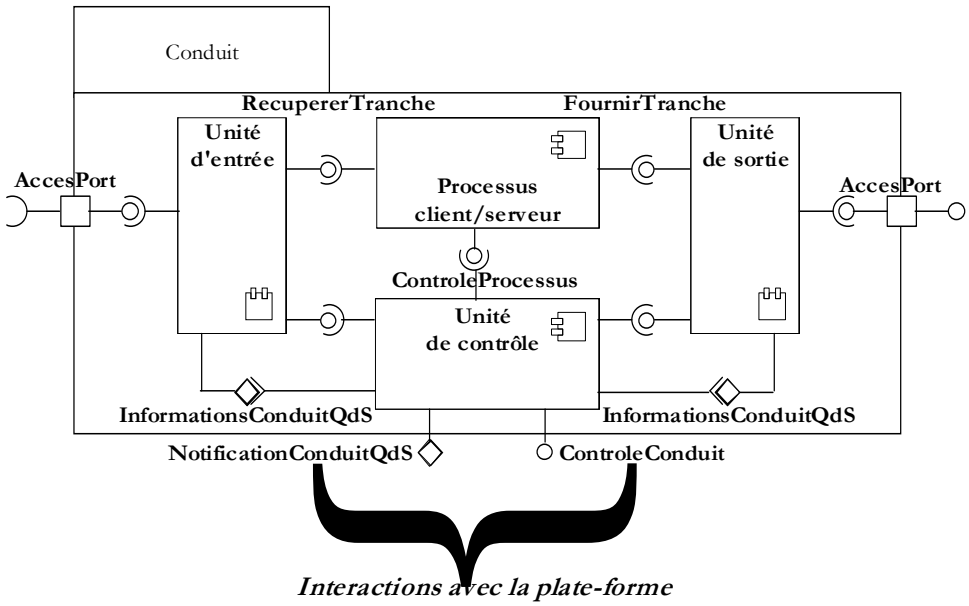


Figure 33 Architecture interne du Conduit

3.9.1 L'unité d'échange

L'unité d'échange se décompose en deux parties distinctes, à savoir l'Unité d'Entrée (UE) et l'Unité de Sortie (US) 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 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 compatibles les connexions de ce dernier et des composants logiciels (le détail des ports est disponibles dans [Bouix, 2007].

Le but du Conduit est de transférer les données du composant producteur vers le composant consommateur. Ces données sont transférées sous la forme de flux synchrones composés ou primitifs. Dès 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. L'utilisation d'un buffer est justifiée par le Conduit distribué. En effet, lors des communications réseau il est nécessaire de bufférisier les tranches synchrones avant de les envoyer. Ainsi, les tranches reçues du composant producteur sont stockées avant d'être envoyées. 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.

3.9.2 L'Unité de Contrôle

L'UC est chargée de la supervision du Conduit par la plate-forme. L'exécution du Conduit est commandée par la plate-forme et ce dernier est susceptible de lui délivrer des informations sous forme d'événements qui traduisent son état de fonctionnement. Dans le cas d'un Conduit distribué, l'UC est distribuée sur deux sites et se divise en deux processus (un client et un serveur). En ce qui concerne les événements traduisant des informations de QdS, ils sont envoyés à la plate-forme locale du site où le morceau de Conduit se trouve. Par exemple, lorsque l'événement est envoyé par l'UE, il est reçu par l'UC correspondante et est transmis à la plate-forme locale à ce site. En fait, lorsque le Conduit est distribué, chaque partie possède une UC locale qui permet de le commander et de recevoir son état de fonctionnement.

3.10 Le modèle de composant Osagaia

Les flux de données ne sont pas le seul type de données échangées entre des entités de l'application, mais il est intéressant de ne voir les échanges que sous cette forme afin d'unifier la gestion des communications. Cette partie de nos travaux consiste à définir un modèle de composant logiciel afin de permettre l'implémentation d'applications distribuées multimédia. Ce modèle est appelé Osagaia (signifiant « Composant » en langue basque).

Nous séparons l'implémentation des composants 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, 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 dans l'architecture, donner des informations sur son contexte local ou recevoir des commandes de la plate-forme de supervision suite à un changement de contexte global.

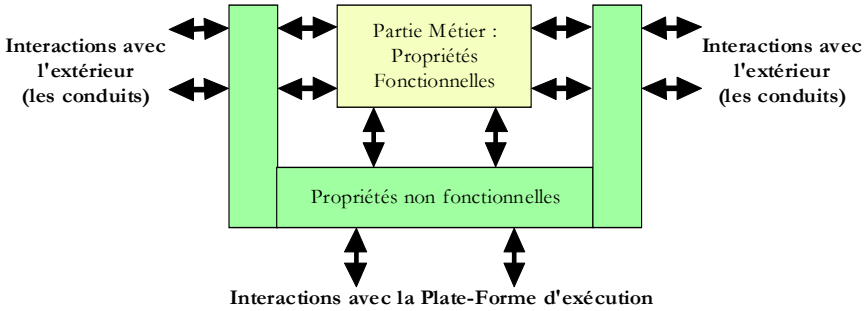


Figure 34 : Principes de Fonctionnement d'Osagaia

Dans le modèle Osagaia (Figure 34), la partie métier utilise des propriétés non-fonctionnelles afin d'assurer son interaction avec la plate-forme d'exécution et les Conduits assurant le transport des données.

Nous avons classé les propriétés fonctionnelles et non fonctionnelles (Figure 35) en fonction de la nature des préoccupations auxquelles elles répondent. On y 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 34. C'est à ce niveau qu'agit directement le modèle de flux de données Korrontea (signifiant « Flux de Données » en langue Basque). 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.

Propriétés non-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
Interaction du composant avec les autres entités (synchronisation des entités)	Liaisons entre les différentes unités d'un composant
Lecture des données provenant de l'extérieur et écriture des données vers l'extérieur	Centralisation des informations 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)	Connexions et déconnexions du composant

Propriétés Fonctionnelles
Accès aux données
Implantation métier du composant

Figure 35 : Liste des propriétés du modèle de composant Osagaia

Dans de nombreux travaux sur les composants logiciels dont on trouvera un descriptif dans [Duclos, 2002], l'accent est mis sur l'importance des propriétés non fonctionnelles. Une proposition de modèle de composant 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. Le problème mis en cause est la rigidité de ce type de structure, ce qui n'est 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 [Lopes, 1995]. 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. Cette approche correspond plus à notre vision des choses en nous permettant de tirer les avantages des propriétés de flexibilité et de malléabilité offertes par le paradigme composant. Nous pensons que plus l'indépendance des propriétés est claire, plus la réutilisabilité est satisfaite. Ce type de solution préconise l'utilisation d'entités dédiées appelées conteneurs [Sun, 2003] [Bruneton, 2002] [Bruneton, 2003a/b]. 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 des composants et des interactions probables avec son environnement. Ainsi, on peut réutiliser ces propriétés en réutilisant des conteneurs pour chaque composant logiciel d'une application.

C'est pourquoi nous fournissons une structure permettant de gérer l'ensemble des propriétés présentées dans la Figure 35. Cette structure se scinde en deux parties distinctes qui interagissent ensemble selon la Figure 34:

- une partie, appelée *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 en entrées/sorties aux données ;
- une partie, appelée *Processeur Elémentaire (PE)*, a pour but de fournir les propriétés non fonctionnelles nécessaires au fonctionnement du CM mais aussi à l'ensemble des Conduits.

L'architecture du **Processeur Élémentaire** (ou PE) est présentée à la Figure 36. Son rôle est de mettre en œuvre les interactions entre les composants métier et leur environnement. Le PE est divisé en deux parties principales. La première, appelée l'unité d'échange (composée des unités d'entrées/sorties) est chargée des connections aux flux de données synchrones. La seconde, l'unité de contrôle gère le cycle de vie du composant métier et les interactions avec la plate-forme d'exécution. C'est cette partie essentielle qui permet de notifier des informations sur le contexte local du composant (sur ou sous employé, état des unités d'entrée/sortie donnant des informations sur les communications, etc.), mais également de recevoir des commandes de reconfiguration de la plate-forme. Outre son rôle de conteneur pour le CM, le PE conserve la synchronisation des flux liés par des relations de synchronisation. 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.

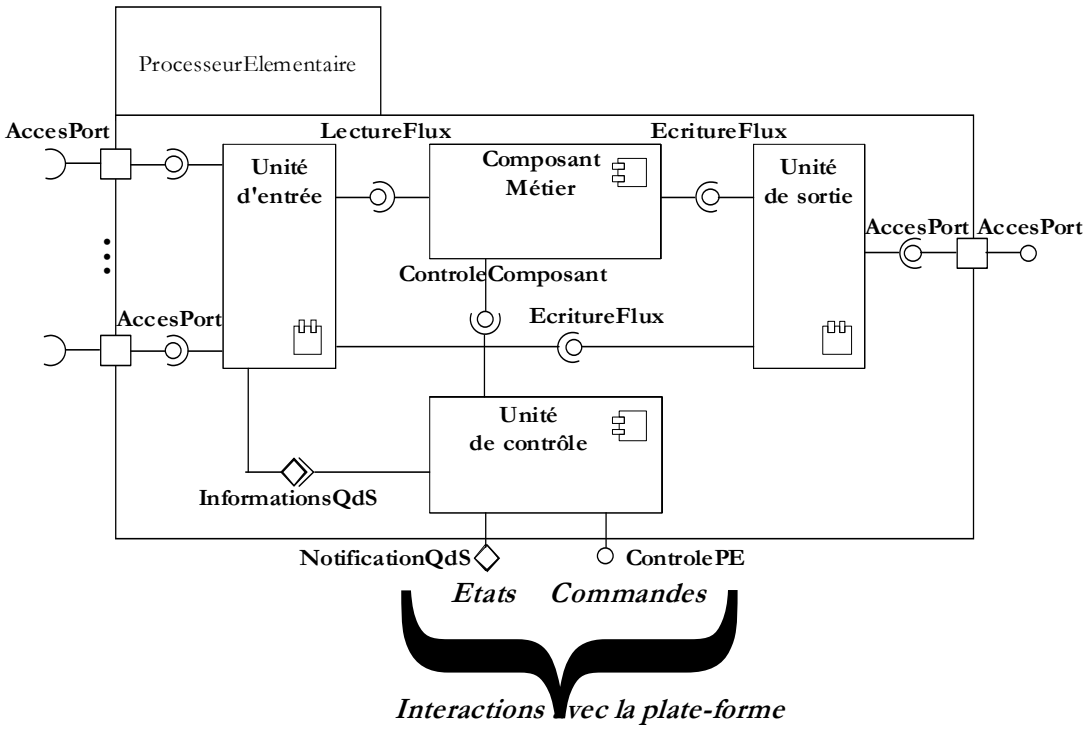


Figure 36 : Architecture du Processeur Élémentaire (PE)

Le nom de Processeur Élémentaire 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 [Carlström, 2004] [Kapasi, 2003]. Dans ces architectures, les applications sont 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 telle architecture. Ce type de séquence permet d'obtenir une synchronisation implicite des traitements [Lee, 1994]. Ce type d'architecture est utilisé en particulier pour le calcul parallèle distribué car il permet une réduction des temps de calcul.

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.

Les AMD sont supervisées par une plate-forme dont le rôle est d'évaluer le contexte global et de proposer des réorganisations de l'application par ajout, suppression, et déplacement de composants s'il s'avère que la qualité de service souhaitée ne peut être atteinte en l'état actuel des choses. Pour ce faire, la plate-forme a besoin d'informations sur le contexte de l'application et doit pouvoir transmettre des ordres aux composants qui la constituent. En tant que conteneur du CM, le PE peut évaluer le fonctionnement de ce dernier et en particulier la QdS. 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 permettant ainsi une réorganisation de l'architecture interne de l'application sous le contrôle de la plate-forme.

3.10.1 Les Unités d'Échange et de Contrôle

L'unité d'échange, composée de l'Unité d'Entrée (UE) et de l'Unité de Sortie (US), a pour rôle de gérer les entrées et les sorties du PE. Elle est chargée de réceptionner les flux de données provenant du ou des Conduits connectés en entrée et de fournir en sortie du PE un flux de données 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 de manière synchrone

avec les flux traités. En fait, l'unité d'échange est chargée de séparer les flux composés 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 temporels des tranches synchrones afin, suivant les types de traitement, de conserver la cohérence des informations.
Enfin 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 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.
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 est capable d'ajouter, de retirer, de remplacer ou de déplacer un CM d'un site vers un autre et enfin de modifier le niveau de qualité fourni par les CM paramétrables lorsqu'ils le sont. 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.
Lorsque la plate-forme décide de changer un CM pour, par exemple, le remplacer par un autre (Figure 37), le PE qui contient le CM à remplacer (celui en cours de fonctionnement) doit être déconnecté de l'application. Ainsi, le CM à ajouter est encapsulé dans un nouveau PE que la plate-forme se charge de construire.

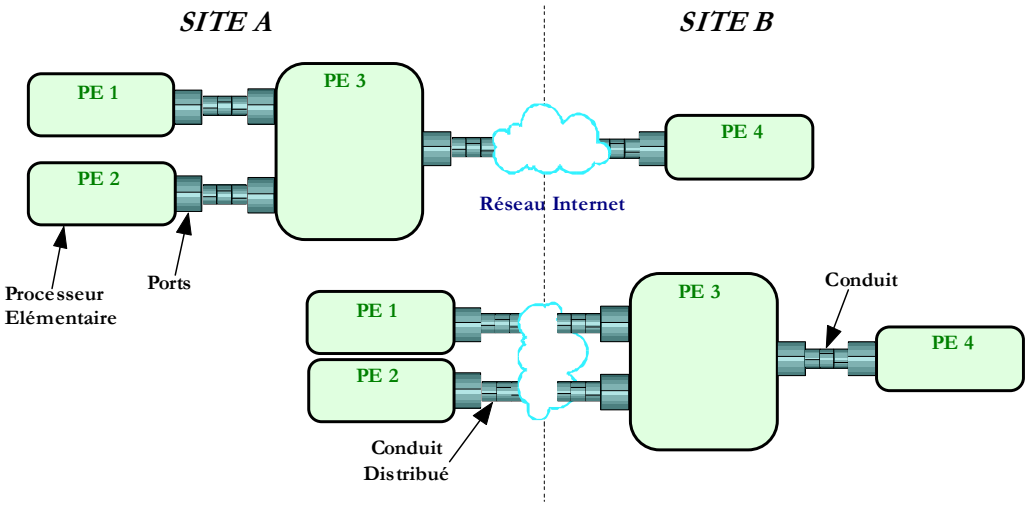


Figure 37 : Migration de composants

3.11 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. 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.

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.
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. La communication événementielle mise en œuvre est décrite à la Figure 38. 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. Il faut préciser que la séquence décrite se répète pour toutes les tranches écrites une à une dans les ports de sortie des Conduits connectés en entrée d'un PE.

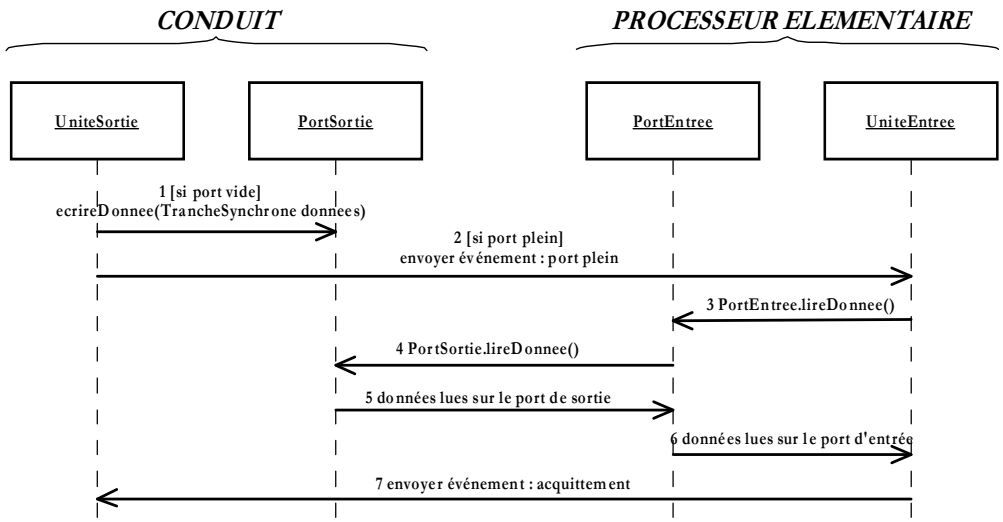


Figure 38 : Diagramme de Séquence

Un Conduit peut accepter plusieurs Conduits en entrée dans le cas où certains des flux nécessaires au traitement du PE se trouvent regroupés dans des Conduits différents. Chacun peut être connecté en entrée d'un PE, plus précisément à son UE. Nous utilisons dans ce cas 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.
Sur la base du diagramme de séquence précédent, on peut ainsi détailler un certains nombre de cas selon le taux de remplissage croissant ou pas des buffers d'entrée/sorties. L'étude de leurs états permet de tirer des conséquences sur le contexte matériel de l'application, à savoir si le réseau est en cause (problème de débit) ou si c'est un problème fonctionnel (le PE ne peut suivre le rythme des tranches synchrones) qui nécessitera une reconfiguration de l'application (cf. Chapitre 5) suite à une modification du contexte. On identifie les cas suivants :

Origine	Cause	Interprétation
Conduit	1. UE vide	Le débit réseau est suffisant
	2. UE pleine	Le débit réseau ne suit pas
	3. US vide	Si l'UE est pleine, le débit réseau ne suffit pas Si l'UE est vide, il n'y a rien à transmettre
	4. US pleine	Le PE ne suit pas
PE	5. UE vide	Le débit réseau est suffisant
	6. UE pleine	Le PE ne suit pas
	7. US vide	Le débit réseau est suffisant
	8. US pleine	Le débit réseau ne suit pas

3.12 Synthèse

Les deux modèles **Osagia** et **Korrontea** ainsi que la plate-forme **Kalinahia** proposés dans ce chapitre sont des outils d'implémentation précieux. Leur conception a été guidée par la plate-forme intrusive **Kalinahia**, c'est-à-dire, permettant des reconfigurations de l'application pilotées par la plate-forme.

Le modèle de composants Osagaia permet la connexion dynamique à d'autres composants permettant ainsi de répondre aux besoins d'adaptation et de migration des composants des applications. Ils permettent de notifier à la plate-forme des informations contextuelles locales sur l'état des composants et de recevoir en retour des commandes agissant sur leur cycle de vie.
Les Conduits permettent de mettre en œuvre les communications entre ces composants. Ils ont été conçus avec les mêmes contraintes et peuvent donc donner des informations contextuelles relatives au matériel. Ils peuvent être supervisés par la plate-forme afin de modifier les connexions. Ce modèle permet de mettre en œuvre différentes politiques de communication de données afin de répondre aux besoins. Nous avons ici proposé la politique de communication synchrone propre aux applications multimédia, mais rien n'interdit d'en mettre d'autres en œuvre proposant par exemple le transport temps réel, le transport avec pertes, etc. en fonction des caractéristiques des applications concernées.

Ces différents outils (Kalinahia, Osagaia et Korrontea) nous permettent de répondre aux différents besoins de reconfiguration. Ils permettent la *reconfiguration évolutive* [Ketfi, 2004], autorisant l'ajout et/ou le retrait de composants. Le contexte mouvant des applications implique des reconfigurations permettant de conserver la cohérence d'exécution. Enfin, ces outils permettent également la *reconfiguration perfectible* permettant d'améliorer les performances d'une application même si son fonctionnement est cohérent. Ces outils d'implémentation nous permettent de nous situer clairement dans le domaine de la reconfiguration dynamique et apportent des solutions viables aux solutions de reconfiguration telles la modification d'architectures et la modification de la localisation (ou migration).

Les informations contextuelles notifiées par les composants et connecteurs sont envoyées à la plate-forme, qui récupère les états représentant le contexte d'exécution local de chaque entité (de communication ou fonctionnelle) et réagit en conséquence en envoyant les commandes nécessaires à la reconfiguration de l'application. Cette dernière peut prendre différentes formes : un composant peut être remplacé par un autre, l'assemblage ou la composition des composants peut être modifié, les communications entre composants peuvent être réorganisées, tout en envisageant si nécessaire des migrations de composants sur différents hôtes.

C'est l'objectif du chapitre suivant qui présente un modèle de qualité de service proposant un outil pour mesurer l'intérêt d'un composant dans un assemblage.

Chapitre 4 Contexte et Qualité de service

La démarche première qui guide tout travail est la recherche de la qualité. En informatique, elle concerne à la fois la qualité matérielle et logicielle. La première pose fréquemment un problème important qui est celui de la portabilité. La seconde est un défit auquel fait face l'informatique depuis sa naissance, elle est difficile à obtenir, et donc coûteuse. Bien qu'auparavant négligée, elle tend à être de plus en plus prise en compte en se basant sur le contexte des applications. Au départ, et encore pour beaucoup aujourd'hui, ce terme a été utilisé dans le domaine des réseaux afin de décrire la qualité du service fourni aux applications par les systèmes de communication. Il est communément admis aujourd'hui que cette seule QdS ne peut perdurer. Les applications impliquent de plus en plus l'utilisateur. Aussi, la QdS doit non seulement prendre en compte le contexte matériel, mais également le contexte environnemental, utilisateur voire même temporel.

4.1 Définition de la qualité de service

En cherchant bien, il doit être possible de trouver autant de définitions de la qualité de service qu'il existe d'équipes travaillant dessus. Cependant, un certain nombre de points de consensus existent quel que soit le contexte. La norme ISO la définit comme un ensemble de caractéristiques se rapportant au comportement collectif d'un ou de plusieurs objets [ISO, 1995].
Comme précédemment dit, au départ, la QdS portait essentiellement sur le domaine des réseaux (contexte machine). Elle était généralement décrite par des critères de délai, de gigue, de débit, de taux d'erreur, etc.
Dans le cas d'Internet, l'AFNOR propose un référentiel qui se place du point de vue de l'utilisateur du réseau. Ce référentiel préconisant les bonnes pratiques pour les fournisseurs d'accès [AFNOR, 2004] regroupe trois critères principaux :

- la disponibilité qui indique si le service est effectivement délivré à partir du nombre de tentatives de connexion ;
- l'intégrité qui traduit que le service n'est pas altéré, ce qui est évalué principalement par les pertes de données et la performance ;
- l'efficience du service illustrée par la bande passante ou le débit.

Dans la même logique, il est donc opportun de définir la qualité de service d'une application vis-à-vis de l'utilisateur final. On trouve dans ce cas deux types de définitions proches :

- Une approche intégrant le point de vue « utilisateur » s'attachant plus au résultat de ces caractéristiques sur les performances: « *Quality of service is the collective effect of service performance which determines the degree of satisfaction of a user of the service* » [CCI, 1989].
- Une autre approche orientée « système » décrivant les caractéristiques d'un système : « *Quality of service represents the set of those quantitative and qualitative characteristics of a distributed MM system necessary to achieve the required fonctionality of an application* » [Vogel, 1995].

De notre côté, nous considérerons qu'évaluer la qualité de service consiste à mesurer **l'adéquation entre le service désiré par l'utilisateur et le service qui lui est fourni.** Nous utilisons donc la seconde définition proposée. Nous allons cependant plus avant dans la démarche car nous soulignons la totale subjectivité de la qualité de service : une caractéristique comme le temps de traitement peut être mesurée de façon objective ; en revanche, sa valeur en terme de qualité de service ne peut être définie qu'à l'aide des attentes de l'utilisateur.

4.2 Concept d'utilité

Il existe un concept appelé « utilité » développé en micro-économie. Il permet de définir la « *satisfaction qu'une personne retire de la consommation d'un bien ou d'un service* » [Parkin, 1992]. Cette définition n'est pas bien

éloignée de celle que nous venons de donner concernant la QdS. On peut donner une définition commune « *évaluer la qualité de service consiste à estimer la satisfaction de l'utilisateur* » [CCI, 1989].

Malheureusement, l'utilité n'est théoriquement ni observable ni mesurable, ce qui peut être mis en parallèle avec la qualité de service qui n'est ni entièrement observable ni entièrement mesurable. L'utilité dépend de la quantité de biens consommés par le client ; la qualité de service dépend des valeurs, en termes de qualité, des paramètres ou critères fournis par l'application.

La micro-économie utilise des courbes de niveau d'utilité appelées courbes d'indifférence formées par l'ensemble des points représentant toutes les combinaisons de biens qui procurent la même satisfaction à l'utilisateur et envers lesquelles il n'a pas de préférence. Elles correspondent à un compromis satisfaisant de quantité de biens obtenus pour chacun des biens considérés. S'appliquant à la qualité de service, elles permettent de définir la qualité de service préférée par l'utilisateur en fonction des critères de qualité c'est-à-dire d'évaluer sa satisfaction et donc la qualité de service non plus au sens du réseau mais élargie à l'utilisateur
La théorie des préférences se définit ainsi :

- Les préférences ne dépendent pas du prix des produits.
- Les préférences ne dépendent pas du revenu.
- Une plus grande quantité d'un bien est préférée à une quantité moindre ce qui équivaut à considérer que les consommateurs ont des besoins illimités.

Il est possible de représenter une carte des préférences par une série de courbes d'indifférence qui, en tant que courbes de niveau, ne se coupent jamais. Cela signifie qu'il n'y a pas d'arbitrage à faire lorsque l'une des quantités est connue. Leur allure dépend du degré de substituabilité des biens. Pour la qualité de service, ces courbes représenteront la fonction de composition permettant de définir la note de qualité de service de l'application en fonction de la mesure de qualité de service. C'est donc la fonction de composition permettant d'obtenir une métrique unique de qualité de service. Plus précisément, de proches substituts peuvent très facilement être remplacés l'un par l'autre. Des substituts parfaits sont représentés par des courbes d'indifférences qui sont des droites à pente négative alors que des biens complémentaires sont non substituables. Plus précisément, de proches substituts peuvent très facilement être remplacés l'un par l'autre. Les courbes obtenues sont représentées par la Figure 39.

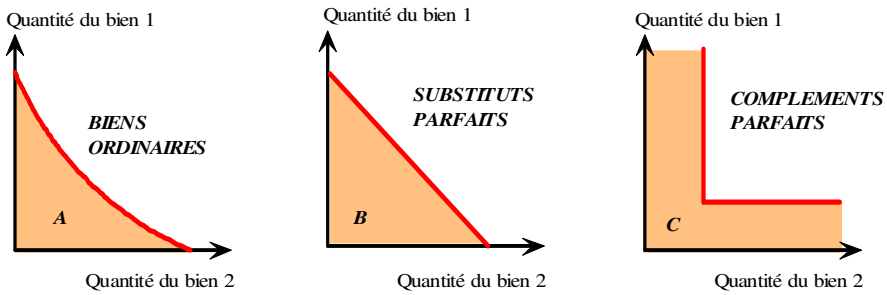


Figure 39 : Satisfaction de l'utilisateur en micro-économie

Dans le cas de la QdS, les deux paramètres de la courbe sont les critères intrinsèque et contextuel (cf. définition en 4.4). Or ces critères sont orthogonaux : une caractéristique ne dépendant pas du contexte ne peut pas être remplacée par une caractéristique dépendant du contexte. Ces critères sont donc non substituables. Par analogie avec les biens, nous dirons qu'ils sont parfaitement complémentaires (*courbe C*). Cette approche économique originale sert au système pour gérer la qualité de service. Elle permet de définir une mesure de qualité traduisant fidèlement les attentes des utilisateurs.

Afin d'illustrer notre modèle de qualité de service, nous allons utiliser un exemple présenté dans le paragraphe suivant.

4.3 Exemple d'application multimédia répartie sur Internet

Nous reprenons l'exemple de la vidéoconférence (Figure 23) auquel nous ajoutons une gestion de la langue des locuteurs, ainsi que quelques fonctionnalités comme la transmission de fichiers, la possibilité de

se déplacer devant un tableau/écran pour y montrer des informations, la possibilité de percevoir le discours soit par voie classique sonore, soit par le biais de sous-titres (cf. Figure 40).
Ce schéma précise les localisations que nous appellerons site, notés $\mathbf{S_i}$, et le matériel utilisé. Pour chaque terminal informatique, nous parlerons de poste informatique, noté $\mathbf{P_i}$. Ainsi, la vidéoconférence possède deux sites, S_1 et S_2, équipés d'ordinateurs personnels P_1 et P_2, d'un casque audio, d'un microphone et d'une caméra motorisée. Deux autres sites servent pour les téléspectateurs. Le premier S_3 héberge un utilisateur qui reçoit la conférence sur son téléphone portable (P_3) et équipé d'un casque audio. Le second S_4 est constitué par l'ordinateur personnel P_4. Nous supposons que tous les postes des locuteurs sont équipés d'une carte d'acquisition vidéo servant également de carte graphique pour la visualisation sur un écran. Cet exemple nous permettra d'illustrer notre modèle de qualité de service.

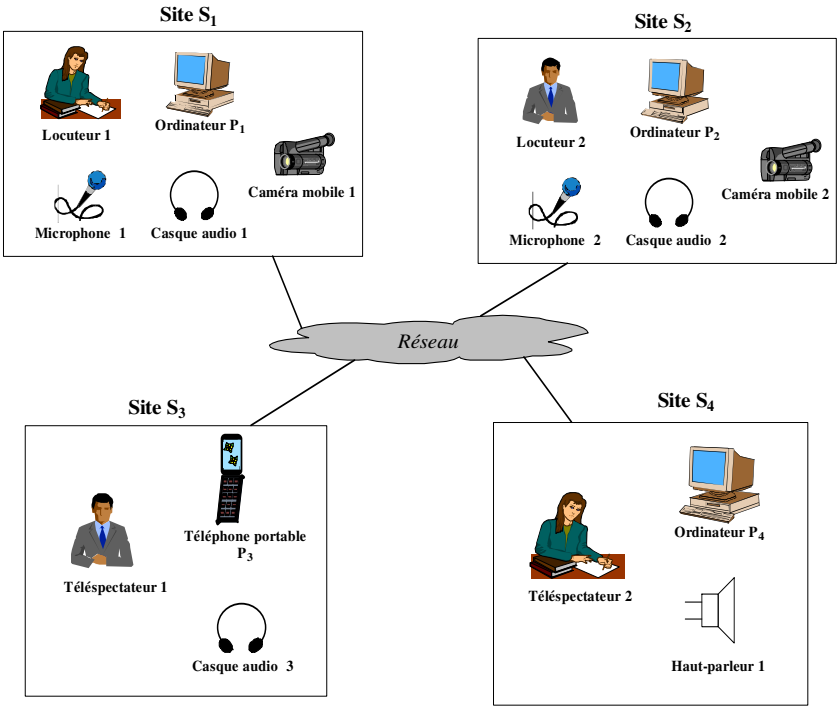


Figure 40 : Exemple de vidéoconférence

4.4 Structure du modèle

L'une des principales contraintes des applications distribuées multimédia est leur caractère imprédictible. Certaines caractéristiques de service dépendent des variations du contexte alors que d'autres sont fixées par les choix de programmation de l'application.
Nous pouvons donc utiliser la sensibilité d'une caractéristique de service au contexte comme argument de discrimination. C'est pourquoi nous regroupons les caractéristiques en deux ensembles appelés **critères de QdS** : un critère intrinsèque et un critère contextuel.
Le **critère intrinsèque** regroupe les caractéristiques indépendantes du contexte c'est-à-dire du réseau, du matériel utilisé ou de l'environnement non informatique. Il reflète la comparaison entre la manière dont l'application a été conçue et les vœux de l'utilisateur permettant de définir leur adéquation du point de vue fonctionnel. Il peut prendre différentes valeurs si un composant (composite ou non) est capable de fournir différentes fonctionnalités ou de fournir différentes qualités. Ainsi, parmi les caractéristiques d'une vidéoconférence précédemment citées, la langue utilisée ou la taille des images participeront à l'évaluation du critère intrinsèque.
Le second critère, le **critère contextuel** regroupe les caractéristiques dépendant du contexte. Il traduit donc l'influence du contexte d'exécution sur l'application. En particulier, il décrit le respect des contraintes temporelles liées soit à l'aspect réparti avec le temps de transmission ou la synchronisation, soit aux capacités du matériel informatique comme les vitesses de traitement. Il décrit également le respect du service malgré les perturbations induites par l'environnement matériel telles que le bruit, les erreurs de transmission, ou la luminosité lorsqu'il s'agit de détecter et suivre un objet dans une image. Dans la vidéoconférence que nous avons présentée, il concerne le débit et le temps de transmission du son ainsi que la cadence vidéo.

Les caractéristiques de QdS permettent de décrire aussi bien un composant simple qu'un composite ou une application. Elles seront définies à partir des souhaits de l'utilisateur et donc seront plus ou moins nombreuses selon les cas. Le problème est alors de faire le lien entre le service spécifié par l'utilisateur, la description du service ou de ses composants et les mesures disponibles de l'environnement de manière à prévoir la QdS: il s'agit d'un problème proche de la transcription *QoS-QoS* et *QoS-ressources* selon [Hafid, 1998] mais en partant du service et en adaptant l'application et non les ressources. Nous proposons un modèle de QdS hiérarchique dans l'esprit de [Firesmith, 2003] avec le "*Groupe de facteurs de qualité orienté utilisateur*".

4.5 Représentation

Nous avons défini les deux critères indépendants qui vont servir de base pour l'évaluation de la QdS, nous allons voir comment cette dernière est représentée. Il est à noter que concernant la QdS intrinsèque cette dernière est fixée à la conception et ne peut évoluer, au contraire de la contextuelle qui bien évidemment dépend fortement du contexte et de son évolution.
Nous utiliserons une représentation à deux dimensions en octroyant une note appelée **In** au critère intrinsèque et une note appelée **Co** au critère contextuel et en représentant la QdS par un point $\mathbf{P_{QdS}}$ dont les coordonnées sont ces deux notes:

Formule 1 : $P_{QdS}=(Co, In)$

La QdS va exprimer une distance entre la qualité spécifiée par l'utilisateur considérée comme optimale car il n'est pas utile de faire mieux si l'utilisateur y est insensible et la pire, celle qui est rédhibitoire. Dans le souci de limiter la dynamique de la représentation de la QdS et de borner celle-ci entre 0 et 1 afin de simplifier les études mathématiques, nous choisissons de représenter les deux qualités extrêmes :

- 0 : service rendu rédhibitoire pour l'utilisateur ;
- 1 : service rendu attendu par l'utilisateur.

Les deux critères intrinsèques et contextuels étant indépendants, le plan de représentation de la QdS est un repère orthogonal : il représente l'ensemble des qualités de service possibles (Figure 41).

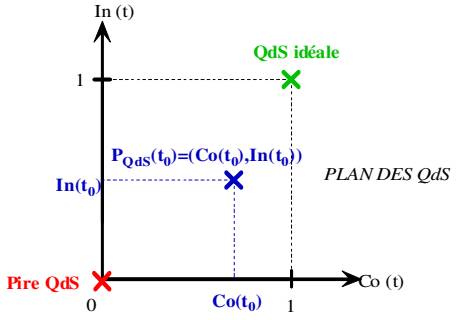


Figure 41: Représentation de la QdS à un instant t_0

La mise en œuvre pratique des applications distribuées multimédia telle que nous les concevons est réalisée à l'aide de composants logiciels éventuellement composites. Chaque composant (ou assemblage) réalisant une fonctionnalité possèdera une qualité de service représentée par l'un des items suivant :

- par un point qui traduit sa QdS à un instant précis ;
- par un ensemble de points par exemple lorsque de par sa conception un ensemble de composants ou un composant peut proposer différentes qualités intrinsèques réparties de façon discrète (E1) ;
- par une courbe qui relie l'ensemble des qualités possibles en particulier lorsque seul le contexte fait varier la QdS (E2) et ceci d'une façon continue ;
- par une surface si les deux types de caractéristiques, contextuelles et intrinsèques, ont une évolution continue (E3) ;
- un ensemble de points, courbes, surfaces, en fonction du ou des services qu'il propose et de la dépendance de ces services envers le contexte.

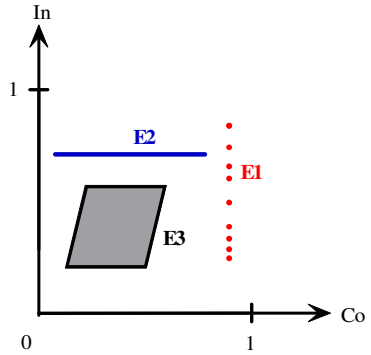


Figure 42 : Qualités de service de différents composants/assemblages

4.5.1 Propriétés mathématiques de la fonction d'évaluation

L'objectif de ce paragraphe est d'établir les propriétés mathématiques que devra respecter la fonction *f*(*Co, In*) représentant la note de QdS d'un composant ou d'un assemblage de composants en fonction des notes de ses critères intrinsèque et contextuel.

Nous souhaitons que la QdS soit normée de manière à garder la même échelle de valeurs que celles des critères. Les valeurs de *f* seront donc comprises entre 0, qualité rédhibitoire, et 1, qualité optimale. La fonction *f* doit donc respecter la première propriété P1 :

P1 : $f : [0;1] \times [0;1] \rightarrow [0;1]$

Le fait qu'une caractéristique de service dépende ou non du contexte n'a pas d'importance pour l'utilisateur. Il est sensible uniquement à la valeur, la note, de cette caractéristique. Nous supposons que l'échelle de valeurs de l'utilisateur vis-à-vis des deux critères est la même. La fonction *f* est donc symétrique par rapport aux critères et respecte la propriété P2 :

P2 : $\forall x, y \in [0;1], f(x,y) = f(y,x)$

Nous considérons que si l'un des aspects du service est rédhibitoire, l'ensemble du service est rédhibitoire. Cette propriété vient des particularités des applications multimédia : si la vidéo est transmise si lentement que l'on ne distingue pas les mouvements, toute la chaîne de transmission est inefficace car le service n'est pas rendu. De même, s'il y a trop de pertes dans la transmission du son, les paroles ne sont plus audibles ni compréhensibles. La fonction *f* possède donc un élément dont le comportement vis-à-vis de la QdS finale est similaire à celui d'un élément absorbant pour un opérateur : une note de 0 représente une qualité rédhibitoire. C'est ce qu'exprime la propriété P3 :

P3 : $\forall x, y \in [0;1], f(x,0) = f(0,y) = 0$

Si un critère de QdS est parfait, seul l'autre interviendra dans l'évaluation. Or la qualité parfaite est notée par la note 1 donc 1 est un élément dont le comportement vis-à-vis de la QdS finale est similaire à celui d'un élément neutre pour un opérateur. C'est la propriété P4 :

P4 : $\forall x \in [0;1], f(x,1) = x$ et $\forall y \in [0;1], f(1,y) = y$

Si la note de l'un des critères augmente, la qualité doit augmenter. *f* est donc une fonction croissante par rapport à chaque variable. Lorsqu'elles sont définies, les dérivées partielles de *f* par rapport à In et à Co sont positives. Ceci se traduit par la propriété P5 sur les dérivées partielles de la fonction *f* :

P5 : $\frac{\partial f}{\partial Co} \geq 0$ et $\frac{\partial f}{\partial In} \geq 0$

4.6 Évaluation de la qualité de service

Nous avons montré dans [Laplace, 2006b] qu'il y a une analogie entre notre modèle de QdS et l'utilité, entre les quantités de biens consommés et les valeurs des deux critères In et Co, entre le prix d'un bien et les ressources nécessaires à l'obtention d'une valeur donnée d'un critère. Ainsi, notre modèle de QdS dépend des deux critères comme l'utilité dépend des quantités consommées dans le cas de la consommation de deux biens. Cette analogie est résumée par le tableau suivant.

	ECONOMIE	INFORMATIQUE
Notions	Utilité	Qualité de service
	Quantité de bien	Valeur d'un critère
	Prix d'un bien *(connu)*	Ressources nécessaires à un critère *(inconnues)*
	Revenu (connu)	Totalité des ressources *(inconnue)*
Objectifs	Maximiser l'utilité (totale) pour un revenu donné mesurable et connu	Maximiser la QdS pour des ressources données (contexte) non mesurables et inconnues
Dépendances	L'utilité dépend des quantités consommées : Plus les quantités des biens consommés sont élevées, plus l'utilité est grande	La QdS dépend de deux critères In et Co : plus les valeurs de ces critères sont élevées, plus la QdS est grande

Tableau 3 : Analogie entre utilité et qualité de service

Cette analogie nous permet de définir des courbes d'indifférence pour la QdS comme on le fait pour l'utilité.
Le degré de convexité d'une courbe d'indifférence (cf. 4.2) nous renseigne sur les conditions dans lesquelles une personne est prête à substituer un bien à un autre tout en restant indifférente, ce qui se nomme degré de substituabilité. Dans le cas de la QdS, les deux paramètres de la courbe sont les critères intrinsèque et contextuel. Ces critères étant orthogonaux, une caractéristique ne dépendant pas du contexte ne peut pas être remplacée par une caractéristique dépendante du contexte. Ces critères sont donc non substituables. Par analogie avec les biens, nous dirons qu'ils sont parfaitement complémentaires. La courbe représentant les équipotentielles de QdS en fonction des critères In et Co a donc la même allure que la courbe représentant les équipotentielles d'utilité en fonction des quantités de bien *i.e.* les courbes de préférences. Une équipotentielle de la fonction *f* telle que $QoS=f(Co, In)$ est ainsi représentée sur la figure suivante.

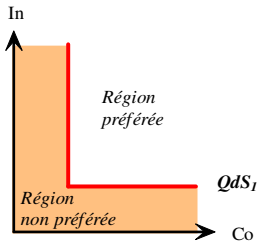


Figure 43 : Equipotentielle de qualité de service

La fonction qui permet d'évaluer la QdS dans notre modèle doit posséder les équipotentielles que nous venons de définir en plus de l'ensemble des propriétés ci-dessus.

4.6.1 Modèle d'évaluation de la qualité de service

A partir de l'analogie avec la fonction d'utilité, nous avons défini la note de QdS d'une entité à partir des notes des critères intrinsèque, In, et contextuel, Co, par la relation :

Formule 2 : $\mathbf{QdS(Co,In) = min(Co,In)}$

Le tracé des courbes de niveau de la QdS est alors celui de la Figure 44.

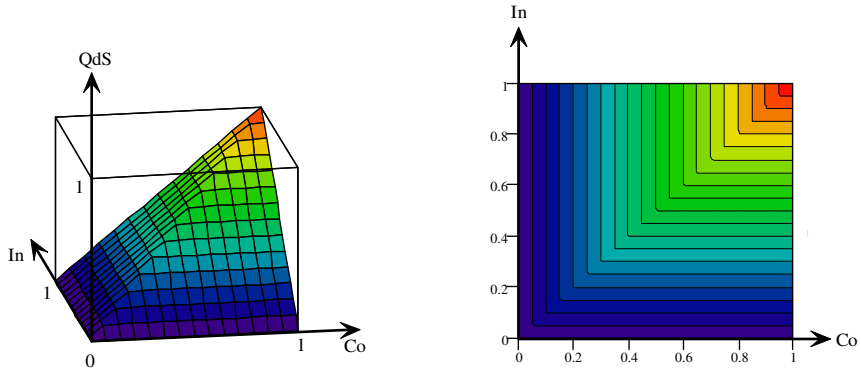


Figure 44 : Courbes de niveau (3D et 2D) de la fonction QdS

a. Propriétés de la fonction QdS : comparaison

Nous avons maintenant un outil mathématique permettant la comparaison de QdS proposée par des composants ou assemblages de composants.

La définition de la QdS à laquelle nous avons abouti nous permet de comparer la QdS de deux entités logicielles. Il est à noter que la fonction obtenue simplifie cette comparaison en particulier dans le cas où les (assemblages de) composants logiciels ne proposent qu'une seule valeur du critère intrinsèque, ce qui signifie qu'ils ne peuvent rendre le service que d'une seule manière.

Prenons l'exemple d'une configuration d'assemblage C_1 où tous les composants (ou assemblages) ne proposent qu'une seule QdS appelée QdS_1, nous n'aurons alors qu'une seule valeur In_1 du critère intrinsèque. La courbe parcourue par sa QdS lorsque le contexte varie est décrite par les figures suivantes (Figure 45).

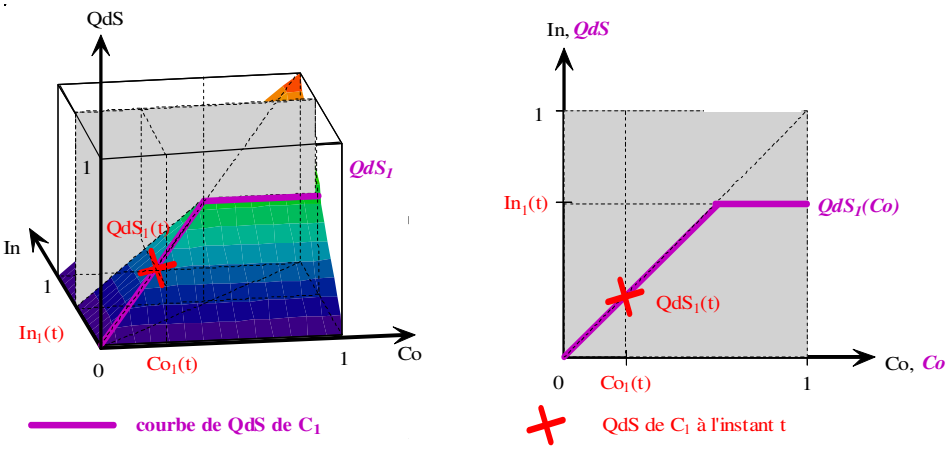


Figure 45 : Courbe de la QdS d'une entité à In constant

Prenons le cas d'une autre configuration C_2 dont la qualité de service QdS_2 est décrite par une seule valeur In_2 du critère intrinsèque. Pour savoir s'il est nécessaire de changer de configuration, il faut comparer la QdS des deux configurations pour le contexte actuel. Deux cas peuvent alors se présenter :

- soit $In_2 \geq In_1$ (cf. Figure 46a) et la QdS de la configuration C_2 peut être meilleure que celle de la configuration C_1 quelle que soit la valeur de QdS_1 ce qui implique qu'il est nécessaire d'évaluer QdS_2 pour savoir si une reconfiguration est utile ;
- soit $In_2 \leq In_1$ (cf. Figure 46b) et deux cas peuvent se produire en fonction des valeurs relatives de QdS_1 et In_2 :
  - si $QdS_1 \geq In_2$, la QdS de la configuration C_2 ne pourra pas être meilleure que celle de C_1. Il est donc inutile d'évaluer QdS_2.
  - si $QdS_1 \leq In_2$, il sera nécessaire d'évaluer QdS_2 pour savoir si la configuration C_2 est meilleure que la configuration C_1.

Les propriétés de la fonction QdS permettent donc d'éviter d'évaluer QdS_2 si $QdS_1 \geq In_2$. Cette particularité permettra d'optimiser la recherche d'une configuration meilleure lorsqu'il faudra adapter une entité logicielle.

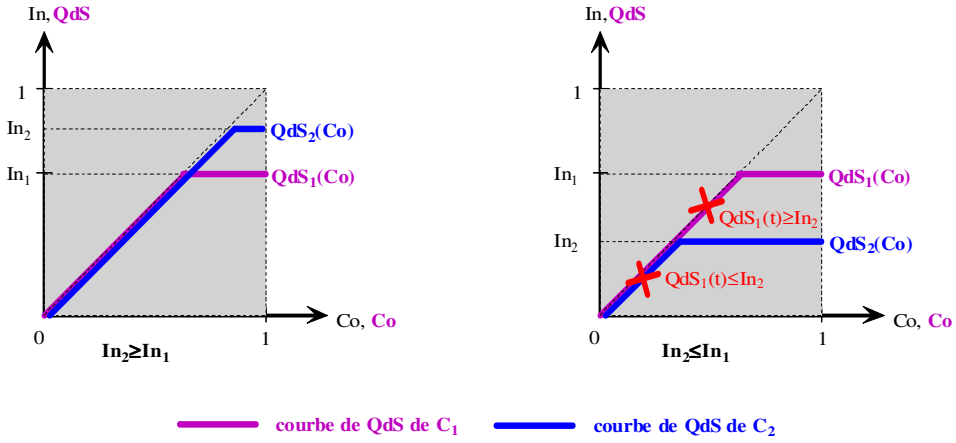


Figure 46 : Comparaison de deux configurations

4.7 Synthèse

Notre modèle souligne l'importance du contexte en intégrant un critère contextuel et un critère intrinsèque dans la définition et l'évaluation de la QdS. Ceci se traduit par un couple de valeurs entre 0 et 1 représentant les qualités rédhibitoires à optimales.
Ce modèle de QdS n'est en aucun cas suffisant en lui-même. Il n'est qu'une base théorique qui va nous servir pour l'ensemble des travaux présentés à venir. Ainsi, il induit un certain nombre de choses qui sont décrites dans le chapitre suivant :

- La mesure : la prise en compte de la qualité de service dans les applications nécessite que ces dernières aient été prévues pour. Nous allons ainsi proposer une modélisation spécifique des applications distribuées multimédia permettant de préciser les types de configurations possibles.
- Le choix d'une configuration à un instant donné peut être amené à évoluer. Il est donc nécessaire de disposer d'outils permettant de connaître le contexte, de l'évaluer et de choisir une configuration adéquate (nous verrons dans le chapitre suivant que choisir **LA** configuration la plus adéquate n'est pas possible car c'est un problème NP-Complet).

Au-delà de l'offre d'un moyen de mesure du contexte (couche 2, Figure 1), la mesure de la QdS offre une information contextuelle importante permettant aux applications de réaliser et de guider efficacement la phase de prise en compte du contexte (couche 3) et ainsi de réaliser les reconfigurations par adaptation, migration ou composition comme nous allons le voir dans le chapitre suivant.

Chapitre 5 Représentation des applications et de leur qualité de service

Le chapitre précédent s'est attaché à définir un modèle de qualité de service. Nous allons maintenant définir une modélisation des aspects fonctionnels permettant une dérivation vers l'implantation et l'évaluation de sa QdS. Nous nous situons ici clairement au niveau de la couche 2 (Figure 2) en offrant une représentation réflexive de l'application permettant les reconfigurations nécessaires réalisées à la couche 3.

5.1 Principes généraux

Afin de réaliser les reconfigurations de l'application (couche 3) selon les évolutions du contexte et dans le but d'assurer en permanence une QdS satisfaisante pour l'utilisateur, nous avons fait le choix de la supervision. Cette dernière sera assurée par la plate-forme de supervision présentée au chapitre 2 (Outils d'adaptation permettant la prise en compte du contexte). Aussi, la représentation de l'application va-t-elle être guidée par ce double objectif d'être utilisable à la fois par le concepteur et par la plate-forme.

5.1.1 Justification de la représentation

Notre choix s'est porté sur les graphes directs, polaires et orientés. C'est un mode de représentation particulièrement adapté à l'informatique mais surtout, ce sont des outils utilisés pour des problématiques proches de la nôtre [Elès, 1998] [Elès, 2000].

La modélisation des applications va se réaliser en 6 étapes, chacune d'elle représentant un aspect du modèle d'une application :

- le **graphe des flots de contrôle** représente les spécifications fonctionnelles fournies par le concepteur;
- le **graphe fonctionnel** représente le modèle fonctionnel de l'application;
- le **graphe de configuration** précise la composition et l'implantation d'une application alors que le super-graphe des configurations décrit toutes les configurations possibles;
- le **graphe d'évaluation** représente l'algorithme permettant d'évaluer la QdS de l'application tenant compte du contexte;
- le **graphe de transition** introduit de manière explicite les composants logiciels ;
- le **graphe d'implantation** fournit une vue directement implantable à l'aide du modèle Osagaia.

En fait, à chaque étape, chaque graphe va représenter une vue de plus en plus complète de l'application.

5.1.2 Symbolisme graphique utilisé pour la représentation

Pour représenter l'application, nous proposons d'adapter le Graphe des Processus [Elès, 1998] et le Graphe Conditionnel des Processus [Elès, 2000] aux applications multimédia réparties sur Internet. Nous utilisons un type de Graphe des Processus pour *le graphe des flots de contrôle*, *le graphe fonctionnel* et *le graphe de transitions*. Le graphe Conditionnel des Processus nous permet de construire *le graphe de configuration* et celui *d'évaluation*. En effet, dans ces deux derniers cas, il est nécessaire de prendre en compte les entités chargées de la communication. Or un Graphe Conditionnel des Processus est un Graphe des Processus enrichi par la représentation de la communication. Enfin, *les graphes de transitions* et *d'implantation* seront dérivés des graphes fonctionnels à l'aide de règles de transformation afin de spécifier les composants et les types de flux avec leurs politiques de synchronisation.

5.2 Représentation des différents aspects de l'application

Nous allons présenter dans ce qui suit chaque graphe avec son rôle pour arriver jusqu'au graphe d'implantation reflétant le déploiement et modélisant les échanges de flux de données avec leurs politiques de synchronisation, c'est-à-dire la représentation de l'application complète.

5.2.1 Représentation fonctionnelle

La première partie de la représentation fonctionnelle est réalisée à l'aide du graphe des flots de contrôle modélisant les spécifications fonctionnelles du concepteur puis du graphe fonctionnel donnant une décomposition fonctionnelle.

5.2.1.1 Graphe des flots de contrôle

Le graphe des flots de contrôle (également appelé graphe de précédence [Demeure, 2002]) permet au concepteur de spécifier les fonctionnalités de l'application à différents niveaux hiérarchiques et sert de base pour le graphe fonctionnel. Son objectif est de permettre la décomposition fonctionnelle, il sera donc particulièrement utile pour modéliser les Sous-Groupes. Dans ce graphe un nœud représentera un assemblage de composants, les arêtes indiquant les contraintes de précédence entre rôles. Les arêtes conditionnelles permettent également d'identifier les différentes décompositions fonctionnelles utilisables pour réaliser la fonctionnalité.
Nous noterons C_{Fi} la condition liée à une arête qui sera parcourue uniquement si le chemin considéré correspond à la décomposition fonctionnelle Fi de l'entité représentée. Ces arêtes indiquent également qu'un rôle impose l'utilisation d'un autre rôle en aval. Nous noterons O_{Ri} la condition liée à une arête qui est parcourue uniquement si le rôle R_i a été utilisé.
Ainsi dans le cas du Sous-Groupe chargé de la transmission de l'image (cf. Figure 47) dans la vidéoconférence, supposons que deux décompositions fonctionnelles soient utilisables, l'une — C_{F1} — transmettant directement l'image après réduction de sa taille, l'autre — C_{F2} — utilisant une compression et une décompression avec réduction de taille. Le choix de l'une ou l'autre des décompositions sera lié aux contraintes sur le temps de traitement. En fonction de la décomposition fonctionnelle utilisée, le format des images sera différent. Le rôle de traitement sera alors implanté à l'aide de composants différents. L'utilisation d'une compression impose l'utilisation en aval d'une décompression ce qu'indique la condition O_{R4} dans la condition liée à l'arête entre le rôle de traitement — r_2 — et le rôle de décompression — r_5.

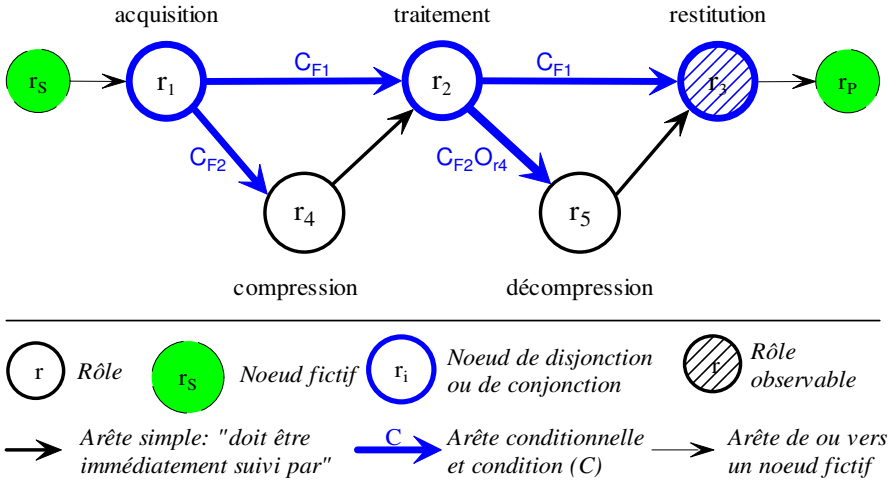


Figure 47: Exemple de Graphe des flots de contrôle du Sous-Groupe Image

Cette représentation ne permet pas d'identifier les échanges de flux d'informations. Elle permet de savoir ce qui doit être terminé avant d'entamer le traitement suivant. Il est donc nécessaire d'affiner ce graphe pour obtenir un graphe fonctionnel où les arêtes signifient "*reçoit des données de*" et constitue un graphe des flux entre rôles.

5.2.1.2 Graphe fonctionnel

Dans le champ d'application qui nous cerne, il est à noter qu'il y a une relation d'équivalence entre la précédence et l'échange de données. En effet, les applications que nous considérons ne traitent que des

flux de données. Ainsi, alors que les arêtes du graphe des flots de contrôle indiquent l'ordre d'exécution des rôles, celles du graphe fonctionnel précisent les échanges de données et permettent ainsi d'identifier les composants ayant plusieurs flux ou données en entrée ou en sortie. De plus, ces deux graphes diffèrent également par le fait que les nœuds du graphe des flots de contrôle représentent aussi bien le rôle d'un composant simple que celui d'un composite — noté par une lettre r minuscule — alors que les nœuds du graphe fonctionnel représentent uniquement les rôles atomiques — notés par une lettre majuscule R. Le graphe fonctionnel est dérivé du graphe des flots de contrôle. Ces arêtes conditionnelles indiquent donc de la même façon les choix de configurations et les contraintes d'utilisation de certains rôles en plus d'indiquer un échange de données.
Le graphe fonctionnel du Sous-Groupe Image de la Figure 47 est donné par la Figure 48. En fonction du format de l'image, la réduction de la taille est réalisée par un composant de rôle R_{2a} ou deux composants de rôles $R_{2b\text{-}1}$ et $R_{2b\text{-}2}$. Tous les rôles utilisés sont atomiques. Le choix entre les deux décompositions fonctionnelles – C_{F1} ou C_{F2} — impose le ou les rôles atomiques réalisant la réduction de la taille de l'image. Notons que la définition des rôles atomiques ne peut être faite qu'en connaissance des composants disponibles pour concevoir l'application. De plus, la nécessité de fournir simultanément à des utilisateurs distincts des qualités d'images différentes issues d'une unique caméra impose de différencier les rôles atomiques et donc les composants d'acquisition, de compression et de traitement.

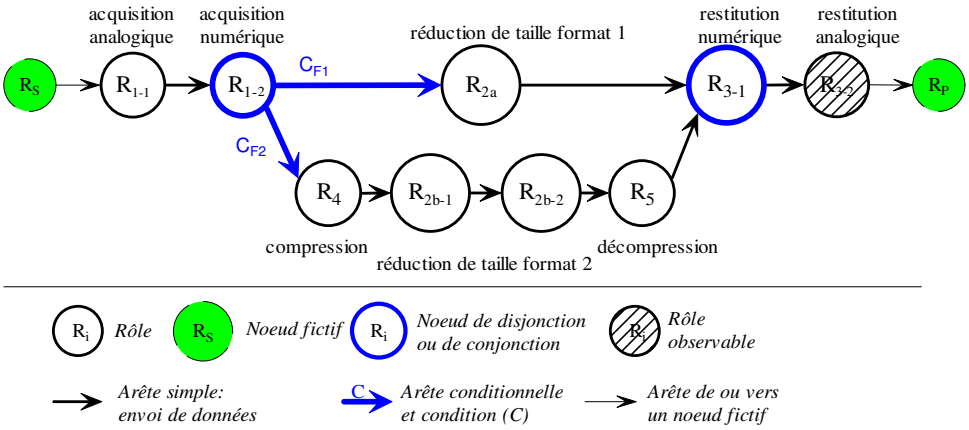


Figure 48: Exemple de Graphe fonctionnel du Sous-Groupe Image

5.2.2 Représentation des configurations

Afin de répertorier de manière exhaustive l'ensemble des configurations possibles, nous passons d'abord par une étape définissant un super-graphe des configurations [Li, 2001] [Cui, 2001].

5.2.2.1 Super-graphe des configurations

Ce graphe précise, pour chaque configuration possible, les composants, les Processeurs Élémentaires, leurs localisations, les Conduits utilisés pour les échanges de flux et les connexions. Il utilise pour ce faire le graphe fonctionnel. Ses arêtes conditionnelles indiquent les choix de décompositions fonctionnelles. En revanche, ce ne sont plus les contraintes d'utilisation des rôles qu'elles précisent mais celles concernant les composants — notées O_{Ci} : le choix d'un composant pour un rôle impose un composant pour un autre rôle comme dans le cas des compressions et décompressions. L'arête conditionnelle est utilisée si le second composant n'est pas relié au premier par un chemin simple c'est-à-dire un chemin direct ou un chemin sans disjonction. Ainsi dans le graphe des flots de contrôle du Sous-Groupe Image, il est nécessaire d'utiliser une arête conditionnelle pour exprimer le lien entre les rôles de compression — r_4 — et de décompression — r_5 — car le rôle représenté par le nœud r_2 (Figure 47) est un lieu de conjonction et de disjonction. En revanche, dans le graphe fonctionnel du Sous-Groupe Image, un chemin direct relie les rôles atomiques de compression et décompression si bien que le choix de la compression — C_{F2} — impose automatiquement l'usage d'une décompression.

5.2.2.2 Graphe d'une configuration

Une configuration possible correspond à un sous-graphe du super-graphe de configuration auquel n'est plus associée une expression logique comme dans [Elès, 2000]. En effet, un label est défini pour un chemin exécuté. Or les configurations de l'application ou d'un Groupe ne correspondent pas forcément à

un seul chemin : elles ne fourniraient qu'une seule fonctionnalité à l'utilisateur. C'est pourquoi une configuration de l'application, d'un Groupe ou d'un Sous-Groupe — représenté par le super-graphe des configurations — est définie par l'ensemble des labels qui déterminent l'ensemble des chemins pris entre la source et le puits (sommets fictifs de début et de fin de graphe). Une configuration peut donc être définie par le langage ou le sous-graphe $\mathbf{G_k}$ que nous nommons **graphe de configuration.**
Les arêtes conditionnelles permettent également de préciser quel choix de composant est fait pour implanter un rôle atomique. Nous noterons C_{Ci} la condition liée à une arête qui ne sera parcourue qu'à condition que le composant C_i soit choisi pour réaliser le rôle suivant. Lorsqu'il n'y a qu'un choix de composant pour implanter un rôle, on utilisera une arête simple.
Enfin, dans les graphes utilisés en cours d'exécution par la plate-forme, les nœuds représentent d'une part, les Processeurs Élémentaires et les Conduits pour les composants et flux. Nous utilisons une représentation permettant d'identifier les flux traités par un composant encapsulé dans un Processeur Élémentaire et ceux synchronisés par cette entité (cf. Figure 49).

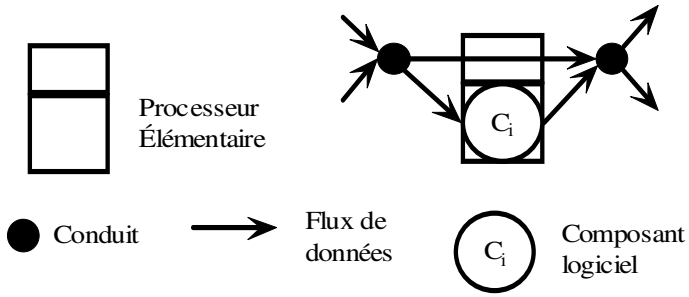


Figure 49 : Représentation de Processeur Élémentaire, de Composant et de Conduit

Enfin, le super-graphe des configurations doit permettre de représenter l'intégralité des composants de l'application en identifiant leur Groupe et leur Sous-Groupe. C'est pourquoi les arêtes conditionnelles sont également utilisées pour préciser l'appartenance d'un flux à un Sous-Groupe ou un Groupe. Dans ce cas, les conditions associées sont notées I_{Gi} pour un Groupe et I_{SGi} pour un Sous-Groupe.

5.2.2.3 Exemple du Sous-Groupe Image

La Figure 50 représente le super-graphe des configurations du Sous-Groupe Image avec les deux configurations possibles de la Figure 48, C_{F1} et C_{F2}, et les implantations possibles entre le poste du locuteur P_1 et celui du téléspectateur P_4. Les composants de traitement peuvent être situés indifféremment sur l'un ou l'autre des deux postes d'où la présence des conditions P_1 et P_4 sur les arêtes conditionnelles. Pour les traitements, deux composants peuvent être utilisés en fonction de leur compatibilité avec la présence ou l'absence de compression.

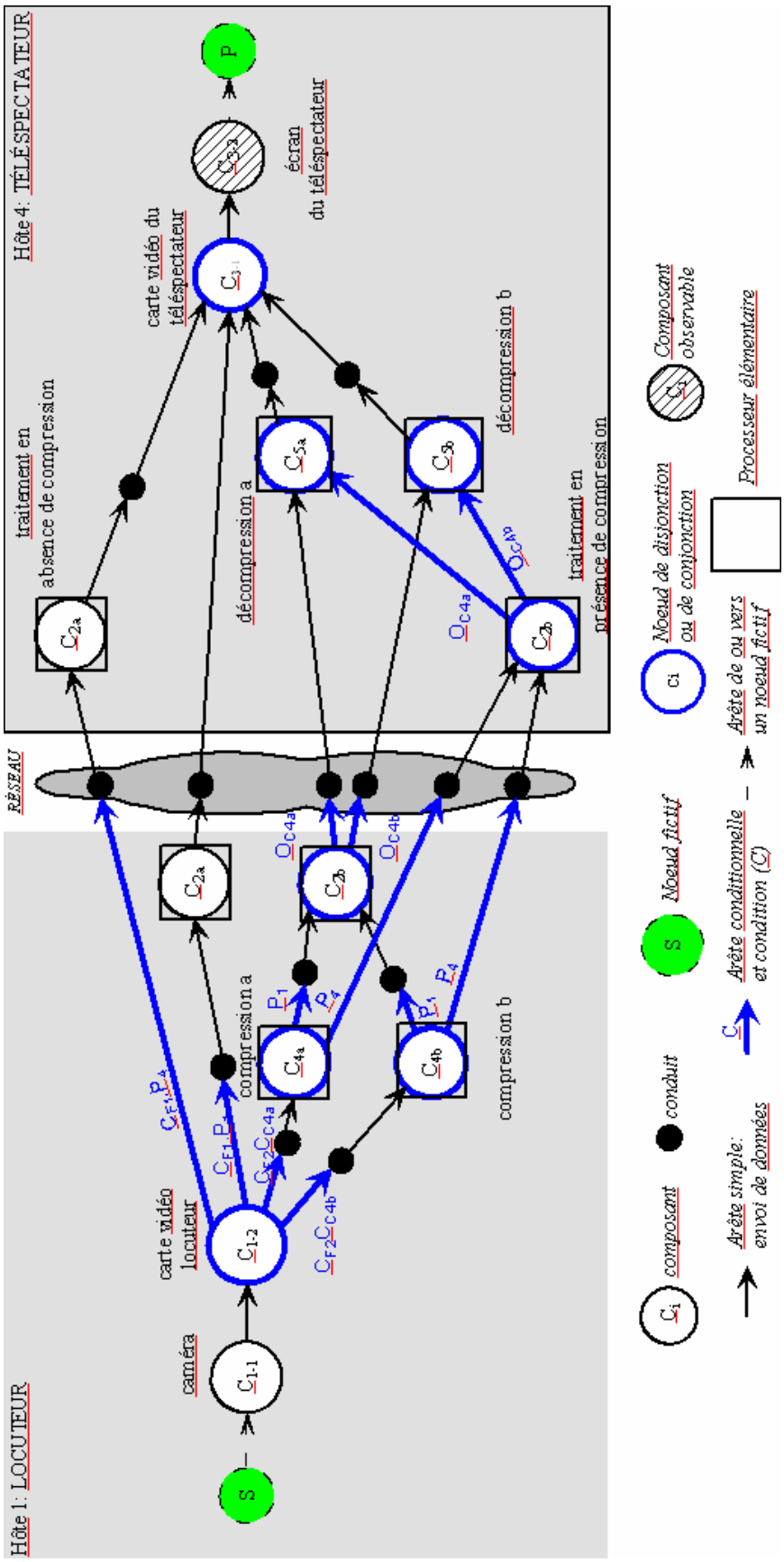


Figure 50: Exemple de Super-Graphe des configurations du Sous-Groupe Image

5.3 Synthèse de la représentation fonctionnelle

Les graphes précédemment introduits indiquent trois types d'informations :

- des choix amenant à l'utilisation d'une configuration particulière de l'application : choix d'une décomposition fonctionnelle, choix d'un composant, choix d'une localisation ;
- des contraintes de dépendance : un rôle en impose un autre, un composant en impose un autre ;
- des informations nécessaires à l'identification des Groupes et Sous-Groupes : appartenance de l'arête à tel Groupe ou Sous-Groupe.

Nous avons regroupé sur le graphe des configurations, l'ensemble des informations relatives au découpage essentiellement fonctionnel de l'application en ayant présenté de manière exhaustive les différents assemblages possibles.

5.4 Représentation de la qualité de service d'une application

La représentation de la QdS va utiliser les mêmes outils (graphes) que précédemment. Cette étude a été inspirée des méthodes d'évaluation du temps d'exécution de systèmes embarqués distribués. Le résultat est ensuite représenté par des notes prenant en compte à la fois les critères intrinsèques des composants mais également les critères contextuels relatifs à leur contexte d'exécution.

5.4.1 Représentation de l'évaluation

Nous présentons tout d'abord les raisons qui nous ont amenés à utiliser des graphes et des vecteurs pour représenter la QdS. Elles sont liées aux caractéristiques des applications multimédia et du modèle de QdS. Puis nous décrirons les graphes d'évaluation ainsi que leurs propriétés.

Évaluer la QdS d'une application, c'est évaluer à la fois le critère intrinsèque regroupant les caractéristiques de qualité indépendantes du contexte et le critère contextuel regroupant celles dépendant du contexte.
Or les caractéristiques de l'application regroupent celles des services — les Groupes — et celles des fonctionnalités constituant ces services — les Sous-Groupes. Définir les caractéristiques de l'application consiste donc à définir celles des Sous-Groupes et à les assembler pour obtenir celles des Groupes puis enfin celles de l'application elle-même.
Nous avons proposé de structurer les caractéristiques de QdS d'un Sous-Groupe en un **vecteur de QdS** [Li, 2001] contenant les caractéristiques intrinsèques et les caractéristiques contextuelles.
Les caractéristiques d'une fonctionnalité (*Sous-Groupe*) dépendent du contexte bien évidemment, mais également des flux reçus par la composition fonctionnelle de ce Sous-Groupe (l'assemblage des composants) et des spécificités définies par le concepteur. Les caractéristiques de QdS des composants en amont dépendent, elles-aussi, des flux reçus et des spécificités de ces composants (ou assemblages). Ainsi, de proche en proche, nous voyons que les caractéristiques de QdS du Sous-Groupe dépendent des spécificités des composants et des flux constituant le Sous-Groupe, reflets du contexte d'exécution.
Comme nous l'avons expliqué, les composants métier sont encapsulés dans un container (le Processeur Élémentaire), les flux sont transmis à l'aide des Conduits ; ces deux entités assurent la politique de communication, en l'occurrence dans notre champ d'étude, la synchronisation.
Ainsi, la QdS sera représentée par un graphe où les nœuds indiquent les modifications que les Processeurs Élémentaires (*composants*) et les Conduits (*flux*) imposent au vecteur de QdS représenté par les arêtes. Nous obtenons alors un graphe d'évaluation de la QdS.

5.4.1.1 Graphes d'évaluation

Les arêtes représentent ici les qualités de service obtenues en entrée ou en sortie des Processeurs Élémentaires et des Conduits qui sont modélisées par des vecteurs de QdS (cf. Figure 51). Nous utilisons ici une représentation très proche de celle proposée par [Li, 2001]. Nous l'adaptons à notre problématique en supprimant la gestion des ressources et en utilisant notre définition de la QdS. De plus, les nœuds ne représentent plus les composants mais leur influence sur la QdS de sorte que le graphe ne modélise pas l'application mais l'évaluation de la QdS.
Dans le graphe d'évaluation, les nœuds représentent l'influence des composants et des Conduits, donc d'une partie du contexte, sur la QdS (cf. Figure 51). Plus précisément, il s'agit de concrétiser ces influences par un traitement informatique ou mathématique appliqué aux vecteurs de qualités de service. Le principe

est le même que celui des règles de composition de paramètres [Wang, 1996]. L'influence d'un composant pourra être donnée par un tableau de valeurs définissant sa caractéristique de sortie en fonction de celle d'entrée et du contexte. Nous nous rapprochons alors de la notion de composant telle qu'elle apparaît en électronique analogique où les circuits sont également réalisés par assemblage de composants.

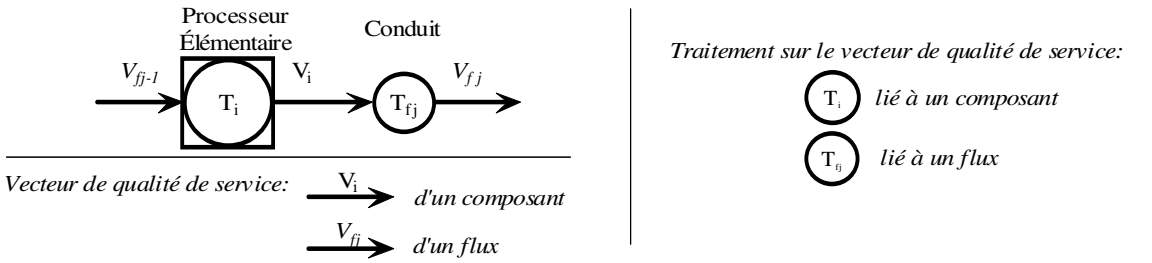


Figure 51 : Qualité de service d'un composant et d'un Conduit

Déterminer la QdS nécessite tout d'abord de définir le premier vecteur de QdS et surtout sa composition [Li, 2001]. Il contient à la fois les caractéristiques contextuelles et intrinsèques de la QdS comme la taille de l'écran, la résolution, etc.

Dans le cas des applications multimédia, il contiendra au moins le temps de traitement et le débit car ces caractéristiques sont critiques pour de telles applications. Les autres composantes du vecteur seront définies en fonction du service attendu et des composants utilisés. Pour définir la QdS, il est ensuite nécessaire de définir les différents traitements, les nœuds. Le parcours du graphe permet alors de calculer le vecteur de QdS du composant observable et donc les caractéristiques de QdS qui seront évaluées par l'utilisateur.

Les figures suivantes reprennent l'exemple de la Figure 50. La Figure 52 représente la configuration définie par L_1=$C_{F1}P_1$ c'est-à-dire que le Sous-Groupe transmet les images sans compression mais après traitement.

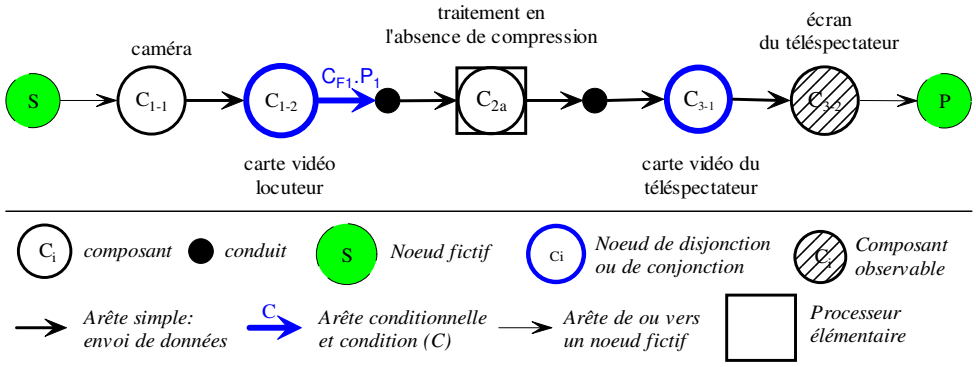


Figure 52: Exemple de configuration du Sous-Groupe Image à évaluer

La Figure 53 décrit l'évaluation de la QdS de la configuration L_1=$C_{F1}S_1$ du Sous-Groupe Image illustrant l'isomorphisme entre les deux représentations.

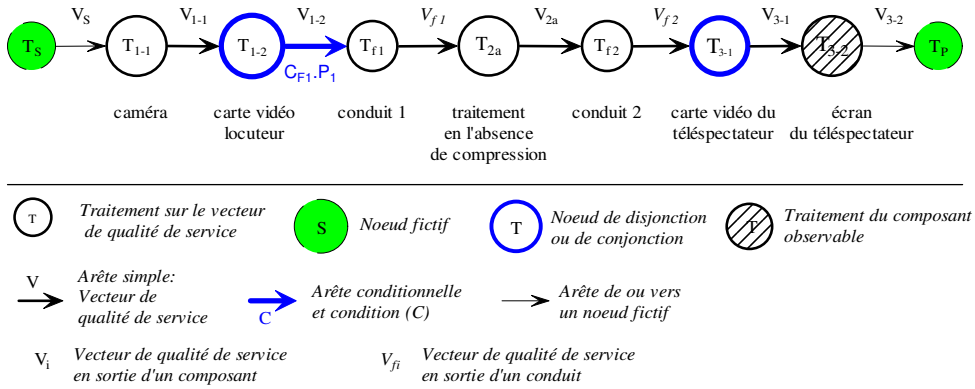


Figure 53: Exemple de Graphe d'évaluation de la qualité de service du Sous-Groupe Image

Enfin, la Figure 54 illustre la définition des valeurs des différents vecteurs prenant en compte le contexte machine ainsi que les traitements qui permettent de définir la QdS du Sous-Groupe Image. Certains temps de traitement sont négligés dans cet exemple comme le temps de transfert d'information sur un même

poste ou le temps de transit par une carte vidéo. Le traitement associé au nœud fictif source permet de définir les caractéristiques de QdS du vecteur de qualité en début de graphe. Le nœud puits traduit la qualité fournie par le Sous-Groupe. Les valeurs intrinsèques sont définies a priori (le composant offre une résolution, propose un codage de couleurs) alors que les valeurs contextuelles sont définies en cours d'exécution (temps de traitement, cadence vidéo, etc.).

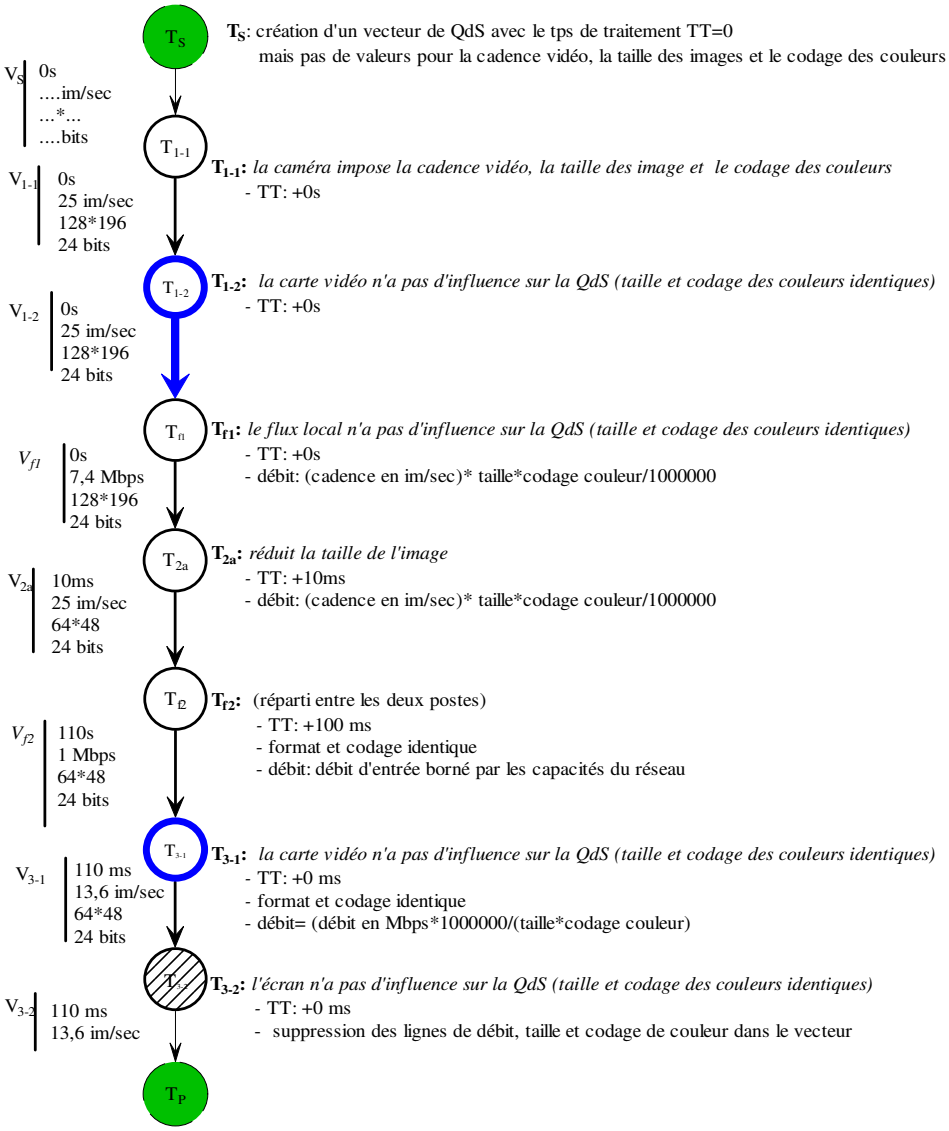


Figure 54 : Exemple d'évaluation du Sous-Groupe Image

Les graphes de configuration et d'évaluation identifient les nœuds de disjonction et de conjonction permettant l'évaluation de la QdS comme nous allons le voir.

5.4.1.2 Intérêt des nœuds de disjonction et de conjonction pour l'évaluation de la qualité de service

Le graphe d'évaluation dérive du graphe de configuration et les nœuds de conjonction/disjonction ont donc été identifiés lors de la définition de la configuration c'est-à-dire du choix des composants et de leur implantation. Connaître les parties d'évaluation communes à différentes configurations permet de ne pas redéfinir les caractéristiques de QdS. L'évaluation peut ainsi être optimisée.

De manière générale, il est possible de définir les parties du graphe d'évaluation communes aux configurations comme les parties en amont d'un nœud de disjonction et en aval d'un nœud de conjonction : les traitements de ces zones sont identiques pour toutes les configurations.

La Figure 55 reprend l'exemple de la Figure 53 et identifie les parties d'évaluation communes à toutes les configurations soit uniquement par leurs traitements soit également par leurs vecteurs.

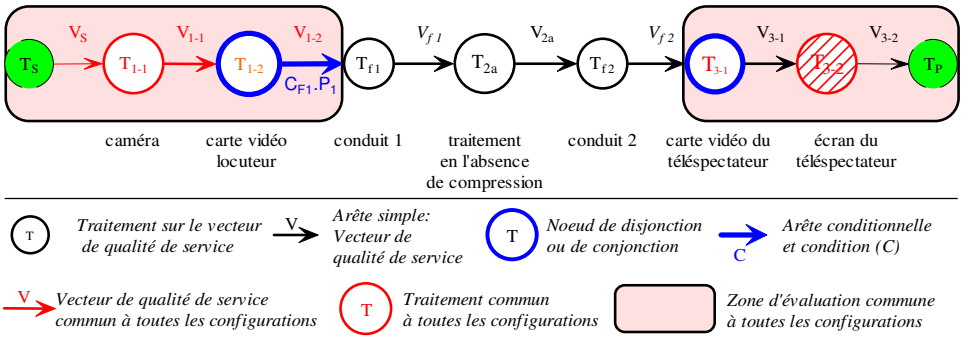


Figure 55: Exemple d'évaluations communes à toutes les configurations

5.4.2 Notation de la qualité de service aux différents niveaux structurels

Les graphes que nous avons présentés permettent de définir les caractéristiques de QdS des Sous-Groupes de l'application. En comparant ces caractéristiques avec les vœux de l'utilisateur nous pouvons octroyer une note de QdS aux Sous-Groupes, puis aux Groupes et enfin à l'application. Nous allons définir ici notre façon de nommer et de représenter ces notes avant d'expliquer comment nous les obtenons.

5.4.2.1 Nomenclature des notes de qualité de service

Il est nécessaire de définir les notes In (intrinsèques) et Co (contextuelles) des Groupes et Sous-Groupes. Le schéma présenté Figure 56 indique la notation que nous utilisons par la suite pour représenter l'ensemble des notes. Pour chaque note, nous précisons entre parenthèses si elle dépend de la configuration de l'application — i —, du Groupe considéré — j — ou du Sous-Groupe considéré — k — , du temps — t —, de la configuration utilisée pour le Groupe — g_J — ou pour le Sous-Groupe — h_K. Ainsi dans la note $In_i(i)$ l'indice i indique que cette note concerne la configuration référencée i et le i entre parenthèses indique que cette note dépend de la configuration i étudiée (l'indice est le même car se rapportant à la même configuration).

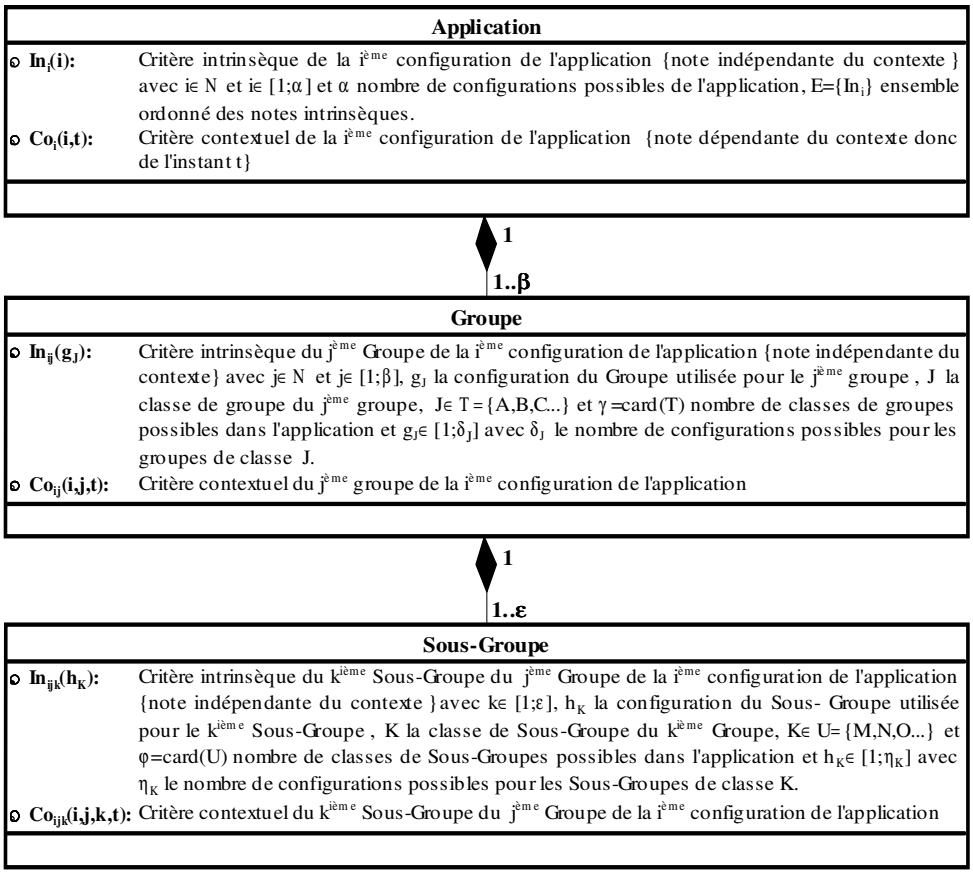


Figure 56 : Notes de qualité de service des constituants de l'application

Les composants n'apparaissent pas dans cette figure puisque nous ne proposons d'évaluer que les services fournis ce qui correspond à des assemblages de composants. Nous allons maintenant préciser comment ces notes sont obtenues.

5.5 Mise en œuvre des modèles d'application et de qualité de service dans la méthode de conception

Nous exploitons la plate-forme d'exécution précédemment présentée qui utilise notre modèle de qualité de service et notre modèle d'application pour superviser l'application et en optimiser la qualité de service.
Mais pour arriver à une exploitation via la plate-forme nous devons concevoir les applications pour cette cible.
La méthode de conception se décompose en une suite d'étapes identifiant tout d'abord la structure de l'application, puis la spécification des contraintes fonctionnelles, des contraintes de synchronisation et des composants disponibles, et enfin permet d'établir l'éventail des configurations possibles pour l'application. Il s'agit donc pour le concepteur de définir ce dont il dispose, ce qu'il souhaite faire et ce qu'il pourra faire.

5.5.1 Phase 1: Définition de la structure

La définition de la structure se résume en 5 étapes :

ETAPES		
n°	**Définition**	**Résultat**
1	Identification des classes de Groupes (service, utilisateurs)	Configuration canonique de l'application en terme de Groupe
2	Identification des classes de Sous-Groupes (fonctionnalités)	Configuration canonique des Groupes en terme de Sous-Groupes
		Graphe des configurations des groupes en terme de Sous-Groupes
3	Identification des rôles des composants nécessaires	Graphe des flots de contrôle des Sous-Groupes
		Graphe fonctionnel des Sous-Groupes (intermédiaire)
4	Inventaire des composants disponibles	Liste des composants
		Graphe fonctionnel des Sous-Groupes

Tableau 4: Étapes de la méthode de conception

Pour illustrer notre méthode, nous décrivons l'une des façons dont un concepteur peut construire l'exemple de vidéoconférence présenté précédemment.

Dans la première étape, le concepteur définit deux classes de Groupes :

- le Groupe Locuteur utilisé pour les intervenants de la conférence qui devront disposer d'une caméra motorisée et d'un microphone sur les postes concernés;
- le Groupe Téléspectateur, muni d'un téléphone portable ou équipé d'un ordinateur personnel, représente les utilisateurs souhaitant assister aux débats.

Le concepteur peut donc définir la configuration canonique de l'application en terme de Groupe (cf. Figure 57).

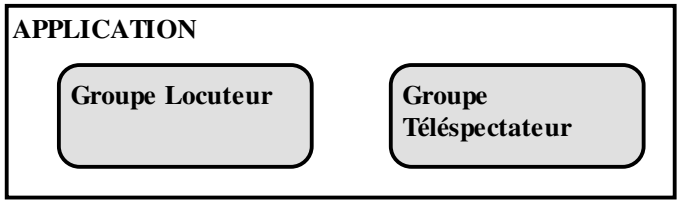


Figure 57 : Configuration canonique de l'application de vidéoconférence

Puis la deuxième étape permet de définir trois classes de Sous-Groupes :

- le Sous-Groupe Son qui transmet le son de la vidéoconférence;

- le Sous-Groupe Image qui en transmet les images ;
- le Sous-Groupe Suivi qui réalise le suivi des déplacements du locuteur par la caméra motorisée.

Il détermine alors les différents assemblages de Sous-Groupes possibles pour réaliser un Groupe :

- un seul assemblage pour le Groupe Téléspectateur composé d'un Sous-Groupe Son et d'un Sous-Groupe Image ;
- deux assemblages pour le Groupe Locuteur composé, d'un Sous-Groupe Son et d'un Sous-Groupe Image ou d'un Sous-Groupe Son, d'un Sous-Groupe Image et du Sous-Groupe Suivi.

Pour chaque Groupe, le concepteur peut définir la configuration canonique (cf. Figure 58 et Figure 59) ainsi que les différents graphes des configurations en terme de Sous-Groupes.

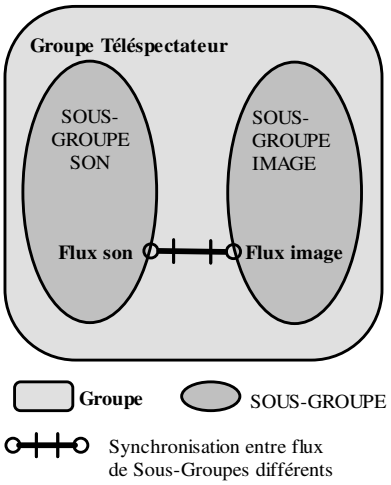


Figure 58 : Configuration canonique du Groupe Téléspectateur

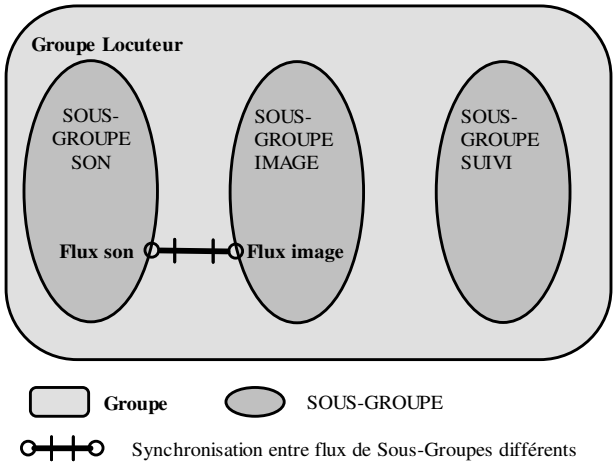


Figure 59 : Configuration canonique du Groupe Locuteur

5.5.1.1 Utilisation des graphes

Le concepteur définit tout d'abord le graphe des flots de contrôle de chaque Sous-Groupe en partant par exemple du rôle observable et en précisant ses prédécesseurs directs ou indirects (cf. Figure 60 et Figure 61). A ce niveau de définition, les rôles ne sont pas atomiques et peuvent correspondre à un ensemble de composants. La granularité des rôles d'un graphe reflète alors la granularité de l'analyse fonctionnelle descendante dont l'aboutissement fournit la définition des rôles atomiques.

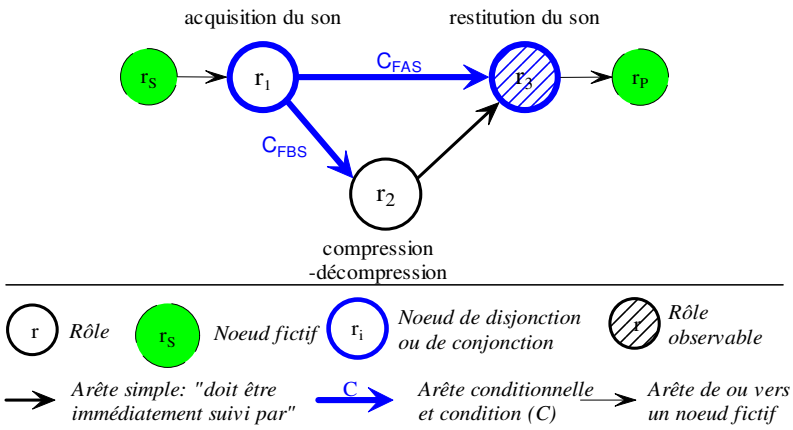


Figure 60 : Graphe des flots de contrôle du Sous-Groupe Son

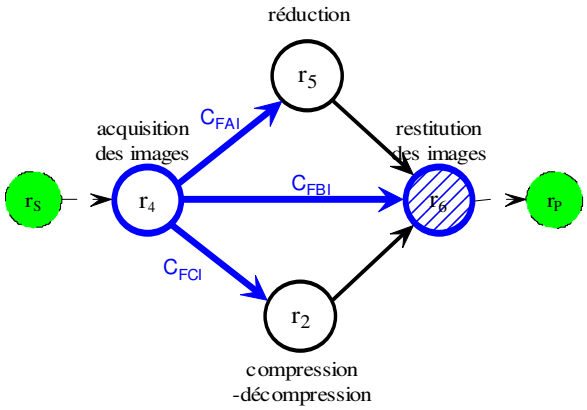


Figure 61 : Graphe des flots de contrôle du Sous-Groupe Image

Il peut ensuite décrire chaque rôle séparément par un assemblage de rôles de granularité plus fine qu'il insère dans le graphe. Puis il indique, pour chaque arête symbolisant la précédence des rôles, si elle doit être remplacée par zéro (*cas du nœud source*), un ou plusieurs flux de données. Finalement, au fur et à mesure que le concepteur définit les composants dont il dispose, il identifie les rôles atomiques et ceux pour lesquels il faudra concevoir ou acquérir un composant à moins qu'il ne choisisse de le réaliser par un assemblage de composants. Lorsque tous les rôles du graphe sont atomiques, le graphe est appelé graphe fonctionnel (cf. Figure 62, Figure 63) et le concepteur sait qu'il dispose d'au moins une configuration permettant à l'application de s'exécuter. La phase suivante lui permet de la définir.

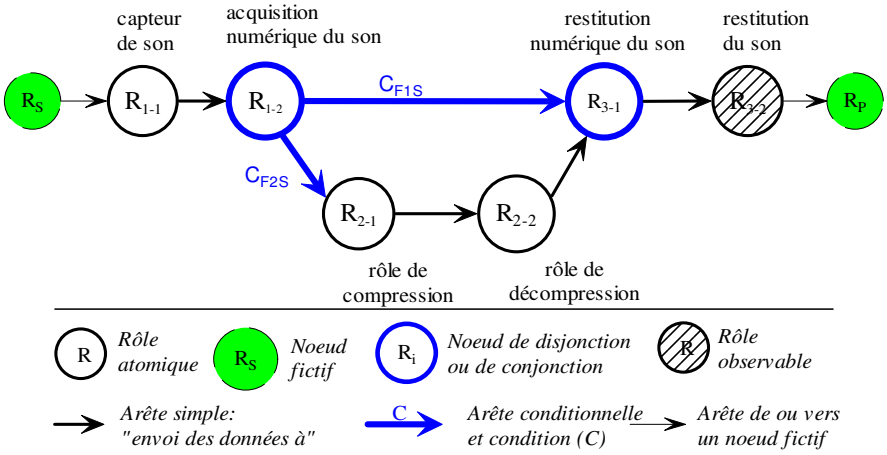


Figure 62 : Graphe fonctionnel du Sous-Groupe Son

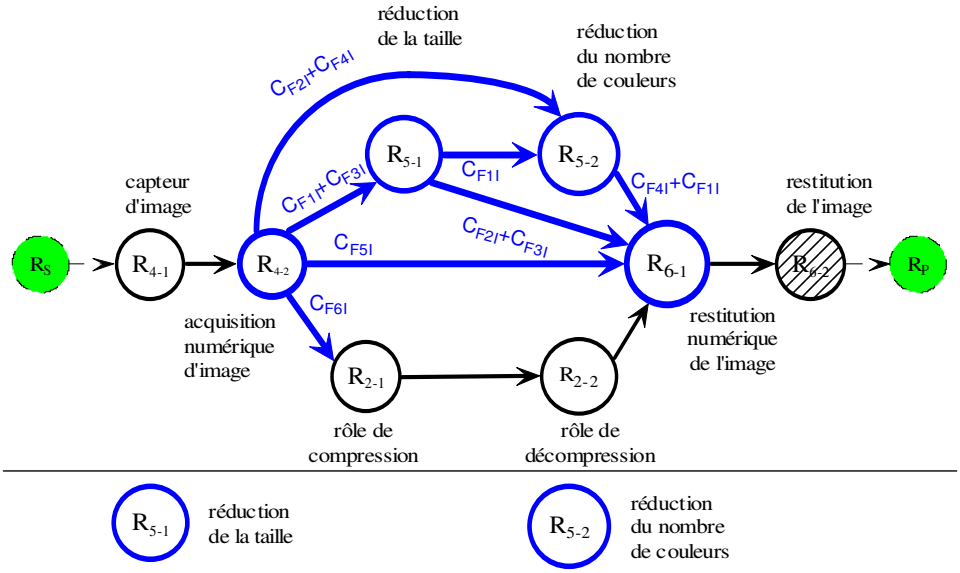


Figure 63 : Graphe fonctionnel du Sous-Groupe Image

5.5.2 Phase 2: Définition des différentes configurations de l'application

Cette phase se décompose en 4 étapes centrées sur la définition des différentes configurations de l'application. Elle permet d'obtenir un graphe canonique dans la mesure où il contient les composants permettant de réaliser tous les Groupes possibles. Il restera encore une étape puisque à ce stade-là les différents postes disponibles ne sont pas connus.

ETAPES		
n°	**Définition**	**Résultat**
5	Décomposition fonctionnelle de l'application	Graphe fonctionnel de l'application
6	Ajout des conduits	Graphe des configurations (intermédiaire)
7	Introduction des flux dans les processeurs élémentaires	Graphe des configurations (intermédiaire)
8	Identification des choix de composants	Graphe des configurations

Tableau 5: Définition des différentes configurations de l'application

5.5.2.1 Transmission et synchronisation

Il reste maintenant à définir quels flux doivent transiter par des Conduits afin d'assurer la synchronisation des données.

La première étape de cette phase va consister à définir le graphe fonctionnel de l'application. Pour cela, le concepteur dispose du graphe fonctionnel de chaque Sous-Groupe, dont les nœuds sont les rôles atomiques, et d'une liste de composants disponibles. Pour chaque Groupe, il peut donc construire un graphe canonique fonctionnel en regroupant les graphes des Sous-Groupes et le graphe de l'application en assemblant les Groupes. Enfin, il précise quels flux du graphe doivent être synchronisés c'est-à-dire quels flux transiteront par des Conduits et des Processeurs Élémentaires.

Ainsi dans notre exemple de vidéoconférence, le concepteur établit le graphe fonctionnel de la décomposition fonctionnelle du Groupe Téléspectateur (cf. Figure 64 p.74) qui réunit la décomposition C_{F1S} du Sous-Groupe Son (cf. Figure 62 p.72) et la décomposition C_{F3I} du Sous-Groupe Image (cf. Figure 63 p.73) pour construire le graphe de l'une des configurations correspondantes (cf. Figure 65 p.74).

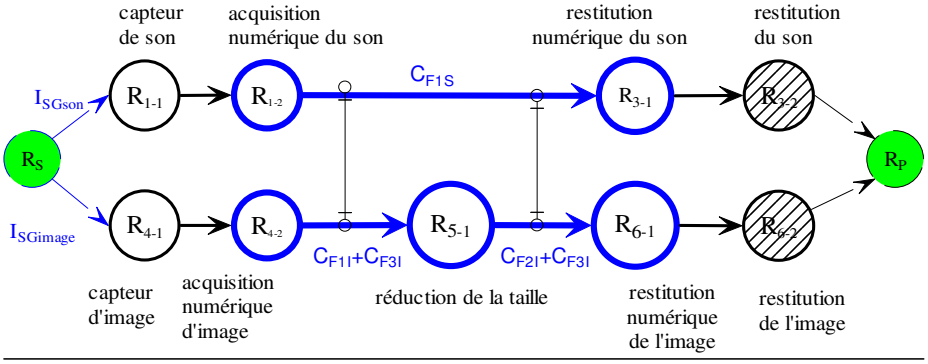


Figure 64 : Graphe fonctionnel d'une décomposition du Groupe Téléspectateur

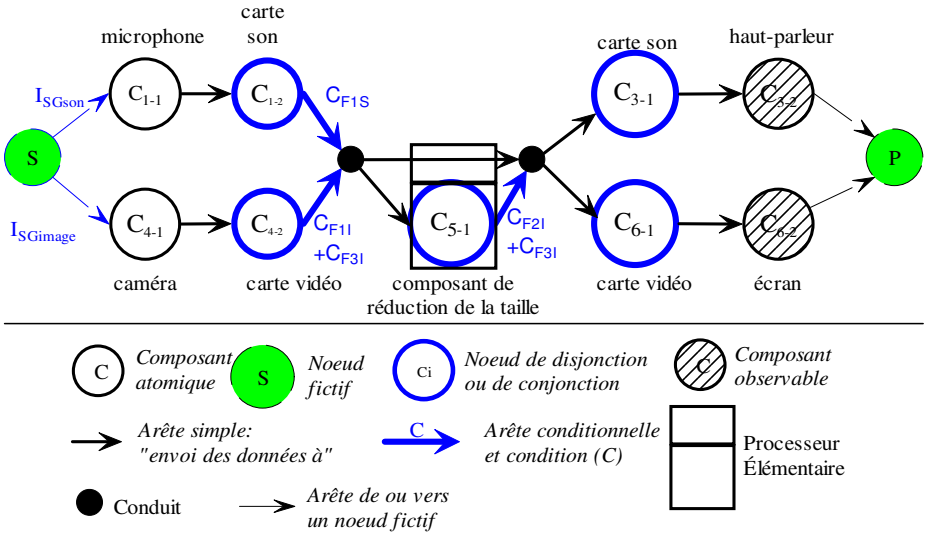


Figure 65 : Graphe d'une configuration du Groupe Téléspectateur

Nous devons ainsi définir des règles de réécriture du graphe fonctionnel permettant de regrouper les flux d'informations dans des Conduits. De plus, lorsque certains flux d'un Conduit doivent transiter par un composant, les autres flux traverseront le Processeur Élémentaire encapsulant ce composant afin d'en conserver le synchronisme. Ces règles sont résumées par les schémas de la Figure 66.

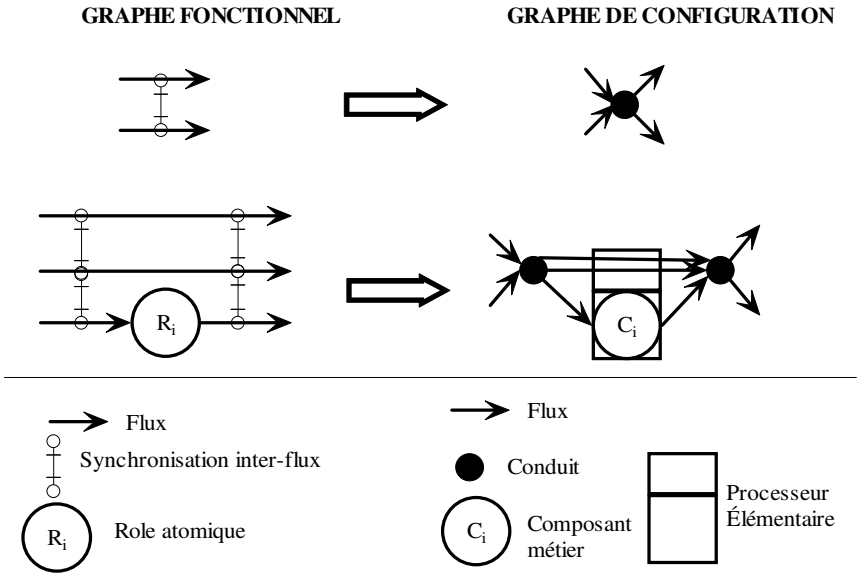


Figure 66 : Règles d'insertion des Conduits et des Processeurs

5.5.2.2 Choix des composants

Le concepteur précise ensuite quels assemblages de composants sont utilisables dans un super-graphe des configurations de l'application. Celui-ci restera partiel puisque les différents postes disponibles pour le déploiement ne sont, bien entendu, pas connus lors de la conception.

La tâche consiste à remplacer un rôle atomique soit par une arête simple si un seul composant réalise ce rôle soit par des arêtes conditionnelles si plusieurs composants peuvent le réaliser. Chaque composant logiciel sera ensuite encapsulé dans un Processeur Élémentaire.

Dans notre exemple de vidéoconférence, un super-graphe partiel des configurations du Sous-Groupe Image (cf. Figure 67) décrit ainsi plusieurs configurations pour la décomposition fonctionnelle C_{F1I} (cf. Figure 63) qui se distinguent par les composants réalisant la réduction de la taille de l'image et le nombre de couleurs utilisées : dans l'un des cas un composant est nécessaire — il réalise les deux rôles $R_{5\text{-}1}$ et $R_{5\text{-}2}$ — et dans l'autre deux composants sont nécessaires — un pour chaque rôle.

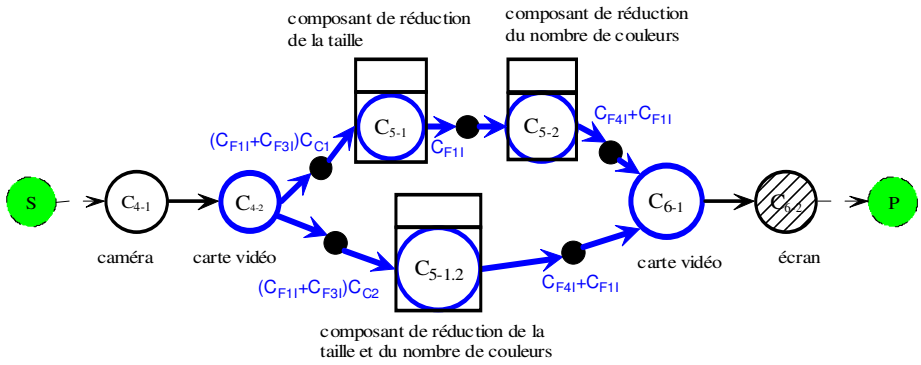


Figure 67 : Super-graphe partiel des configurations du Sous-Groupe Image

Le graphe obtenu est canonique dans la mesure où il contient les composants permettant de réaliser tous les Groupes possibles. Pour chaque Groupe, il définit les différentes configurations en termes de composants à l'aide des arêtes conditionnelles. A ce stade de la conception, l'implantation n'est pas encore définie puisque les différents postes (les hôtes) disponibles ne sont pas connus.

5.6 Graphe de transition

Ce paragraphe introduit une transformation des graphes de configuration 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.
Un autre objectif des graphes de transition est d'introduire le modèle Korrontea vu au chapitre 2 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 68 résume ce principe en situant les règles de transformation introduites par rapport aux modèles déjà définis.

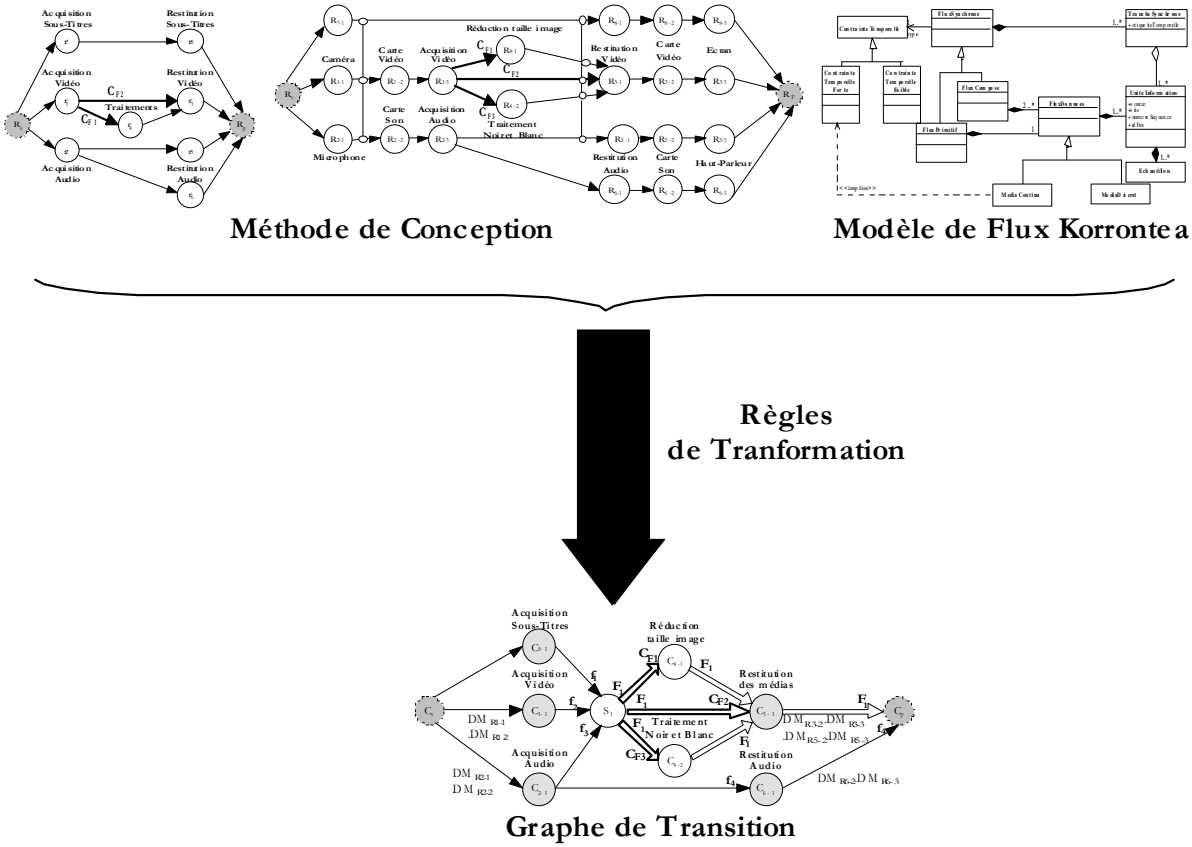


Figure 68 : Définition des Graphes de Transition

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

5.6.1 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. L'ensemble de ces moyens est recensé par la méthode car ils influent sur la QdS globale d'une AMD. Le contexte machine (propriétés et caractéristiques des ressources matérielles) est à considérer car il constitue un ensemble d'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, 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 pour le déploiement d'une AMD. L'exemple utilisé dans ce mémoire 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 composant 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, le déplacement de ces composants ne constitue pas une solution à favoriser a priori par la plate-forme.

5.6.2 Mise en évidence des Flux Synchrones

Un autre aspect important de la transformation proposée, plus liée au domaine d'application, 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 données ainsi représentées doivent être manipulables par les composants. La gestion des flux synchrones a des répercutions sur ces entités fonctionnelles car elles doivent être capables de manipuler indifféremment les flux primitifs et les flux composés.

5.6.3 Représentation des Graphes de Transition

Le principe de base de cette transformation est de dériver les graphes 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 abstraits de modélisation dès 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 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 composant logiciel qui devient désormais un concept central.

5.6.3.1 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 [Aldrich, 2002], [Heineman, 2001], [Samtinger, 1997].
Les composants logiciels 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. Le tableau suivant est un exemple d'une telle table qui correspond à l'exemple donné en début de chapitre. Cette table énumère les rôles atomiques obtenus précédemment. 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.

Correspondance Rôles/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
…	…	…	…

Tableau 6 : Table de Correspondance entre Rôles atomiques et Composants

Le Tableau 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 fonctionnalité. 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[3] [Lopes, 1995] (la programmation orientée aspects [Bouraquadi, 2001] [Kiczales, 1997] est l'une de ces approches) et la Figure 69 présente les différents types de composants logiciels que l'on identifie.

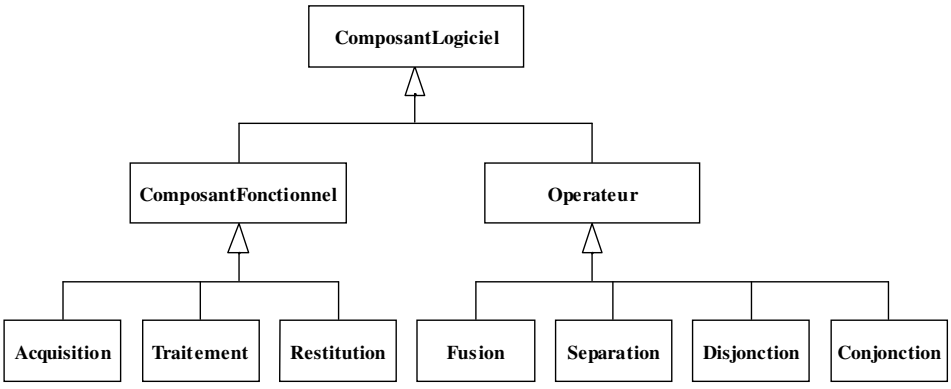


Figure 69 : Les Types de Composants Logiciels

5.6.3.2 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. Les opérateurs font également partie de l'ensemble CL, ils sont notés M_i, S_i, D_i et Co_i.

[3] 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.

5.6.3.3 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.

Nous précisons, sur la Figure 70, les représentations des arcs et des nœuds utilisés par les graphes de transition.

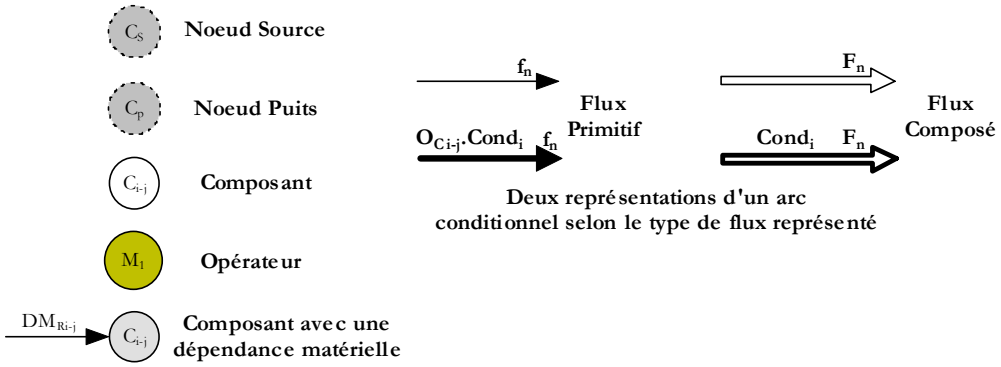


Figure 70 : Représentations des éléments du Graphe de Transition

5.6.3.4 Transformation des Graphes Fonctionnels en Graphes de Transition

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. Une description complète se trouve dans [Bouix, 2007]. Nous en faisons ici état d'un résumé :

Règle 1 *Les Composants logiciels*

Chaque rôle atomique est remplacé par un composant logiciel chargé de réaliser la fonctionnalité qu'il représente. La Figure 71 résume cette règle. Les flux entrants et sortants des composants logiciels peuvent être primitifs ou composés. Par simplification, la Figure 71 ne représente que des flux primitifs.

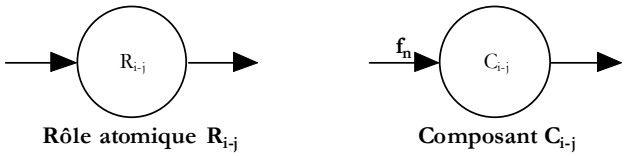


Figure 71 : Passage des rôles atomiques aux composants logiciels

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 que nous faisions 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 une dimension 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 partiellement 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. 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.

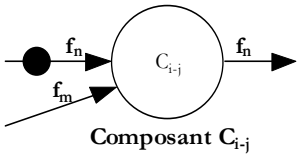
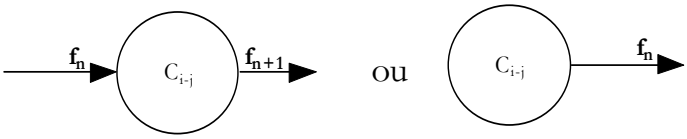


Figure 72 : Flux prépondérant en entrée d'un composant

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 son initiales. La notion de flux prépondérant permet de résoudre ce problème en introduisant de la sémantique dans ce type de traitement.

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 (Figure 73). Dans ces cas de figure, le ou les flux sortants d'une source localisée sont toujours différents du ou des flux entrants s'il y en a.



Source localisée C_{i-j}

Figure 73 : 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.

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 sortants, suivant que ces ressources sont nécessaires en amont ou en aval du composant concerné (Figure 74).

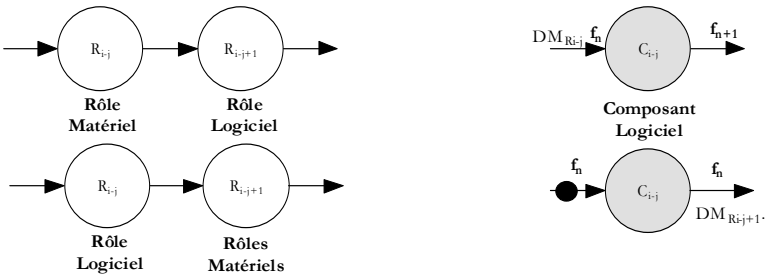


Figure 74 : Représentation des dépendances matérielles

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. Pour mener à bien son rôle, M_i utilise les politiques de synchronisation définies par le modèle Korrontea qu'il applique en fonction des contraintes des flux synchrones qu'il reçoit en entrée.

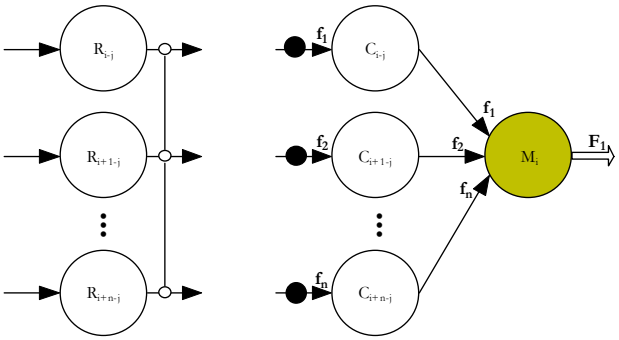


Figure 75 : Opérateur de fusion

Il est à noter que 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.

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 et est spécifié par un nouveau lien de synchronisation entre ces mêmes flux.

Règle 6 *Opérateur de séparation*
L'opérateur de séparation S_i 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.

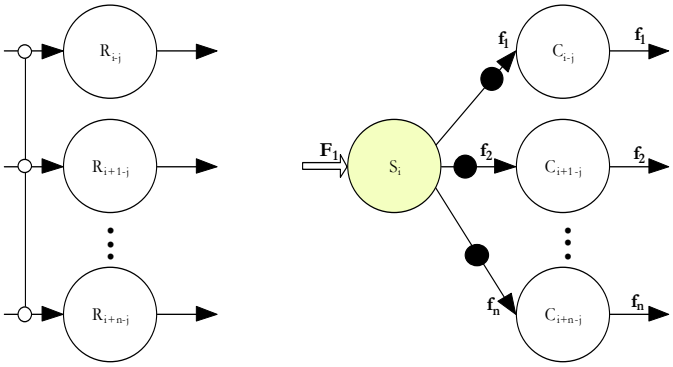


Figure 76 : Opérateur de séparation

La Figure 76 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 l'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 77. 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.

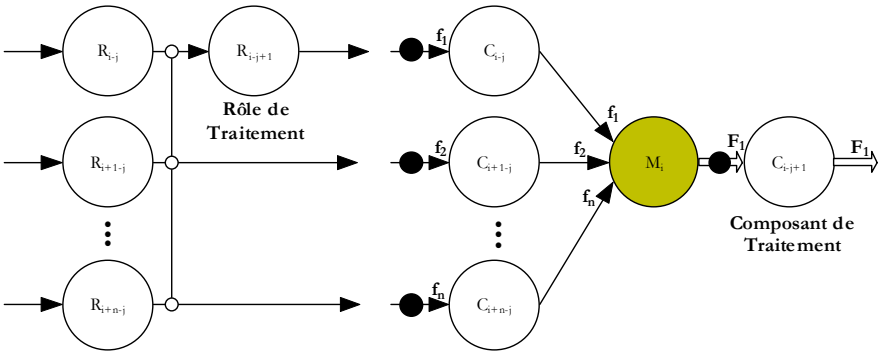


Figure 77 : Traitement des flux de données synchrones

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 offre 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 78 donne une représentation de cette transformation. Pour réaliser ce type de structure sur les graphes de transition, l'opérateur de disjonction permet de produire n flux synchrones à destination des composants de traitement comme le montre la Figure 78. 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.

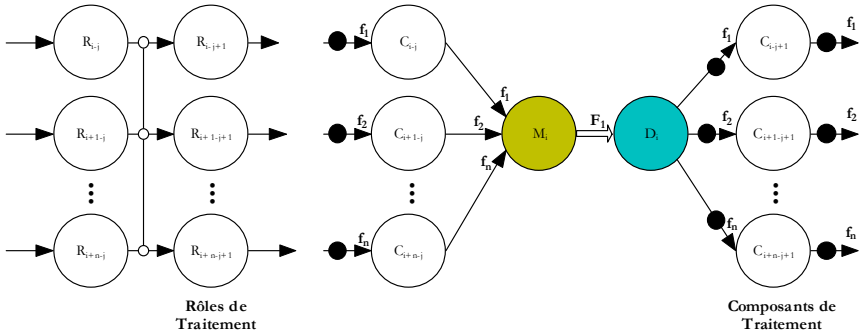


Figure 78 : Opérateur de disjonction

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 79. Cet opérateur ne peut être utilisé que si l'on a utilisé auparavant l'opérateur de disjonction, sinon c'est une fusion. Cette condition est spécifiée à l'aide de la contrainte d'utilisation O_{Di}.

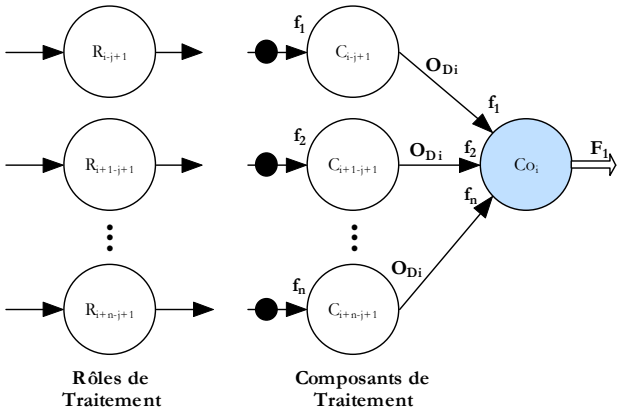


Figure 79 : Opérateur de conjonction

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 79 que tous les flux appartenant à ce type de chemin sur le graphe sont prépondérants (cf. règle 8). Dans le cas contraire, 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 indispensables 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 que lors des graphes fonctionnels. Les choix de configuration sont réalisés à l'aide des conditions apposées sur chacun de ces arcs ($Cond_i$). La Figure 80 décrit ces représentations. De plus, des contraintes d'utilisation de certains composants peuvent être indiquées dans ce type de spécification.

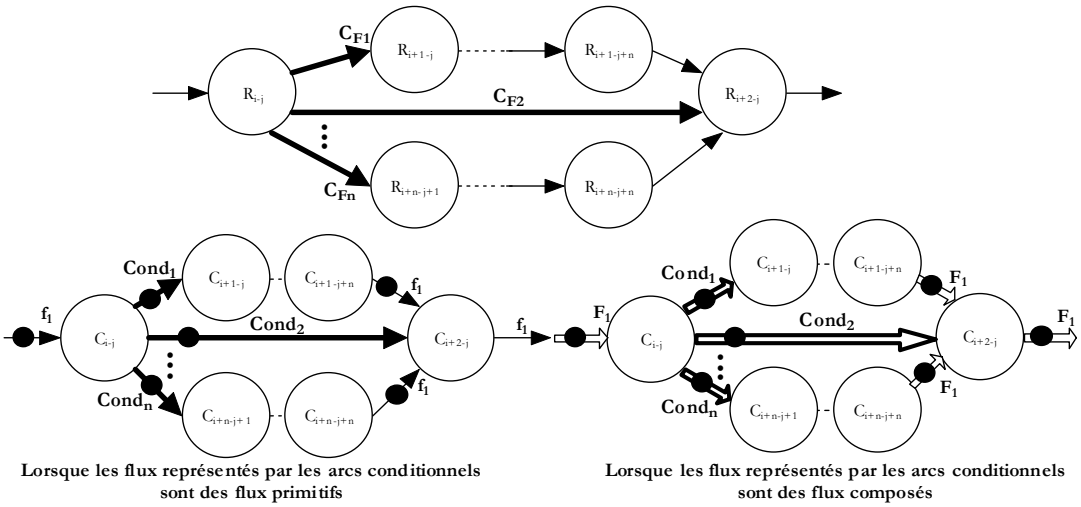


Figure 80 : Représentation des arcs conditionnels

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 composant 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.

5.6.3.5 Synthèse avant implantation

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 entité que sont les arcs, est destinée à la mise en œuvre des échanges de données entre les composants logiciels. Elle assure que les connexions puissent 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 l'utilisation d'opérateurs de flux synchrones qui vont permettre d'assurer certaines spécifications fonctionnelles. Ces opérateurs ne sont autre 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 des 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 capables 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 du point de vue de la synchronisation. 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 ni 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.

5.7 Graphe 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. 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 81 résume le principe de la transformation présentée dans ce chapitre.

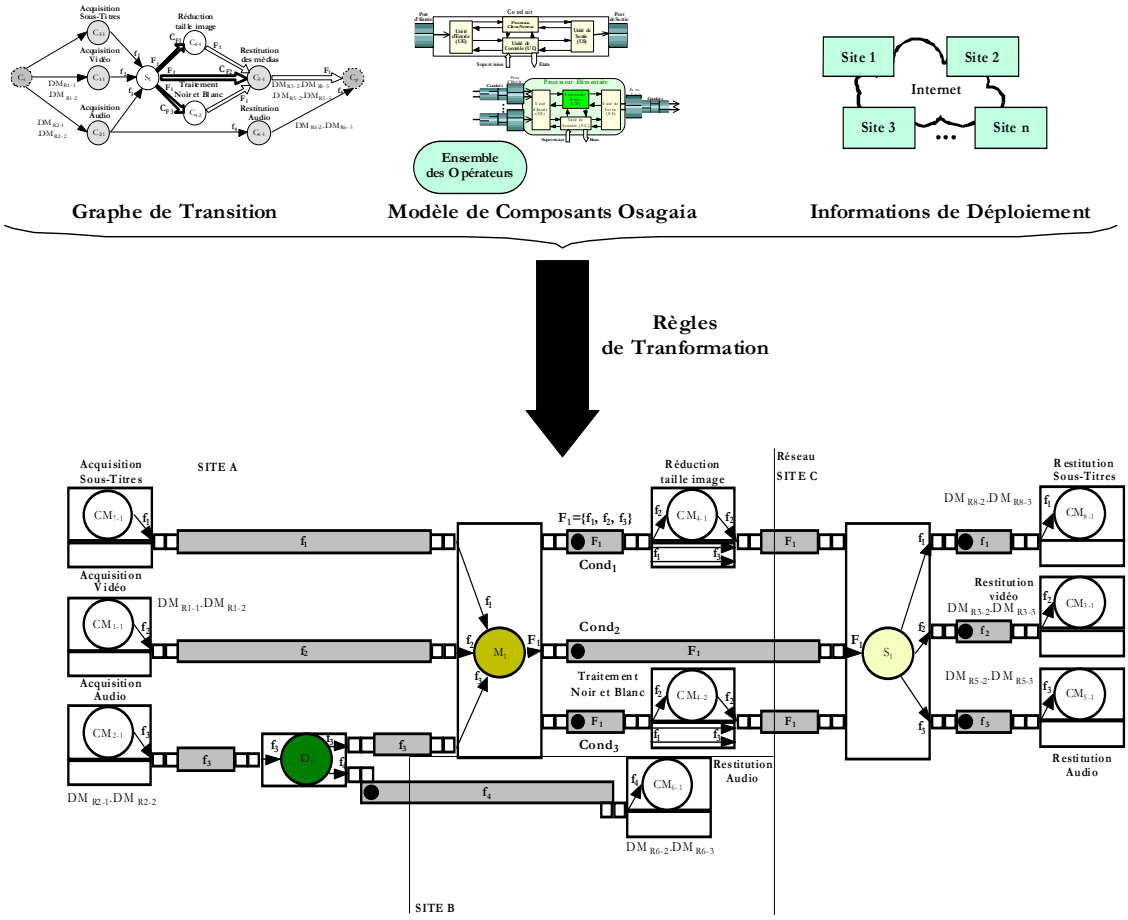


Figure 81 : Définition des graphes d'implantation

Les informations apportées par les graphes de transition, le modèle Osagaia et les sites de déploiement permettent de déduire ces graphes.

5.7.1 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.

5.7.1.1 Les Nœuds

Chaque composant (CM) C_{i-j} identifié sur les graphes de transition décrit une fonctionnalité atomique d'une AMD. 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 transition 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}.

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. Ainsi, nous distinguons les différents aspects d'une AMD.

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.

5.7.1.2 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. Ils sont modélisés 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 également conservées. Elles traduisent des obligations dans l'utilisation du composant spécifié en amont.

Nous donnons sur la Figure 82 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.

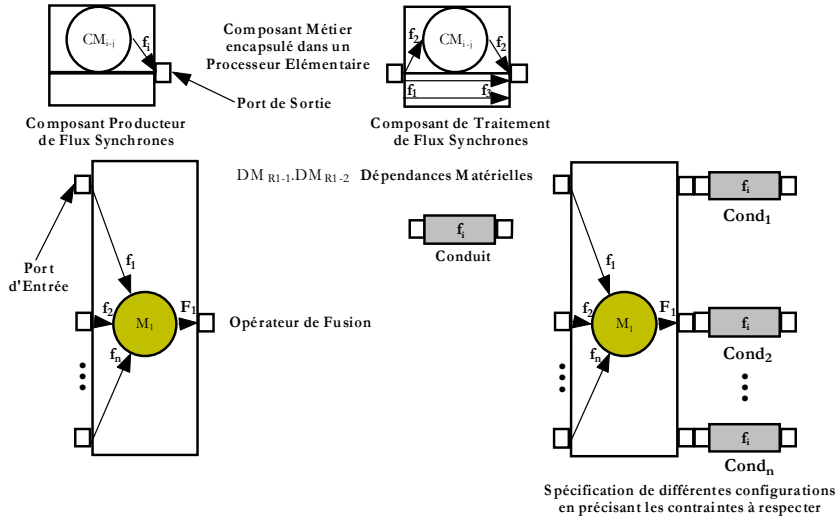


Figure 82 : Représentation des éléments des graphes d'implantation

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.

5.7.1.3 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. Comme l'avons fait précédemment pour le passage aux graphes de transition, il existe les mêmes

types de règles permettant le passage aux graphes d'implantation. Le détail des règles et leur représentation graphique est donné dans [Bouix, 2007], une synthèse est donnée à la Figure 84.

Avant de refermer ce chapitre, nous donnons dans la section suivante des exemples de transformation ainsi que les graphes d'implantation correspondants.

5.7.2 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 84 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 83 le graphe de transition correspondant qui a permis d'obtenir le graphe d'implantation.

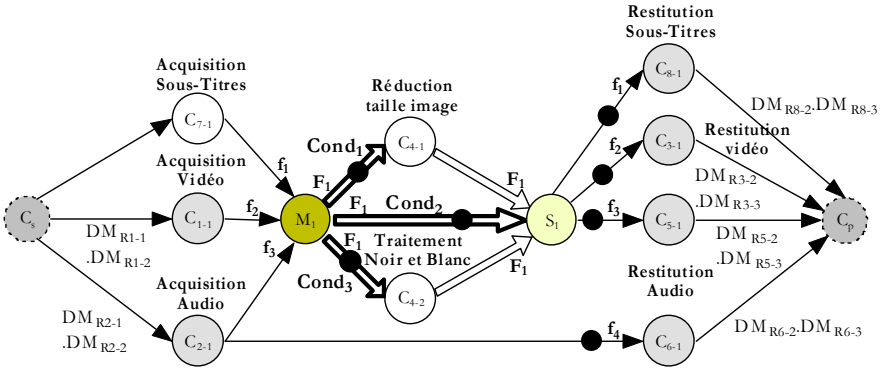


Figure 83 : Graphe de transition de l'application de formation à distance

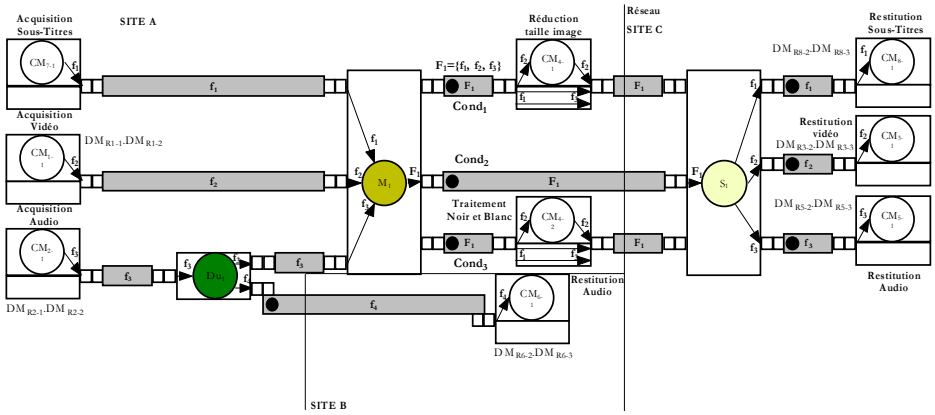


Figure 84 : Graphe d'implantation de l'application de formation à distance

Le site A correspond au site du locuteur. Il se dote des composants d'acquisition de flux nécessaires pour 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 85. 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 86.

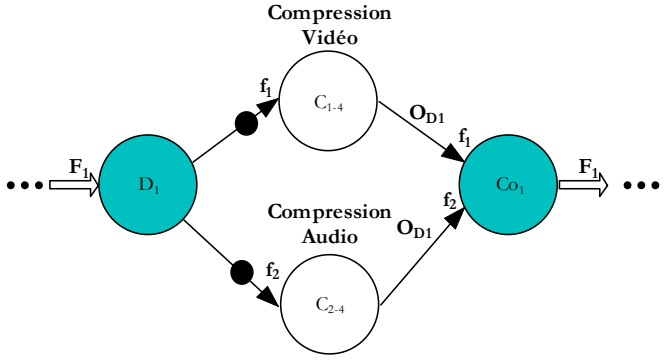


Figure 85 : Exemple de conjonction et de disjonction

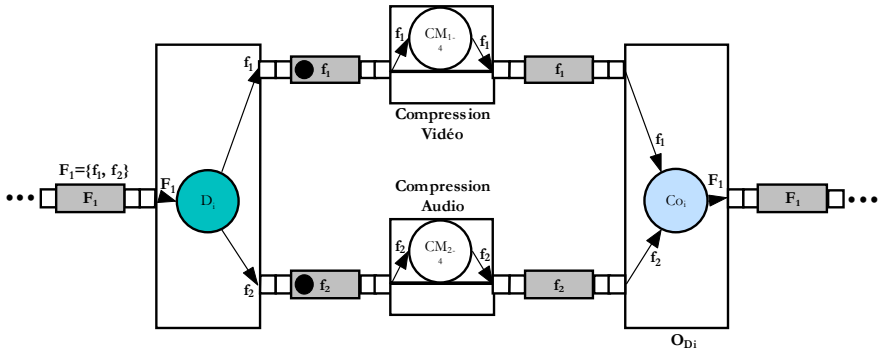


Figure 86 : Graphe d'implantation avec Conjonction et Disjonction

5.7.3 Reconfigurations

La modification du contexte de l'application peut engendrer une baisse de la QdS ou au contraire permettre son augmentation. Si la recherche de la meilleure solution est un problème identifié comme NP-Complet dans le cas général, il est possible à l'aide d'une heuristique de chercher une meilleure solution itérativement et ainsi, tendre vers la meilleure.

Pour ce faire nous avons mis en œuvre des algorithmes de reconfiguration de l'application qui utilisent les mécanismes de connexion/déconnexion/migration/remplacement des composants et Conduits précédemment décrits et qui se basent sur les graphes d'évaluation.

La recherche d'une configuration meilleure repose alors essentiellement sur deux critères. Le premier est imposé par les contraintes temporelles des applications multimédia : c'est l'efficacité de la plate-forme. Il est nécessaire que la plate-forme réagisse rapidement afin d'éviter les ruptures du service. Ceci est obtenu par la génération d'événements de reconfiguration. Le second critère vient, lui aussi, des particularités des applications multimédia où la perception que l'utilisateur a de l'application est centrale pour évaluer la QdS : c'est le maintien de la continuité ergonomique lors d'une reconfiguration appelée plasticité [Coutaz, 2001]. Il est respecté grâce à l'étude de la proximité de service entre configuration. Celle-ci est déterminée, d'une part, en utilisant l'architecture que nous avons conçue de manière à ce qu'elle reflète la vision qu'a l'utilisateur du service et, d'autre part, en utilisant les vœux que l'utilisateur exprimera.

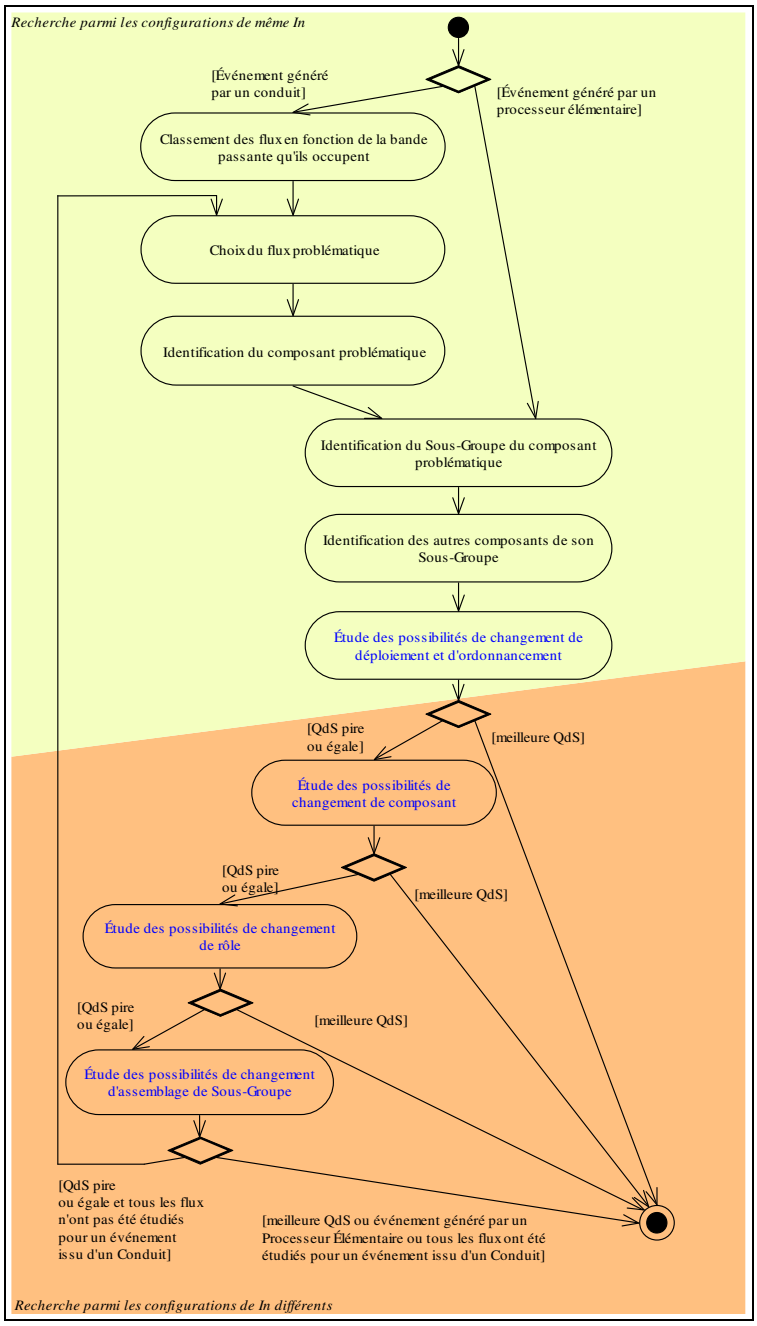


Figure 87 : Recherche provoquée par un composant de l'application

Nous avons ainsi pu construire une heuristique (Figure 87) dont la complexité est polynomiale et ne dépend plus que de la complexité intrinsèque de l'application et de l'étendue de l'offre de QdS que le concepteur souhaite proposer [Laplace, 2006].

Nous donnons ici les résultats obtenus pour une application de vidéosurveillance permettant de transmettre à un utilisateur distant les images et le son captés sur un site [Laplace, 2007]. Un seul Groupe appelé *Tel* correspondant à ce téléspectateur est disponible sous un seul assemblage constitué des deux Sous-Groupes synchronisés, *Im* transmettant l'image et *So* le son. Pour transmettre le son, le Sous-Groupe *So* propose deux décompositions fonctionnelles avec ou sans compression des données lors de la transmission. Pour transmettre les images, le Sous-Groupe *Im* propose six décompositions fonctionnelles qui diffèrent soit par la présence de compression ou de modification des images soit par l'ordre des traitements appliqués aux images. Enfin un composant utilisé dans l'application peut être remplacé par un composant jouant le même rôle mais offrant une qualité différente. Cette application propose alors 135 configurations possibles.
L'un des tests effectués (Figure 88) a permis de caractériser le comportement de la plate-forme et de l'application lorsque le contexte d'exécution oscille. L'application est tout d'abord déployée dans un

contexte favorable – contexte 1, C.1 – où ni les postes ni le réseau ne sont saturés puis ce contexte subit une dégradation – contexte 2, C.2 – due à la saturation de l'un des postes. Il revient ensuite à son état antérieur – contexte 1 – avant de se dégrader à nouveau –contexte 2.

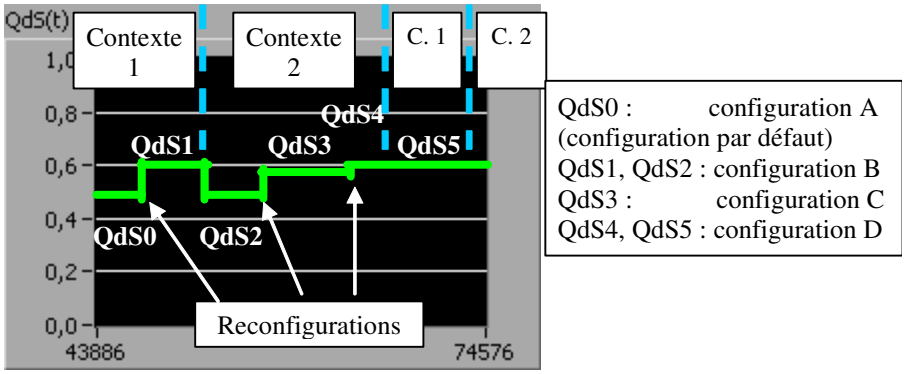


Figure 88 : Evolution de la QdS lors d'oscillations du contexte

La première reconfiguration correspond à une amélioration du déploiement –configuration par défaut – et consiste à remplacer un composant par un autre plus performant. La deuxième reconfiguration intervient après une dégradation du contexte et consiste à déplacer un composant du poste saturé vers un autre poste. La troisième reconfiguration améliore le résultat ainsi obtenu par le déplacement d'un autre composant vers un poste non saturé.

Nous remarquons que, suite à cette troisième reconfiguration, lorsque le contexte se dégrade à nouveau – passage du contexte 1 au contexte 2 –, la QdS ne varie pas et la plate-forme ne reconfigure plus l'application. Notons que ce ne serait pas le cas si dans la phase précédente la plate-forme avait optimisé les deux critères, intrinsèque et contextuel, et avait proposé d'atteindre QdS1 par une reconfiguration. Comme elle s'est contentée de QdS4 qui a moins de potentiel (figure 12) mais fournit le même service du point de vue de l'utilisateur, aucune reconfiguration n'est nécessaire lorsque le contexte continue à osciller. L'imprédictibilité des variations du contexte et le souhait de maintenir la plasticité [Coutaz, 2001] de l'application justifient donc *a posteriori* l'algorithme utilisé par la plate-forme.

Ce test est particulièrement intéressant car il permet de mettre en évidence que, lorsque le contexte retrouve sa situation antérieure après avoir évolué, la plate-forme propose une QdS de note identique mais en utilisant une configuration différente. La note de QdS reflétant l'adéquation entre le service fourni et celui souhaité par l'utilisateur, la configuration choisie, bien que différente, fournit donc une satisfaction équivalente à l'utilisateur. Ceci illustre bien le fait que seule la satisfaction de l'utilisateur a de l'importance. De plus, en ne cherchant pas l'optimum, la plate-forme Kalinahia stabilise le service et donne une plasticité meilleure et donc un comportement meilleur à l'application illustrant ainsi une notion proche du dilemme stabilité/précision connu en automatique. Une recherche exhaustive de la meilleure configuration – si elle était possible – ne présenterait pas cette plasticité ni ce pouvoir stabilisant.

5.8 Synthèse du chapitre

Nous avons décrit dans ce chapitre une nouvelle transformation dont le but est d'aboutir à une vue des AMD directement implantable. Dans la partie précédente, 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 multimédia 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 où 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.

Le passage d'une étape de la méthodologie à une autre n'est pas forcément automatique. Le tableau suivant résume l'effort à faire à chacune d'elle le cas échéant :

ETAPES			Automatisable	Justification
n°	**Définition**	**Résultat**		
1	Identification des classes de Groupes (service, utilisateurs)	Configuration canonique de l'application en terme de Groupe	**non**	définition des services par le concepteur
2	Identification des classes de Sous-Groupes (fonctionnalités)	Configuration canonique des Groupes en terme de Sous-Groupes	**partiellement**	définition des fonctionnalités par le concepteur
		Graphe des configurations des groupes en terme de Sous-Groupes	**non**	définition des QoS possibles par le concepteur
3	Identification des rôles des composants nécessaires	Graphe des flots de contrôle des Sous-Groupes	**partiellement**	décomposition fonctionnelle par le concepteur: utilisation de design pattern
		Graphe fonctionnel des Sous-Groupes (intermédiaire)	**partiellement**	décomposition fonctionnelle par le concepteur: utilisation de design pattern et taxonomie de rôles
4	Inventaire des composants disponibles	Liste des composants	**non**	inventaire des composants par le concepteur
		Graphe fonctionnel des Sous-Groupes	**partiellement**	si taxonomie des rôles: décomposition d'un rôle en rôle atomique
5	Décomposition fonctionnelle de l'application	Graphe fonctionnel de l'application	**partiellement**	définition manuelle des flux synchronisés/union automatique de graphes des groupes
6	Ajout des conduits	Graphe des configurations (intermédiaire)	**oui**	application de règles d'insertions des flux et des conduits
7	Introduction des flux dans les processeurs élémentaires	Graphe des configurations (intermédiaire)	**oui**	application de règles d'insertions des flux et des conduits
8	Identification des choix de composants	Graphe des configurations	**partiellement**	exclusion automatique des composants non compatibles

L'un des avantages de notre approche est l'unicité de représentation. En effet nous utilisons les graphes orientés pour représenter à la fois l'évaluation de la QdS et l'application avec ses différentes configurations et contraintes (comme la synchronisation).
De façon générale, les nœuds de ces graphes sont associés aux éléments opérationnels de l'application alors que les arêtes réifient les flux de données. Ces graphes permettent d'identifier les assemblages de composants et de flux communs à plusieurs configurations et d'optimiser ainsi l'évaluation en évitant la réévaluation de ces parties.
D'autre part, l'évaluation de l'application repose sur l'utilisation de moyennes qui permettent de déterminer la qualité des Sous-Groupes en fonction de celle des caractéristiques, la qualité des Groupes en

fonction de celle des Sous-Groupes et enfin la qualité de l'application en fonction de celle des Groupes. Ces moyennes sont pondérées à chaque fois que l'utilisateur peut exprimer des préférences.
Enfin nous proposons une méthode de conception qui utilise ces graphes pour élaborer une application pouvant être supervisée par notre plate-forme.

Chapitre 6 Conclusion

6.1 Bilan

Les travaux présentés de manière non chronologique dans ce mémoire reflètent une activité de recherche soutenue depuis 1997. Dans un premiers temps, mes travaux ont été guidés par une approche de type réingénierie afin de permettre l'amélioration d'applications existantes. L'idée était de réaliser une analyse de l'environnement, de son évolution et de proposer une approche méthodologique ainsi qu'une plate-forme technique permettant le remplacement et l'ajout de fonctionnalités sans modification de l'implémentation. Pour ce faire j'ai analysé le contexte de l'application en partant de son système d'information, en y ajoutant des informations de localisation, de mise en forme, etc. Le résultat de ces premiers travaux est ma thèse intitulée **ELKAR** (méthodologie & plate-forme).

Les données échangées offrent une vision intéressante mais ne sont pas suffisantes lorsque l'application est à développer. Il est tout aussi important de connaître le contexte d'intégration afin de vérifier les problèmes de compatibilité ou de non compatibilité, de mesurer le degré de compétence nécessaire pour réaliser cette intégration, de vérifier les environnements d'exécution, etc. Ce travail a été mené au cours de la thèse **SI-COTS** de Raphaël Michel dont le prototype est disponible à l'URL suivante : *http://www.iutbayonne.univ-pau.fr/~roose/pub/recherche/SICOTS/Setup_SICots.msi*

Partir d'applications totalement existantes (**ELKAR**) ou partiellement existantes (**SI-COTS**) permet certes une prise en compte de nombreuses informations contextuelles, mais n'offre pas de moyen d'intervention permettant ainsi d'agir ou de réagir aux évolutions du contexte.

Fort de ce constat, j'ai mis en évidence le besoin d'avoir des outils d'implémentation spécifiques permettant à la fois de rendre compte de leur contexte d'exécution local mais permettant également une supervision, seul moyen d'avoir une vision globale du contexte de l'application. J'ai également mis en évidence la spécificité de l'application cible, et proposé des outils de communication adaptés (dans ce cas au transfert de flux de données multimédia). La thèse d'Emmanuel Bouix a permis de proposer les modèles de composant **Osagaia** et de connecteur **Korrontea** implémentés via un prototype de vidéoconférence adaptable (téléchargeable sur *http://www.iutbayonne.univ-pau.fr/~roose/pub/recherche/osagaiakorrontea/*).

Les outils précédemment développés m'ont permis d'imaginer une vision plus ambitieuse de la prise en compte du contexte à savoir, intégrer l'expression des besoins des utilisateurs et les corréler aux possibilités offertes par le système à l'application, et ce de manière dynamique. La thèse de Sophie Laplace a permis de proposer une modélisation de la qualité de service avec prise en compte du contexte intrinsèque et contextuel et le développement de la plate-forme de supervision **Kalinahia**. Elle utilise pour ce faire les états fournis par les composants logiciels et les connecteurs afin de réaliser les reconfigurations (incluant migration, adaptation, composition de composants) nécessaires pour fournir une qualité de service acceptable. Le prototype est disponible à l'URL suivante : *http://www.iutbayonne.univ-pau.fr/~roose/pub/recherche/kalinahia/*

De manière générale, les travaux sur Korrontea, Osagaia et Kalinahia peuvent se synthétiser selon le schéma suivant :

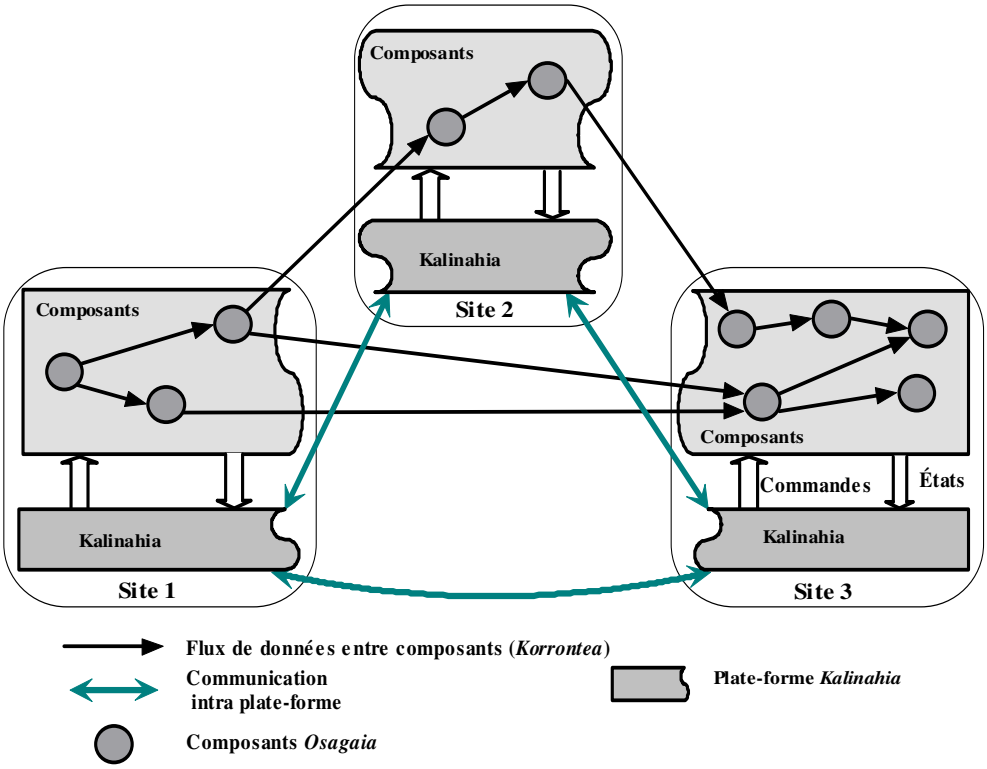


Figure 89: Osagaia, Korrontea, Kalinahia

L'ensemble des travaux présentés jusqu'ici couvre finalement un spectre large lié aux applications sensibles au contexte dont la particularité principale est d'avoir la capacité de détecter et de s'adapter aux modifications de leur environnement, permettant ainsi de répondre à la problématique de la qualité de service.
Si ce sujet de recherche est maintenant bien connu, il n'existe que peu de solutions méthodologiques permettant la conception d'applications dont l'objectif (au-delà des objectifs fonctionnels) est de fournir la meilleure qualité de service possible. C'est ce que nous avons proposé dans le Chapitre 5.

6.2 Perspectives

Afin de compléter notre approche verticale allant de la conception à l'implantation, nous avons également souhaité pouvoir prendre en compte le contexte physique des composants de l'application (localisation, vitesse de déplacement, luminosité) de même que les caractéristiques physiques de l'environnement d'exécution (*Capteur, Téléphone, PDA, PC*). Cela permettra à la fois d'ajouter des informations utiles pour la reconfiguration dynamique de l'application, mais aussi des adaptations plus pertinentes. Nous travaillons actuellement sur ces aspects dans le cadre de la thèse de Christine Louberry *(Transports de Flux Synchrones sur réseaux de capteurs sans-fil pour la surveillance à la demande)*. L'un des objectifs est de proposer une uniformisation du modèle de composant Osagaia et de flux Korrontea et une adaptation de la plate-forme Kalinahia aux périphériques contraints permettant de concevoir des applications sans se préoccuper de la cible de déploiement des composants, et offrant des reconfigurations dynamiques selon l'évolution du contexte environnemental, utilisateur, machine et temporel. La soutenance est prévue fin 2009.
La thèse de Christine Louberry démarrée en 2006 (contrat **ANR JCJC TCAP**) nous ouvre de nouvelles pistes très prometteuses. En effet, la prise en considération du matériel et de ses caractéristiques offre de nouvelles perspectives à tout point de vue. Comme mentionné précédemment, l'hétérogénéité des hôtes sur lesquels seront exécutés à la fois la plate-forme distribuée, les composants logiciels et les connecteurs implique non seulement l'adaptation des outils que nous avons jusqu'à présent utilisés mais demande également une méthode de conception spécifique. De plus, un travail orienté systèmes d'information est nécessaire afin de proposer un formalisme permettant de spécifier les connecteurs et le format des données échangées. Le défi qui s'offre à nous est de proposer un moyen de concevoir des applications sans se soucier de l'hôte sur lequel elles seront exécutées (hôte « traditionnel », CDC – moyennement contraint, CLDC – très contraint). Les premiers résultats publiés [Roose, 2006] [Louberry, 2007] nous ont

valu de devenir partenaire de Sun Microsystems, nous permettant ainsi de proposer des implémentations sur capteurs Sun Spot mais également d'obtenir un contrat avec le Conseil Général des Pyrénées Atlantiques et la Société Dev 1.0 pour du transfert de technologie.

Mais il va falloir aller plus loin dans notre approche. En effet, l'ensemble des travaux que nous avons abordés ne fonctionne qu'avec des entités identifiées au préalable. Ainsi la découverte et par conséquent l'assemblage dynamique ne sont pas possibles. Nous devons donc faire évoluer nos travaux vers les architectures orientées services [Hamida, 2008].
Une première piste de travail à courte échéance concerne les connecteurs, et plus précisément la génération automatique de connecteurs. En effet, l'hétérogénéité des composants implique également une interopérabilité complexe entre ces derniers [Passama, 2006]. Ainsi, deux cas peuvent se présenter : l'information produite est au même format que celle désirée, mais le mode de communication est différent ou le mode de communication est compatible, mais le format demande à être adapté (une image au format JPG transmise, une image PNG attendue). Il est nécessaire dans le premier cas d'utiliser un connecteur « intermédiaire » tandis que dans le deuxième, il est nécessaire d'avoir un connecteur permettant la transformation. Si d'un point de vue technique ceci ne pose pas de difficulté, il est intéressant, d'un point de vue méthodologique, d'intégrer ces préoccupations afin de permettre un déploiement des composants et des connecteurs intermédiaires nécessaires (et ce transitivement le cas échéant). Ce travail est en cours dans le cadre de la thèse de Makhlouf Derdour (Modélisation et implémentation d'un système d'information de gestion de flux multimédia pour des architectures logicielles intégrant des capteurs sans-fil mobiles).

La seconde piste que nous n'avons pas abordée et qui semble plus que prometteuse est l'intégration de la composante géographique dans les applications. Cette thématique est nommée « *location-aware* » par analogie avec « *context-aware* ». Si l'on se positionne sur la première couche de la Figure 2, il est nécessaire d'y ajouter un autre type de contexte appelé *géographique*. Dans ce mot se retrouve beaucoup de choses (lieu physique, logique, déplacements, vitesse, coordonnées GPS voire GSM, etc.). De telles informations sont le plus souvent obtenues via des capteurs de type physique, virtuel ou logique [Baldauf, 2007]. La couche 2 représente de telles informations en vue de leur mise à disposition au niveau de l'application. Or, si la capture d'informations contextuelles géographiques est intéressante, il est tout aussi intéressant d'en profiter d'un point de vue fonctionnel pour optimiser les traitements, permettre l'obtention d'informations plus pertinentes, voire permettre des traitements impossibles sans elle. Ainsi, il est possible d'imaginer la migration de composants sur les périphériques de capture de contexte. Malheureusement, il n'est pas possible de réaliser ceci via un outil « classique » de migration. En effet, un composant logiciel s'exécute la plupart du temps sur une plate-forme (au sens système d'exploitation, intergiciel) qui ne peut être étendue telle quelle sur des périphériques de capture tels que les capteurs ou autres téléphones/smartphones pour des raisons matérielles. Ceci demande quelques explications : certaines plates-formes (*Squawk – Machine virtuelle Java pour périphériques de type CLDC*) dédiées aux systèmes embarqués offrent des possibilités de migration des composants. Toutefois ceci impose que la décision de migration soit faite par l'application elle-même et soit donc prévue dans le code de celle-ci.
Il est alors nécessaire de faire des adaptations. C'est pourquoi nous proposons d'ajouter une dimension appelée « *polymorphisme* » dans la panoplie des outils de la dernière couche. L'idée générale est de proposer un modèle de composant générique qui peut être déployé et/ou migré sur différents supports avec une adaptation par polymorphisme de ses caractéristiques et de son état afin d'assurer d'une part son fonctionnement mais également de permettre une continuité d'exécution dans le cas de migration. Si l'on reprend la Figure 2 de l'introduction, nous la modifions comme suit :

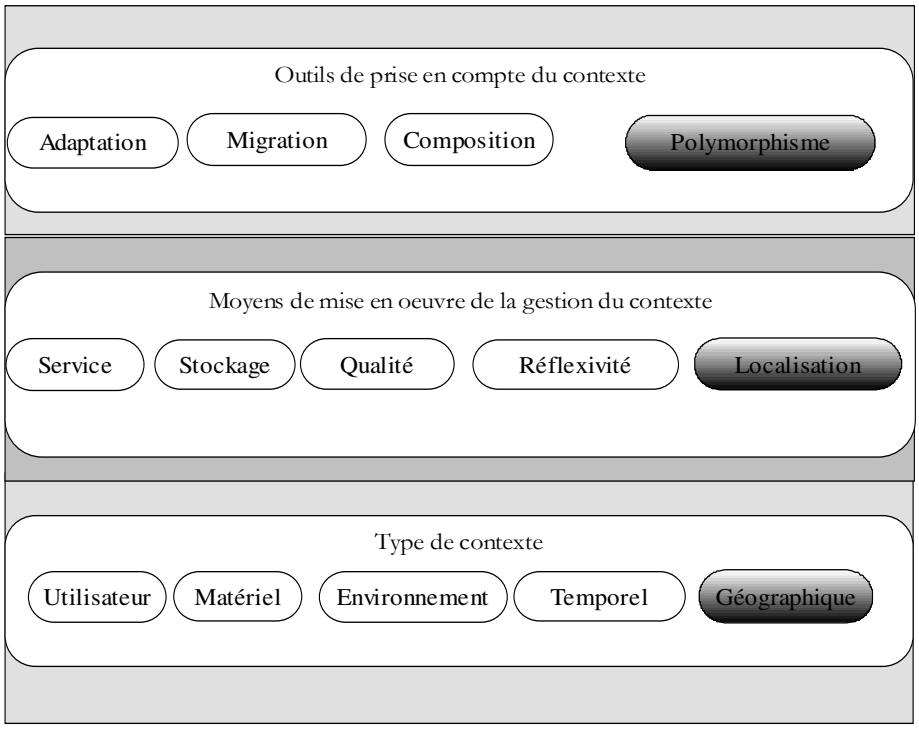


Figure 90 : Ajout de la dimension 'localication'

Bien que non abordé dans ce mémoire, j'ai également travaillé tout au long de ces années sur les aspects liés aux plate-formes d'exécution. Depuis le début de mes travaux de recherche, je me suis toujours préoccupé d'avoir une approche verticale, à savoir des aspects méthodologiques jusqu'à l'implémentation en passant par la couche « plate-forme ».
Les orientations actuelles vers l'informatique ubiquitaire demandent d'une part aux applications d'être capable de prendre en compte leur contexte de manière générale (*utilisateur, matériel, environnement, temporel et géographique*) mais également de pouvoir gérer un matériel de plus en plus hétérogène tout comme le deviennent les environnements logiciels. Aussi, en corrélation avec l'approche contextuelle, j'envisage de travailler sur une machine virtuelle et une plate-forme de gestion de la mobilité permettant aisément de migrer d'un site à un autre et appliquant des politiques de polymorphisme si nécessaire. La mobilité des périphériques légers est, de fait, subie par l'application et non dirigée par elle. Il n'est, par conséquent, pas possible de concevoir une application capable de gérer cette mobilité sans devoir y introduire une surveillance permanente des dispositifs mobiles.
En revanche une plate-forme de supervision peut libérer l'application de ce rôle de surveillance du contexte et provoquer les ajouts, suppressions et migrations de composants nécessaires.
Afin de rendre ceci possible il est envisageable de concevoir une machine virtuelle au-dessus de laquelle s'exécutent les composants de l'application. Cette machine virtuelle offre un middleware au travers duquel s'effectuent les communications entre composants et entre les composants et leur environnement. De par son rôle de machine hôte, elle peut créer de nouveaux composants, les lancer, les arrêter et en récupérer l'état. Grâce à son middleware elle peut rediriger les entrées/sorties des composants.

La plate-forme de supervision peut s'adresser à ces machines virtuelles lorsqu'elle veut ajouter, enlever ou faire migrer des composants. Les machines virtuelles vont alors récupérer l'état des composants, les transmettre et rediriger leurs entrées/sorties. Ainsi la migration devient transparente à l'application et les entrées et sorties des composants sont automatiquement redirigées sans qu'il soit nécessaire d'intervenir sur le composant lui-même.
Le développement d'application pour périphériques mobiles n'est plus une nouveauté en soi. Néanmoins, ce type d'application soulève des problèmes de conception et de mise en œuvre qui n'ont, pour l'instant, pas été résolus dans une optique de déploiement à grande échelle. Ceci demande une refonte complète de l'approche afin de permettre aux développeurs de composants de s'affranchir totalement de l'aspect de mobilité et aux architectes d'applications de ne se concentrer que sur la gestion du contexte d'exécution. Seule une organisation en couches à la manière des couches réseaux (norme ISO) permet une telle indépendance.

La séparation des aspects fonctionnels, contextuels et d'exécution permet une modélisation fine de chacun des aspects, offre une séparation claire des préoccupations pour les concepteurs, et les développeurs et surtout ouvre la voie vers une mobilité totale des composants sans devoir "tisser" dans leur code ces aspects de mobilité. Ainsi l'implémentation des composants n'est plus dictée que par l'application cible et non par son contexte d'exécution. Le modèle à proposer permettra une totale indépendance, dans la conception et l'implémentation des composants logiciels pour applications mobiles, du contexte applicatif. Il semble assez logique dans cette optique-là de partir sur une approche de type MDA, avec un PIM exprimé non pas en UML, mais avec SysML (*System Modeling Language*) qui est formalisé sous forme de profil UML 2.x.

Chapitre 7 Animation scientifique

Depuis mon arrivée dans le monde de la recherche en 1997 comme doctorant, puis en 2001 comme maître de conférences, je n'ai cessé d'avoir une activité de recherche, et ce même en ayant des responsabilités au sein de l'IUT de Bayonne tant pour le montage et le portage de formations que comme directeur des études du DUT Informatique (2002-2005) et de la Licence professionnelle (2001-2003), puis comme Responsable du Pôle Multimédia regroupant 3 formations (depuis 2006) :

- Une Licence professionnelle intitulée : Systèmes Informatiques et Logiciels, option Communication Multimédia (*LP SIL-CM*), licence ouverte en formation initiale et continue.
- Un DU Technologies de l'Information et de la Communication (*DU TIC*). Formation continue.
- Un DU Administrateur de Systèmes et de Réseaux (*DU ASR*). Formation initiale et à distance – *en cours pour ce dernier point.*

Relations Internationales
Parallèlement à mes activités d'enseignement et de chef de service, je participe également aux relations internationales de l'IUT de Bayonne. Ceci s'est traduit par la participation au montage du DU ASR en double diplôme avec l'Université Catholique du Nord de la Serena au Chili. J'ai eu l'occasion d'assurer en collaboration avec mon collègue Marc Dalmau le module de programmation Java sur 1 semaine. Il y a également eu 2 missions de formation d'enseignants à la Havane (Cuba) pour des cours sur la programmation web (PHP) pour le compte du MINED (Ministère de l'Education) et de l'ISPETP (Institut Supérieur Pédagogique pour l'Education Technique et Professionnel). Les enseignements et supports de cours ont été réalisés en espagnol.
Enfin, dans mon rôle de responsable du Pôle Multimédia, j'ai mis en place des relations avec l'ADIUT et participe au programme d'accueil d'étudiants Vénézuéliens et Mexicains.

Au-delà de mes activités d'enseignant/chercheur, j'ai toujours tenté d'être également un animateur. Ce dernier point s'est exprimé au travers d'activités de formation par la recherche, de montage et de prise en charge de contrats, de participation à des comités d'organisation, lecture et programme, par des fonctions électives dans les instances de l'université, par la participation à différents groupes de travail et bien évidemment par l'écriture et la présentation de papiers dans des conférences et revues nationales et internationales.

Le présent chapitre tâche de lister chacune de ces activités.

7.1 Formation par la recherche

Encadrement de DEA/Master Recherche/Ingénieurs

- **2008** : co-encadrement du Master Recherche de Cyril Cassagne (*Labri/Bordeaux I*)
- **2008** : co-encadrement stage PFE Ingénieur de Wissem Ben Ammar – (*ENSI/Tunis, Tunisie*)
- **2008** : Encadrement stage Ingénieur d'Antoine Vigneau – (*Polytech'Lille*)
- **2006** : co-encadrement du Master Recherche de Christine Louberry (*LIUPPA/Pau*)
- **2003** : co-encadrement du DEA d'Emmanuel Bouix (*LIUM/Le Mans*)
- **2002** : co-encadrement du DEA de Sophie Laplace (*LIUM/Le Mans*)

Trois des quatre stages « recherche » se sont poursuivis par une inscription en thèse et un co-encadrement. Le 4ème (Cyril Cassagne) se poursuit par la signature d'un CDD d'un an sur contrat.

Co-encadrement de thèses

- **2003-2007** : Emmanuel Bouix (*le 29/11/2007*) – co-encadrement à 40%
  - *Modèle de composants et de flux pour applications multimédia distribuées dynamiquement reconfigurables*
  - Co-direction : Marc Dalmau, Franck Luthon
  - Financement **:** *Contractuel PRAG à mi-temps, puis ATER.*
  - Jury composé de :
    - Didier Donsez, Rapporteur, Professeur, IMAG, Grenoble
    - Daniel Hagimont, Rapporteur, Professeur, IRIT, Toulouse
    - Jean Christophe Lapayre, Président, Professeur, LIFC, Besançon
    - Stéphane Frénot, Maître de Conférences, Examinateur, CITI-INRIA Ares, Lyon
    - Franck Luthon, Professeur, Directeur de thèse, LIUPPA, Bayonne
    - Marc Dalmau, Maître de Conférences, co-encadrant, LIUPPA, Bayonne
    - Philippe Roose, Maître de Conférences, co-encadrant, LIUPPA, Bayonne

- **2003-2006** : Raphaël Michel *(le 04/12/2006)* – co-encadrement à 50%
  - *Système d'Informations pour l'évaluation des composants sur étagères (COTS Component)*
  - Co-direction : Franck Barbier
  - Financement **:** *Fongécif.*
  - Jury composé de :
    - Bernard Coulette, Rapporteur, Professeur à l'université de Toulouse le Mirail,
    - Colette Rolland, Rapporteur, Professeur à l'université de Paris1-Panthéon-Sorbonne.
    - Mr Pascal Weil, Président, Examinateur, Directeur de recherche au LaBRI.
    - Mr Franck Barbier, Directeur de thèse, Professeur à l'UPPA,
    - Mr Philippe Roose, co-encadrant, Maître de conférences à l'UPPA.

- **2002-2006** : Sophie Laplace (*11/05/2006*) – co-encadrement à 40%
  - *Conception d'Architectures Logicielles pour intégrer la qualité de service dans les applications multimédia réparties*
  - Co-direction : Marc Dalmau, Franck Luthon
  - Financement : PRAG Titulaire, congé de formation
  - Jury composé de :
    - Isabelle Gérard-Demeure, Rapporteur, Professeur à l'E.N.S.T.
    - Jean-Pierre Giraudin, Rapporteur, Professeur à l'ENSIMAG
    - Daniel Hagimont, Président, Professeur, IRIT, Toulouse
    - Franck Luthon, Directeur de thèse, Professeur à l'U.P.P.A
    - Marc Dalmau, Co-Encadrant, Maître de Conférences à l'U.P.P.A.
    - Philippe Roose, Co-encadrant, Maître de Conférences à l'U.P.P.A.

- **2006-2009** : Christine Louberry (2006-) - co-encadrement à 40%
  - *Gestion de la qualité de service par déploiement et adaptation dynamique de composants hétérogènes en environnement mouvant*
  - Co-direction : Marc Dalmau, Congduc Pham
  - Financement sur contrat.

- **2007-2010** : Makhlouf Derdour (2007-) - co-encadrement à 30%
  - *Modélisation et implémentation d'un système d'information de gestion de flux multimédia pour des architectures logicielles intégrant des capteurs sans-fil mobiles*
  - Co-tutelle avec l'Algérie, co-direction Marc Dalmau, Congduc Pham (France), Nacira Ghoualmi (Algérie)
  - Autofinancement (*Makhlouf est enseignant en informatique – une demande d'aide est en cours auprès d'EGIDE*).

Thèse rapportée (*sous couvert de Congduc Pham***)**

- **Septembre 2008** : Eve Atallah
  - *Une solution pour l'établissement non planifié de groupes sécurisés permettant des communications sûres dans les réseaux MANets purs.*
  - Rapporteurs : Christian Laforest, Philippe Roose, Congduc Pham

- Examinateurs : Jean-Pierre Borel, Serge Chaumette, Jean-Louis Lanet, Pierre-François Bonnefoi, Oliver Heen
- Invité : Olivier Ly

7.2 Contrats (en tant que porteur)

- **Projet KALINAHIA :** *Développement d'applications multimédias pour périphériques mobiles/nomades* (**93,5 K€ dont 50 K€ financés par le Conseil Général des Pyrénées Atlantiques**)
  - Responsable du projet
  - Suivi scientifique et comptable
  - Animation du groupe de projet
  - Emploi d'un ingénieur (CDD d'1 an)
  - **Projet de Transfert de Technologies** en collaboration avec l'entreprise Dev 1.0

- **Projet ANR JCJC** : *TCAP : Transport de flux vidéo sur réseaux de capteurs pour la surveillance à la demande (**900 K€, dont 126 K€ financés par l'ANR**)*
  - Responsable du projet
  - Suivi scientifique et comptable
  - Animation du groupe de projet
  - Emploi d'un ingénieur (CDD d'1 an)
  - Partenaires officiels de Sun Micro Systems.

- **Projet LIUPPA :** *Alcool : Architecture logicielle, Composants et Protocoles (**2x15 K€**)*
  - Responsable du projet
  - Suivi scientifique et comptable
  - Animation du groupe de projet
  - Projet partiellement soutenu par le Conseil Régional d'Aquitaine.

7.3 Groupes de Travail

- Membre du groupe de travail « ERTSI - Evolution, Réutilisation et Traçabilité des SI » (ex-groupe OCM-SI du GdR I3)
  - Membre depuis la formation du groupe
  - Participation aux réunions
  - Co-présidence et membre des différents appels pour le Workshop OCM-SI (*7ème édition depuis 2000*)
- Membre de l'action "Adaptation Dynamique aux Environnements d'Exécution", GdR ASR du CNRS, Pôle GSP (Grille, Système et Parallélisme)
- Membre du groupe de travail MADSI « Méthodes Avancées de Développement des Systèmes d'Information »
- Membre de la plate-forme CNRS RECAP

7.4 Expertise

- Expert extérieur auprès du GIP ANR (2006)
- Membre du "IEEE Communication Society - Multimedia Communications Technical Committee"

7.5 Comités

Programme

- 2nd International Conference on Internet Multimedia Services Architecture and Applications (IMSAA-08)
- Workshops OCM-SI (*Objets, Composants, Modèles pour les Systèmes d'information*) - 2005, 06, 07, 08
- Workshop ERTSI (*sur l'Évolution, la réutilisabilité et la traçabilité des systèmes d'information*) - 2008
- Workshops MADSI (*Méthodes avancées de développement des systèmes d'information*) - 2007, 08

- Workshop OOPSLA 2007/PLAC (*The First International Workshop on Patterns Languages: Addressing Challenges*)
- SASA 2007 - The Second IEEE International Workshop Towards Stable and Adaptable Software Architectures
- SIPE'08, Third ACM International Workshop on Services Integration in Pervasive Environments
- SIPE'07, Second IEEE International Workshop on Services Integration in Pervasive Environments
- Inforsid 2006, 2009
- CollaborateCom 2005, 2006, 2007, 2008
- 2004 IRMA International Conference
- COMPSAC 2002 (IEEE Computer Society)
- SNPD'02 & SNPD'03 publié chez ACIS
- ACM SAC 2004, 2005, 2006 - ACM/SIGAPP

Lecture

- Relecteur pour la revue "IEEE Communications Magazine" - Numéro sur "Consumer Communications and Networking"
- Membre du bureau de la revue International Journal of Software Architecture
- Membre du comité de lecture du numéro spécial RSTI - L'objet : « Architectures Logicielles ».
- Membre du comité de lecture du numéro spécial de la revue « Document Numérique - Recherche d'Information dans les documents structurés (Web, XML) » - Juin-Juillet 2007.
- Membre du comité de lecture du Numéro spécial : Adaptation et gestion du contexte Revue Ingénierie des systèmes d'information (ISI)
- Relecteur pour IEEE Communications Magazine – Numéro « Consumer Communications and Networking »
- Relecteur pour International Journal of Computing and Information Sciences - (IJCIS) (ISSN 1708-0460, ISSN 1708-0479 (On-Line)
- Relecteur pour la revue "Journal of Systems and Software" - Elsevier ed.
- Relecteur pour la "Journal of Cases on Information Technology" (ACIT)"
- Relecteur pour "Guide to the Swebok" - Trial version 1.0 - IEEE Computer Society Editions.

Organisation & Divers

- Co-rédacteur du Numéro Spécial « Composants et modèles dans l'ingénierie des SI » - Revue ISI, Hermès-Lavoisier, 3/2008
- Membre du comité d'organisation d'INFORSID 2004 à Biarritz
- Co-président de l'édition 2007 du Workshop OCM-SI (Inforsid, 2007)

7.6 Fonctions Électives & Divers

- Élu au Conseil Scientifique de l'UPPA en 2005-2008.
- Membre suppléant de la Commission de Spécialiste – 27ème section
- Titulaire de la **PEDR** depuis 2006.

7.7 Collaborations universitaires

Conscient de l'ouverture nécessaire, j'entretien avec un certain nombre de collègues de relations suivies.

- **Nacira Ghoualmi**, MGEP, Algérie
  - Dépôt d'un projet Egide-Tassili
  - Accueil de Mme Ghoualmi en septembre 2006, octobre 2008
  - Publications communes
  - Co-tutelle de thèse (démarrée en 2007) de M. Makhlouf Derdour.

- **Guillermo Barrutieta** (Mondragon Goi Eskola Politeknikoa, Espagne)
  - Dépôt de projet Aquitaine/Euskadi à 2 reprises.
  - Réunions communes sur Bilbao, San Sebastian, Bayonne

- **Frédéric Le Mouël**, CITI/INRIA Ares, France
  - Invitation à un séminaire en 2007
  - Nombreuses entrevues
  - Projet GdR-ASP déposé (refus, problème de budget du GdR au dernier moment)

- **Djamel Seriai**, EMN Douai, France
  - Co-organisation et co-présidence du workshop OCM-SI 2007
  - Co-rédacteur numéro spécial « Composants et modèles dans l'ingénierie des SI » - Revue ISI, Hermès-Lavoisier, 2007.

- **Agnès Front**, IMAG, Grenoble, France
  - Participation commune à l'élaboration d'un état de l'art dans le cadre du GdR MADSI

- **Vincent Lecuire**, CRAN, Nancy-Université, France
  - Participation au projet ANR JCJC TCAP
  - Projet de dépôt d'une ANR pour 2009 (avec France Telecom)

7.8 Collaborations industrielles

- **Société Dev 1.0** : Dépôt de projet LIUPPA/DEV 1.0 auprès du Conseil Général des Pyrénnées Atlantiques. ***Accepté (50 K€)***
- **Sun Microsystems** : Partenariat Sun MicroSystems/LIUPPA pour le test et le développement d'applications pour capteurs Sun Spots. Contact : Eric Arsenau, Chef du projet Squawk (Machine virtuelle pour capteurs Sun Spots)/Sun Microsystems.
- **Régie des Transports des Landes (RDTL)** : Projet de dépôt d'une ANR avec France Télécom (pour ***2009***).
- **Tecnalia-Robotiker** : 2 dossiers Aquitaine/Euskadi déposés (refusés).

7.9 Publications personnelles

Conférences Pédagogiques

- **Conférence Invitée**
  - *Internet pour les associations* – Cycle de Formation, Mairie de Biarritz, Avril 2008.
  - *Histoire de la micro*-informatique – Festival des Sciences – Parc Montaury, Anglet, 10/10/2008
- **Roose Philippe** - *Retour sur un transfert de connaissances techniques - Synthèse et critiques* - IV Encuentro Europa-America Latina sobre la formación y la cooperación technológica y profesional - Isla Margarita, Venezuela – 2004.
- **Roose Philippe**, Filbet Michel - *Expérience d'une formation individualisée dans l'enseignement supérieur - Cas d'un étudiant handicapé par une myopathie* - III Encuentro Europa-America Latina sobre la formación y la cooperación technológica y profesional - La Habana, Cuba – 2002.

Livres, chapitres de livres

- **Philippe Roose** - L'âge d'Or – Histoire des Micro-ordinateurs, Ed. Cépadues, ISBN : 2.85428.696.0 - 120 pages (12/2005).
- Corédacteur - Numéro Spécial Revue ISI - Objets, Composants et Modèles dans l'Ingénierie des Systèmes d'Information - Hermès 3/2008 (juin 2008).

Revues internationales

- Christine Louberry, **Philippe Roose**, Marc Dalmau - *Heterogeneous component interactions: Sensors integration into multimedia applications* - Journal of Networks, Issue N°6, Academy Publisher - ISSN : 1796-2056 – 2007.
- Aniorté Philippe, **Roose Philippe**, Dalmau Marc - *Using active Database Mechanisms to Build Cooperative Work* - Journal of Integrated Design and Process Science (JIDPS) - Vol. 3, N°1, pp1-14 - ISSN : 1092-0617 - March 1999 -

Revues nationales

- Christine Louberry, Marc Dalmau, **Philippe Roose -** Intégration de périphériques contraints par reconfigurations dynamique d'applications - Numéro Spécial Revue ISI - Objets Composants Modèles dans l'ingénierie des Systèmes d'Information - Hermès 3/2008 (juin 2008).
- Sophie Laplace, Marc Dalmau, **Philippe** Roose - *Kalinahia : modèle de qualité de service pour les applications multimédia reconfigurables* - Numéro Spécial Revue ISI "Conception : patrons et spécifications formelles" - Volume 4, 2007.

Conférences internationales

- Sma Saighi, Nacira Ghoualmi, **Philippe Roose** - Towards an Open Exchange System for Multimedia Mobile Phone - 10th International Conference on Enterprise Information Systems (ICEIS), 12 - 16, June 2008, Barcelona, Spain.
- Emmanuel Bouix, **Philippe Roose**, Marc Dalmau - *The Korrontea Data Modeling* - Ambi Sys 2008 - International Conference on Ambient Media and Systems - Quebec City, Canada - 11-14/02/2008
- Sophie Laplace, Marc Dalmau, **Philippe Roose** - *Kalinahia: Considering quality of service to design and execute distributed multimedia applications* - NOMS 2008 - IEEE/IFIP Int'l Conference on Network Management and Management Symposium - 7-11/04/2008 Salvador de Bahia, Brazil*, 2008*.
- Louberry Christine, **Roose Philippe**, Dalmau Marc - *Towards sensor integration into multimedia applications* - 4th European Conference on Universal Multiservice Networks ECUMN'2007 - Toulouse, France - 12-14 February, 2007 - IEEE Computer Society Press ISBN : 0-7695-2768-X pp. 355-363
- Michel Raphaël, **Roose Philippe**, Barbier Franck - *Information System for Evaluation of COTS* - 3rd International Conference on Software Engineering Research, Management and Applications (SERA2005) - Central Michigan University, Mount. Pleasant, Michigan, USA - IEEE Computer Society 2005, ISBN 0-7695-2297-1, pp.64-69 - August 11 - 13 2005
- Bouix Emmanuel, Dalmau Marc, **Roose Philippe**, Luthon Franck - *A Multimedia Oriented Component Model* - AINA 2005 - The IEEE 19th International Conference on Advanced Information Networking and Applications - Tamkang University, Taiwan - March 28 - March 30, 2005
- Bouix Emmanuel, Dalmau Marc, **Roose Philippe**, Luthon Franck - *A Component Model for transmission and processing of Synchronized Multimedia Data Flows* - DFMA 05 (IEEE Int'l Conference - The First International Conference on Distributed Frameworks for Multimedia Applications) - 0-7695-2273-4, pp. 45-53, Besancon, France - February 6-9, 2005
- Bouix Emmanuel, **Roose Philippe** , Dalmau Marc - *A model for components used for transporting and handling synchronous information flows* - ICSSEA 2004, 17th International Conference on Software and Systems Engineering and their Applications - Paris, France - November 30 and December 1-2, 2004
- Michel Raphaël, **Roose Philippe,** Barbier Franck - *Identification Card for COTS-C use and development* - 15th IRMA International Conference - May, 23-26, New Orleans, USA - 2004
- Laplace Sophie , Dalmau Marc , **Roose Philippe** - *A formal method for accessing quality of service in distributed multimedia applications* - Software and Systems Engineering and their Applications - December 2-4 2003 - CNAM _ Paris, France -
- Laplace Sophie , Dalmau Marc , **Roose Philippe** - *New formal approach for QoS management in Distributed Multimedia Applications* - 6th IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS 2003) - s Univerity of Belfast, Northern Ireland, Lecture Notes in Computer Science (Volume 2839/2003), 978-3-540-20050-5, ISSN 0302-9743, pp 529-530 - Poster
- **Roose Philippe**, Dalmau Marc , Luthon Franck - *A distributed Architecture for Cooperative and Adaptative Multimedia Applications* - 2002 IEEE 26th International Computer Software and Applications Conference COMPSAC 2002 - IEEE Computer Society Press - 26-29 August 2002 - Oxford - England - ISBN : 0-7695-1727-7, ISSN : 0730-3157 - pp. 444-449 –
- Dalmau Marc , **Roose Philippe**, Luthon Franck - *A Software Architecture for Component Based Multimedia Applications* - 6th International Conference on Integrated Design and Process

Technology (IDPT 2002) - June 2002 - Pasadena (California) - USA. - ISSN 1090-9389 (8 pages) -

- **Roose Philippe** - *ELKAR : A Component Based Re-engineering Methodology to Provide Cooperation* - 2001 IEEE 25th International Computer Software and Applications Conference COMPSAC 2001 - IEEE Computer Society Press, pp. 65-70 - ISBN 0-7695-1372-7, ISSN 0730-3157 - Chicago - USA - October 2001 -
- **Roose Philippe**, Etcheverry Patrick - *Elkar-Toon : Providing Cooperation with Automation of Interoperability* - International Conference On Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Conputing (SNPD 2001) - ISBN : 0-9700776-1-0 - pp. 463-470 -August 2001 - Nagoya - Japan -
- **Roose Philippe** - *ELKAR - A Re-Engineering Methodology : Results* - 5th (WOON 2001) International Conference on Onject-Oriented Technology : The White Object Oriented Nights - 14-15 June 2001 - St Petersburg - Russia - 2001 –
- Dalmau Marc, Aniorté Philippe, **Roose Philippe** - *A Method for Designing Cooperative Distributed Applications* - COOP 2000 - 4th International conf. on The Design on Cooperative Systems - IOS Press ISBN 1-58603-042-6 - pp. 369-383 - Nice/Sophia Antipolis (France) - May 2000 -
- Aniorté Philippe, Dalmau Marc , **Roose Philippe** - *A Method for Re-Engineering/Redesigning Cooperatively Distributed Applications* - 5th World Conf. on IDPT - ISSN 1090-9389 - Dallas (USA) - June 2000 -
- **Roose Philippe**, Dalmau Marc , Aniorté Philippe - *Redesigning Distributed Applications To Provide Cooperation* - 14th Int'l Conference on On Object-Oriented Programming (ECOOP 2000) - LNCS N°1964 - ISSN : 0302-9743 - Cannes (France) - Juin 2000 - Poster
- **Roose Philippe**, Dalmau Marc , Aniorté Philippe - *Turning non-cooperative applications into cooperative ones by adding dynamics to the information system* - 5th Annual Conference of the UKAIS (UK Academy for Information Systems) - Mc Graw Hill - ISBN : 007 709755 6 - pp. 57-67 - Cardiff - U.K. - Avril 2000 -
- Aniorté Philippe, **Roose Philippe**, Dalmau Marc - *Using active Database Mechanisms to Build Cooperative Work* - Journal of Integrated Design and Process Science (JIDPS) - Vol. 3, N°1, pp1-14 - ISSN : 1092-0617 - March 1999 -
- Aniorté Philippe, **Roose Philippe**, Dalmau Marc - *Supporting Collaborative Work in Intranet Using Active DBMS implemented with Java* - Association for the Advancement of Computing in Education (AACE) - WebNet'97 - ISBN: 1880094274 - pp. 935-937 - Toronto (Canada) - November 1997 –
- Aniorté Philippe, Dalmau Marc , **Roose Philippe** - *Using Active Database Mechanisms to Build Cooperative Work* - Integrated Design and Process Science (IDPS) - Vol. 2,1998 - ISSN : 1090-9389 - pp. 119-127 - Berlin (Germany) - July 1998 -
- **Roose Philippe**, Dalmau Marc , Aniorté Philippe - *Active Interfaces to Facilitate Network Based Cooperative Work* - 6th Int'l Conf. - British Computer Science - Information Systems Methodologies (BCS/ISM) - Springer Edition - ISBN 1-85233-079-1 - pp. 102-116 - September 1998, Salford, England

Conférences nationales/workshops

- Christine Louberry, Marc Dalmau, **Philippe Roose** - Architecture Logicielles pour des Applications hétérogènes, distribuées et reconfigurables - 8ème Conférence Internationale sur les NOuvelles TEchnologies de la REpartition (NOTERE 2008) - Lyon, Juin 2008.
- **Roose Philippe** - *SI-COTS : Aide à l'intégration de COTS Products* - Workshop MADSI (Méthodes avancées de développement des systèmes d'information) - 25ème congrès INFORSID - Perros-Guirec, France - 22 Mai, 2007 -
- **Roose Philippe**, Dalmau Marc, Louberry Christine - *A unified components model for sensor integration into multimedia applications* - ACM SIGPLAN - Workshop at OOPSLA 'Building Software for Sensor Networks' - Portland, Oregon (USA) - 22-26 October 2006
- Louberry Christine, **Roose Philippe**, Dalmau Marc - *Modèle unifié de composants pour l'intégration de capteurs dans des applications multimédia* - Workshop OCM - Inforsid 2006 - Hammameth, Tunisie - 01-03 Juin 2006

- Michel Raphael, **Roose Philippe**, Barbier Franck - *Utilisation d'un Système d'Information pour la réalisation de prototypes dans le cadre d'un développement à base de COTS* - Workshop OCM - Inforsid 2006 - Hammameth, Tunisie - 01-03 Juin 2006
- Laplace Sophie, Dalmau Marc, **Roose Philippe** - *Prise en compte de la qualité de service dans la conception et l'exploitation d'applications multimédia réparties* - Inforsid 2006 - Hammameth, Tunisie - 01-03 Juin 2006 - Papier sélectionné pour la revie ISI - Ingénierie des Systèmes d'Information - Conception : patrons et spécifications formelles
- **Roose Philippe**, Dalmau Marc, Laplace Sophie - *Gestion de la qualité de service de la conception à l'exécution dans les applications distribuées multimédia* - Journées Composants 2005 - 4ème Conférence Francophone autour des Composants Logiciels - Le Croisic, France - 6-8 Avril 2005
- Laplace Sophie, Dalmau Marc, **Roose Philippe** - *QoS Oriented Design and Management for Multimedia Distributed Applications* - 6ème colloque francophone GRES 2005 - Luchon, France, ISBN : 2-9520326-5-3, pp.57-71 - 28 Février-3 Mars 2005
- Bouix Emmanuel, **Roose Philippe** , Dalmau Marc - *OSAGAIA : un modèle de composants pour le traitements de flux synchrones* - Workshop OCM-SI (Objets Composants et Modeles dans l ingénierie des SI) - Inforsid 2004 - Biarritz, France - 25 - 28 Mai 2004
- Dalmau Marc, Laplace Sophie, **Roose Philippe** - *Gestion de la QdS des applications réparties par adaptation dynamique au contexte* - Action Spécifique CNRS-GET - Systèmes répartis et réseaux adaptatifs au contexte - AS 150 (RTP 1 Réseaux et RTP 5 Systèmes Répartis) - 1er Avril 2004 - CNAM/Paris
- Dalmau Marc , Laplace Sophie , **Roose Philippe** - *Méthode de choix d'une configuration de composants logiciels optimale guidée par l'évaluation de la qualité de service* - Journées Composants 2004, ASF : Association ACM-SIGOPS de France - 17,18 Mars 2004 – Lille.
- **Roose Philippe**, Dalmau Marc, Luthon Franck , Laplace Sophie - *Gestion de la Qualité de Service par Reconfiguration Dynamique dans les Applications Interactives Multimédia* - Journées ALP (GDR du CNRS) - ACM/SIGOPS - Systèmes à composants adaptables et extensibles - Ed. INRIA - ISBN 2-7261-1229-3 - 17,18/10/2002 - Grenoble, France.
- Aniorté Philippe, **Roose Philippe** - *Ingénierie de systèmes distribués* - Journée du GDR I3 - 13 décembre 2001 - Lyon -
- Seyler Frédéric, Aniorté Philippe, **Roose Philippe** - *Une plate-forme pour gérer la coopération de composants distribués* - ALCAA Agents Logiciels et Activité et l'Apprentissage - pp. 152-163 - Biarritz - Septembre 2001.
- **Roose Philippe**, Aniorté Philippe, Dalmau Marc - *ELKAR : Une méthode de ré-ingénierie pour la mise en oeuvre de la coopération* - INFORSID 2001 (19ème Congrès Informatique des Organisations et Systèmes d'Information et de Décision) - Edition Inforsid - ISBN : 2-906855-17-0 - pp. 178-193 - Martigny - Suisse - Juin 2001.
- Dalmau Marc, Aniorté Philippe, **Roose Philippe** - *How to achieve synchronous and asynchronous communication in distributed cooperative systems using ECA mechanisms* - Simposio Español de Informática Distribuida (SEID) - pp. 333-344 - Torculo Edición - ISBN : 84-8408-060-9 - Santiago de Compostela (Spain) - February 1999.

Conférences invités

- 4ème Workshop RECAP - Présentation du projet ANR TCAP – Montpellier, Novembre 2007.

Position Paper

- **Philippe Roose** - *IS-COTS : a help to COTS Products Integration* -Third International Off-The-Shelf-Based Development Methods Workshop (IOTSDM '08) - *Position Paper* - Held in conjunction with the Seventh International Conference on COTS-Based Software Systems (ICCBSS '08), 25 february, Madrid, Spain, 2008.

Thèse

- **Roose Philippe** - *ELKAR - Ré-Ingénierie d'applications pour la mise en œuvre de la coopération : Méthodologie et Architecture* - Thèse de l'Université de Pau et des Pays de l'Adour - soutenue le 14 Décembre 2000.

Bibliographie

[Abts, 1998] Abts, Christopher M. & Boehm, Barry. "COCOTS (Constructive COTS) Software Integration Cost Model: An Overview", Los Angeles, USC Center for Software Engineering, University of Southern California, 16 pages, Août 1998.

[Adlrich, 2002] Jonathan Aldrich Craig Chambers. David Notkin. - ArchJava : connecting software architecture to implementation – 24th International Conference on Software Engineering (ICSE'02) – ACM Press - pp. 187-197 – New York, Mai 2002.

[AFNOR, 2004] AFNOR, Qualité des services Internet - Accès à l'Internet et services associés - Spécifications des critères de qualité du service réalisé, des niveaux de performance et de leur déclaration. Editions AFNOR. Février 2004.

[Aldrich, 2002] 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.

[Ammour, 2004] Salah-Eddine Ammour, Antoine Beugnard, Selma Matougui, Bruno Traverson - Architecture logicielle et abstractions de communication : une application dans le domaine du calcul scientifique - Les Nouvelles TEchnnologies de la RÉpartition - NOTERE'2004 – 27-30 juin 2004 - Saidia, Maroc

[Aniorté, 1999] Aniorté Philippe, Roose Philippe, Dalmau Marc - *Using active Database Mechanisms to Build Cooperative Work* - Journal of Integrated Design and Process Science (JIDPS) - Vol. 3, N°1, pp1-14 - ISSN : 1092-0617 - March 1999.

[Aniorte, 2000a] Aniorté P., Dalmau M., Roose P. – « A method for designing cooperative distributed applications » - Coop'2000 – 4th Int'l Conference on the Design of Cooperative Systems – IOS Press – « Designing Cooperative Systems - The use of Theories and Models » - ISBN 1-58603-042-6 – pp. 369-383 - Sophia Antipolis – France – Mai 2000.

[Aniorte, 2000b] Aniorté P., Dalmau M., Roose P. – « A method for re-engineering/redesigning cooperatively distributed applications » - 5th Int'l conf. On Integrated Design & Process Scienc4e (IDPT/IDEATE) - Dallas -

[Baldauf, 2007] Matthias Baldaf, A survey on context-aware systems – International Journal on Ad Hoc and Ubiquitous Computing, vol. 2, no°4, 2007.

[Bastien, 1993] J.M.C. Bastien, D.L. Scapin « Critères ergonomiques pour l'Evaluation d'Interfaces Utilisateurs », INRIA 1993.

[Bellissard 95a] L. Bellissard, S. Ben Atallah, A. Kerbrat, M. Riveill - Component-based Programming and Application Management with Olan - Workshop on Object-Based Parallel and Distributed Computation - Lecture Notes on Computer Science - Springer Verlag, 1996 – Tokyo - Juin 1995.

[Bellissard, 1995] L. Bellissard, F. Boyer, M.Riveill – « Construction and Management of Cooperative Distributed Applications » - Int'l Workshop on Object Orientation in Operating Systems (IWOOOS'95) - pp. 149-152 – IEEE - University of Lund – Sweden - August 1995.

[Blakowski, 1996] 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.

[Bouix, 2007] Emmanuel Bouix - *Modèle de composants et de flux pour applications multimédia distribuées dynamiquement reconfigurables* – Thèse de l'Université de Pau et des Pays de l'Adour – 11/2007.

[Bouix, 2008] Emmanuel Bouix, Philippe Roose, Marc Dalmau - The Korrontea Data Modeling - Ambi Sys 2008 - International Conference on Ambient Media and Systems - Quebec City, Canada – 2008.

[Bouraquadi, 2001] 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.

[Bruneton, 2002] 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.

[Bruneton, 2003a] E. Bruneton. Developing with Fractal. Tutoriel v.1.0, France Télécom R&D, septembre 2003.

[Bruneton, 2003b] Bruneton E., Coupaye T., Stéfani J.-B., The Fractal Component Model, Technical report, The ObjectWeb Consortium, September, 2003. version 2.0.

[Campbell, 1992] 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.

[Carlström, 2004] J. Carlström, T. Bodén. « Synchronous Dataflow Architecture for Network Processors ». IEEE Micro, vol. 24, n°5, pp. 10-18, septembre 2004.

[Carney, 1997] David Carney , "Assembling Large Systems from COTS Components: Oportunities, Cautions, and Complexities" , SEI , 14 pages , Pitsburgh , PA 15213-3890, June 1997.

[Carney, 2003] David J. Carney , Patricia A. Oberndorf , Patrick R.H. Plac , "A Basis for an Assembly Process forn COTS-Based Systems" , Technical Report , 47 pages , CMU/SEI-2003-TR-010, ESC-TR-2003-010 , May 2003
[CCI, 1989] Comité Consultatif International Télégraphique et Téléphonique, CCITT, Rec.I.350, General Aspects of Quality of Service and Network Performance in Digital Networks, International Telecommunication Union, Geneva, 1989.
[Chen, 2000] Guanling Chen, David Kotz - A Survey of Context-Aware Mobile Computing Research - Dartmouth College Technical Report TR2000-381, November 2000.
[Coupaye, 2001] Coupaye T., Lenglet R., Beauvois M., Bruneton E., Déchamboux P., « Composants et composition dans l'architecture des systèmes répartis », Journées Composants 2001 Besançon, octobre 2001.
[Coutaz, 2001] Coutaz Joëlle, Nigay Laurence, « Architecture logicielle conceptuelle des systèmes interactifs », Analyse et conception de l'IHM, Hermès 2001 Chapitre 7.
[Cui, 2001] Cui Y., Nahrstedt K., « QoS-Aware Dependency Management for Component-Based Systems », in Proc. of International Symposium on High Performance Distributed Computing (HPDC '01).
[Dalmau, 2002] Dalmau Marc , Roose Philippe, Luthon Franck - *A Software Architecture for Component Based Multimedia Applications* - 6th International Conference on Integrated Design and Process Technology (IDPT 2002) - June 2002 - Pasadena (California) - USA. - ISSN 1090-9389 (8 pages)
[Dalmau, 2004] Dalmau Marc, Laplace Sophie , Roose Philippe - *Méthode de choix d'une configuration de composants logiciels optimale guidée par l_évaluation de la qualité de service* - Journées Composants 2004, ASF : Association ACM-SIGOPS de France - 17,18 Mars 2004 - Lille
[David, 2005] Pierre-Charles David, Développement de composants Fractal adaptatifs : un langage dédié à l'aspect d'adaptation. Thèse de doctorat de l'Université de Nantes, juillet 2005.
[David, 2005] Pierre-Charles David, Thomas Ledoux - WildCAT: a generic framework for context-aware applications, Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing, ACM International Conference Proceeding Series; Vol. 115
[Dayal 88] Dayal U., Blaustein B., and Buchmann A. - The HiPAC Project : Combining Active Databases and Timing Constraints. SIGMOD RECORD, vol. 17, no 1 - pp 51-70 - Mars 1988.
[Dayal, 1988] Dayal U., Blaustein B., Buchmann A. – « The HiPAC Project: Combining Active Database and Timing Constraints » - SIGMOD RECORD, Vol 17, N°1, March 1988, pp 51-70.
[Demeure, 2002] DEMEURE I., *Une contribution à la conception et à la mise en œuvre d'applications sous contraintes de QoS temporelle, réparties, adaptables*. H.D.R., Discipline : Informatique, Université des Sciences et Technologies de Lille, décembre 2002.
[Dey, 1998] Anind K. Dey, Gregory D. Abowd, Andrew Wood - a framework for providing self-integrating context-aware services - ACM Symposium on User Interface Software and Technology – 1998
[Diot, 1995] C. Diot. « Adaptive Applications and QoS Guaranties ». In Proceedings of the International Conference on Multimedia and Networking, Aizu, Japon, 27-29 septembre 1995.
[Duclos, 2002] F. Duclos. Environnement de Gestion de Services Non Fonctionnels dans les Applications à Composants. Thèse de Doctorat, Université Joseph Fourier, Grenoble, France, octobre 2002.
[Elès, 1998] Eles P., Kuchcinski K., Peng Z., Doboli A., Pop P., « Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems," in Proc. Of the Design Automation and Test in Europ, DATE Conference, Paris 1998, pp. 132-138.
[Elès, 2000] Eles P., Doboli A., Pop P., Peng Z., « Scheduling with Bus Access Optimization for Distributed Embedded Systems », IEEE Transactions on VLSI Systems, vol. 8, No 5, 472-491, October 2000.
[Firesmith, 2003] Firesmith D.G., « Using Quality Models to Engineer Quality Requirements », Journal of Object Technology (JOT), 2(5), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, p. 67-75, September/October 2003.
[Franke, 1991] FrankE D. W., Purvis M. K., "Hardware/Software CoDesign: A Perspective," in Proceedings of the 13th International Conference on Software Engineering, 1991, pp. 344-352.
[Front, 1997] Agnès Front – « Développement de systèmes d'informations à l'aide de patrons : application aux bases de données actives » - Thèse de Doctorat - IMAG-LSR – Grenoble - 1997.
[Ghinea, 1998] G. Ghinea, J. Thomas – QoS Impact on user perception and understanding of multimedia video clips. 6th ACM International Conference on Multimedia- Bristol, England, September 12-16 1998.
[Hafid, 1998] Hafid A., Von Bochmann G., Dssouli R., «Distribued Multimedia Application and Quality of Service : A Review » Electronic Journal on Networks and Distributed Processing, N°6, 1998, p 1-50.
[Hagimont, 2002] 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.
[Hamida, 2008] Amira Ben Hamida, Frédéric Le Mouël, Stéphane Frénot, Mohamed Ben Ahmed –Une approche pour un chargement contextuel de services dans les environnements pervasifs – Numéro Spécial Revue ISI Objets Composants Modèles pour les Systèmes d'Information, Hermès, Juin 2008
[Harter, 1994] A. Harter, A. Hopper – A distributed location system for the active office. IEEE Networks, 8(1):6270, 1994.
[Heinman, 2001] G. T. Heineman, W. T. Councill. Component-Based Software Engineering: Putting the Pieces Together, Addison-Wesley, 2001.
[ISO, 1995] ISO, QoS Framework, ISO/IEC/JTC1/SC21/WG1 N9680, 1995.

[ISO, 2004] Software engineering -- Product quality -- Part 4: Quality in use metrics - SO/IEC TR 9126-4:2004
[Kapasi, 2003] U. J. Kapasi et al. « Programmable Stream Processors ». IEEE Computer, vol. 36, n°8, pp. 54-62, août 2003.
[Ketfi, 2004] Abdelmadjid Ketfi – Une approche générique pour la reconfiguration dynamique des applications à base de composants logiciels - Thèse de l'Université Joseph Fourier, Grenoble – Décembre 2004.
[Kiczales, 1997] G. Kiczales et al. « Aspect-Oriented Programming ». In Proceedings of 11th European Conference on Object-Oriented Programming, Jyväskylä, Finlande, 9-13 juin 1997.
[Kjær, 2007] A Survey of Context-Aware Middleware - Software Engineering - SE 2007 - Innsbruck, Austria, 2007.
[Kon, 2001] 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.
[Koshel 98] Arne Koshel, Peter C. Lockemann - Distributed Events in Active Database Systems – Letting the Genie out of the Bottle - Journal of Data and Knowledge Engineering (DKE) – Vol. 25 – pp. 11-28 – 1998.
[Laforest, 2008] Frédérique Laforest - De l'adaptation à la prise en compte du contexte – Une contribution aux systèmes d'information pervasifs – Habilitation à Diriger les Recherches, Université Claude Barnard, Lyon I, 2008.
[Laplace, 2005] Laplace Sophie, Dalmau Marc, Roose Philippe - *QoS Oriented Design and Management for Multimedia Distributed Applications* - 6ème colloque francophone GRES 2005 - Luchon, France, ISBN : 2-9520326-5-3, pp.57-71 - 28 Février-3 Mars 2005
[Laplace, 2006] Sophie Laplace, Conception d'Architectures Logicielles pour intégrer la qualité de service dans les applications multimédia réparties, Thèse de l'Université de Pau et des Pays de l'Adour, 2006.
[Laplace, 2006b] Sophie Laplace, Marc Dalmau, Philippe Roose, Prise en compte de la qualité de service dans la conception et l'exploitation d'applications multimédia réparties, Inforsid 2006, Hammameth, Tunisie.
[Laplace, 2007] Sophie Laplace, Marc Dalmau, Philippe Roose - Kalinahia : modèle de qualité de service pour les applications multimédia reconfigurables - Numéro Spécial Revue ISI "Conception : patrons et spécifications formelles" - Vol. 12 N°4/2007, ISBN: 978-2-7462-1968-7, 2007.
[Lavirotte, 2006] Stéphane Lavirotte, Jean-Yves Tigli, Diane Lingrand - Acquisitions et Modélisations pour la Gestion des Contextes – Action Adaptation dynamique aux environnements d'exécution – Paris, 2006.
[Layaida, 2004] Layaida O., Benatallah S., Hagimont D., « Reconfiguration-based QoS Management in Multimedia Streaming A plications », 30th Euromicro Conference, Track on "Multimedia & Telecommunications: Challenges in Distributed Multimedia Systems", Rennes, September 2004
[Lebastard 93] Franck Lebastard - CHOOE : Un gestionnaire d'environnement distribué - Rapport technique 93-22 – Sophia Antipolis – CERMICS - INRIA - Décembre 1993
[Lee, 1994] B. Lee, A. R. Hurson. - Dataflow Architectures and Multithreading . IEEE Computer, vol. 27, n°8, pp. 27-39, août 1994.
[Leung, 2002] Karl R.P.H. Leung , Hareton K.N. Leung ; "On the efficiency of domain-based COTS product selection method" , Information and Software Technology , 2002 , pages 703-715 .
[Li, 2001] Li B., Kalter W., Nahrstedt K., « A Hierarchical Quality of Service Control Architecture for Configurable Multimedia Applications », Journal of High Speed Networks, Special Issue on Management of Multimedia Networking, IOS Press, Vol. 9, pp. 153-174, 2001.
[Liebermann, 2000] H. Lieberman and T. Selker - Out of context: Computer systems that adapt to, and learn from, context – IBM System Journal - Volume 39, Numbers 3 & 4, MIT Media Laboratory 2000.
[Tigli, 2006] Jean-Yves Tigli, Daniel Cheung-Foo-Wo, Stéphane Lavirotte et Michel Riveill. « Adaptation au contexte par tissage d'aspects d'assemblage de composants déclenchés par des conditions contextuelles ». RSTI Série ISI - Adaptation et Gestion du Contexte, volume 11, numéro 5, pages 89-114, 2006. ISBN 2-7462-1672-8
[Lopes, 1995] C. Lopes, W. Hursch. Separation of Concerns. Rapport technique, College of Computer Science, Northeastern University, Boston, Etats-Unis, février 1995.
[Louberry, 2007] Louberry Christine, Roose Philippe, Dalmau Marc - Towards sensor integration into multimedia applications - 4th European Conference on Universal Multiservice Networks ECUMN'2007 - Toulouse, France - 12-14 February, 2007 - IEEE Computer Society Press ISBN : 0-7695-2768-X pp. 355-363.
[Magee, 1996] Magee J., Kramer J., « Dynamic Structure in Software Architectures », Proceedings of the fourth ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE'96), San Francisco, USA, p. 3-14, 1996.
[Magee, 1996] Magee J., Kramer J., « Dynamic Structure in Software Architectures », ***Proceedings of the fourth ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE'96)***, San Francisco, USA, p. 3-14, 1996.
[Magee, 1995] 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.

[Mao, 2001] 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.

[Martineau, 1996] P. Martineau, J-L. Labesse, C. Carmier, F. Cotard, C. Proust - A generic model for flexible manufacturing systems - (FMS) modelling, Workshop on Production Planning and Control – (WPPC'96), FUCaM/EIASM/ICM - pp. 246-250 - Mons – Belgique – Septembre 1996.
[Martineau, 1997] P. Martineau, C. Tacquard, A. Rouillon Génération automatique de la Conduite d'un système flexible de production - E3i Université F. Rabelais - Rapport interne n°197 - Tours - Octobre 1997.
[Medvidovic, 2000] Nenad Medvidovic, Richard N. Taylor, - A classification and comparison framework for software architecture description languages - IEEE Transactions on Software Engineering, 26(1):70-93 – 2000.
[Mehta, 2000] Nikunj R. Mehta, Nenad Medvidovic , Sandeep Phadke - Towards a taxonomy of software connectors - Proceedings of the 22nd international conference on Software engineering (ICSE 2000) – pp. 178 – 187, ISBN:1-58113-206-9, 2000
[Michel, 2004] Michel Raphaël, Roose Philippe**,** Barbier Franck - *Identification Card for COTS-C use and development* - 15th IRMA International Conference - May, 23-26, New Orleans, USA - 2004
[Michel, 2005] Michel Raphaël , Roose Philippe , Barbier Franck , " Information System for Evaluation of COTS" , 3rd International Conference on Software Engineering Research, Management and Applications (SERA2005) , Central Michigan University, Mount. Pleasant, Michigan, USA , Published by the IEEE Computer Society , Août 2005.
[Michel, 2006] Michel Raphael - Système d'Information pour l'Évaluation des Composants Logiciels sur étagère – (COTS Components) – Thèse de Doctorat, UPPA, Décembre 2006
[Morisio, 2000] Maurizio Morisio, Nancy Sunderhaft - Commercial-Off-The-Shelf (COTS): A Survey A DACS State-of-the-Art Report ; December 2000.
[Mungee, 1999] 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.
[NCube, 1999] NCube Cornelius, Maiden Neil A.M., "PORE: Procurement-Oriented Requirements Engineering Method for the Component Based Systems Engineering Development Paradigm", 13 pages , 1999 , International Workshop on Component-Based Software Engineering.
[Nguyen, 2007] Hai Quan NGUYEN - Conception d'architectures logicielles fiables par transformation et analyse formelle. – Thèse de l'Université de Lille1 – LIFL – 2007.
[Obendorf, 1997] Oberndorf Patricia, "COTS and Open Systems - An Overview". 1997 , available at http://www.sei.cmu.edu/str/descriptions/cots.html#ndi.
[Obendorf, 2000] Patricia Oberndorf , Lisa Brownsword , Carol A. Sledge , "An Activity Framework for COTS-Based Systems", Technical Report , 107 pages , CMU/SEI-2000-TR-010 ESC-TR-2000-010 , October 2000.
[Occello, 2003] 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.
[Parkin, 1992] Parkin M., Fluet C-D, Bade R., Introduction à la microéconomie moderne, Editions ERPI, 1992, ISBN 2-7613-0673-2.
[Pascoe, 2000] Jason Pascoe , Nick Ryan, David - Using while moving: HCI issues in fieldwork environments –ACM Transactions on Computer-Human Interaction (TOCHI) Vol. 7 , Issue 3 (09/2000) - Special issue on human-computer interaction with mobile systems (ISSN:1073-0516) - pp: 417 – 437- 2000
[Passama, 2006] Robin Passama Conception et développement de contrôleurs de robots – une méthodologie basée sur les composants logiciels – Thèse de l'Université de Montpellier II – LIRMM – 2006.
[Riveil, 2000] ARCAD – Architecture Répartie extensible pour Composants Adaptables – Projet RNTL – Poster - http://arcad.essi.fr/arcad/livrables/RNTL-Poster-ARCAD-2002.pdf
[Roose, 2000] Roose Philippe - ELKAR - Ré-Ingénierie d'applications pour la mise en oeuvre de la coopération : Méthodologie et Architecture - Thèse de l'Université de Pau et des Pays de l'Adour - soutenue le 14 Décembre 2000
[Roose, 2001] Roose Philippe - ELKAR: A Component Based Re-engineering Methodology to Provide Cooperation - 2001 IEEE 25th International Computer Software and Applications Conference COMPSAC 2001 - IEEE Computer Society Press, pp. 65-70 - ISBN 0-7695-1372-7, ISSN 0730-3157 - Chicago - USA - October 2001.
[Roose, 2006] Roose Philippe, Dalmau Marc, Louberry Christine - A unified components model for sensor integration into multimedia applications - ACM SIGPLAN - Workshop at OOPSLA 'Building Software for Sensor Networks' - Portland, Oregon (USA) - 22-26 October 2006.
[Roose, 2008] Philippe Roose - IS-COTS : a help to COTS Products Integration -Third International Off-The-Shelf-Based Development Methods Workshop (IOTSDM '08) - Held in conjunction with the Seventh International Conference on COTS-Based Software Systems (ICCBSS '08) in Madrid, Spain
[Sametinger, 1997] J. Sametinger. Software Engineering with Reusable Components, Springer-Verlag, 1997.
[Schilit, 1994] Bill Schilit, Marvin Theimer - Disseminating Active Map Information to Mobile Hosts - IEEE Network, September, 1994
[Segarra , 2002] 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.

[Singhoff, 1999] F. Singhoff. Spécification temporelle modulaire et support pour les applications multimédia réparties. Thèse de Doctorat, Ecole Nationale Supérieure des Télécommunications, Paris, France, décembre 1999.
[Steinmetz, 1996] R. Steinmetz – Human perception of jitter and media synchronization – IEEE Journal on Selected Area in Communication - 14,1 (January 1996), 61-72.
[Sun, 2003] Sun Microsystems. Enterprise Java Beans Specification 2.1 Final Release, 2003.
[Szyperski, 2002] C. Szyperski, D. Gruntz, S. Murer. *Component Software – Beyond Object-Oriented Programming*, Addison-Wesley, 2002.
[Tigli, 2006] Jean-Yves Tigli, Daniel Cheung-Foo-Wo, Stéphane Lavirotte, Michel Riveill. - Adaptation au contexte par tissage d'aspects d'assemblage de composants déclenchés par des conditions contextuelles - RTSI Série ISI, 11 (5), pages 89--114, 2006 ISBN 2-7462-1672-8
[Vogel, 1995] Vogel A., , Kerherve B., Bochmann G., Gecei., « Distribued multimedia applications and Quality of Service :- A Survey-» IEEE Mutimedia Journal, august 1995.
[Wang, 1996] Wang Z., Crowcroft J., « Quality-of-service routing for supporting multimedia applications », IEEE JSAC, Journal of Selected Areas in Communications, vol. 14, no. 7, pp. 1228-1234, September, 1996.
[Want, 1995] Roy Want, Bill N.Schilit, Norman I.Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, MarkWeiser - An overview of the PARCTab ubiquitous computing environment. IEEE Personal Communications, 2(6):2833, 1995.