Numéro de bibliothèque

THÈSE

PRÉSENTÉE Á

L'UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR

ÉCOLE DOCTORALE DES SCIENCES EXACTES ET DE LEURS APPLICATIONS

PAR

Marie Sophie VOISIN LAPLACE

POUR OBTENIR LE GRADE DE

DOCTEUR

Spécialité :

INFORMATIQUE

CONCEPTION D'ARCHITECTURES LOGICIELLES POUR INTÉGRER LA QUALITÉ DE SERVICE DANS LES APPLICATIONS MULTIMÉDIAS RÉPARTIES

Soutenue le 11 mai 2006

Après avis de :

M^me^ Isabelle GÉRARD DEMEURE, Professeur à l'E.N.S.T.
M. Jean-Pierre GIRAUDIN, Professeur à l'U.P.M.F. } Rapporteurs

Devant la Commission d'examen formée de :

M. Daniel HAGIMONT, Professeur à l'ENSEEIHT — Président
M. Franck LUTHON, Directeur de thèse, Professeur à l'U.P.P.A.
M. Marc DALMAU, Encadrant, Maître de Conférences à l'U.P.P.A.
M. Philippe ROOSE, Co-encadrant, Maître de Conférences à l'U.P.P.A. } Examinateurs

- 2006 -

Ce que ces trois années de thèse m'ont apporté dans les domaines scientifique et technique est présenté dans les pages qui suivent. Ce qu'elles m'ont apporté sur le plan personnel ne nécessite que ces quelques mots pour être résumé

« Connais-toi toi-même »

Temple d'Apollon, Delphes, Grèce.

A Christophe.

A Marie.

A Anna.

A Juliette.

A ceux qui viendront…

Remerciements

Je remercie du fond du cœur mon époux, Christophe Laplace, à qui cette thèse a donné autant de travail qu'à moi-même, un travail tout aussi gratifiant mais beaucoup plus ingrat car aucune reconnaissance officielle ne lui est accordée.

J'adresse toute ma gratitude à Marc Dalmau et à Philippe Roose qui m'ont encadrée au long de ces quatre années de D.E.A. et de thèse. Je les remercie pour leur investissement dans ces travaux et pour leur disponibilité de tous les instants. Je n'oublierai pas combien je me suis sentie encouragée par le caractère inépuisable des corrections et des idées de Marc qui mena en quelque sorte notre barque. Je dois à Philippe ce doctorat dans la mesure où il m'en souffla l'idée le jour de sa propre soutenance de thèse. Son optimisme à toute épreuve n'a ensuite jamais cessé d'être le moteur de notre progression.

Je remercie tout particulièrement Franck Luthon, directeur de cette thèse, pour sa franchise et sa rigueur qui m'ont permis de naviguer dans la bonne direction.

Ma gratitude va également aux rapporteurs, Isabelle Demeure et Jean-Pierre Giraudin, dont les remarques constructives et bienveillantes m'ont aidée à parachever ces travaux. Elle n'oublie pas Daniel Hagimont, président de la commission d'examen, qui a ainsi manifesté son intérêt pour nos recherches.

Je remercie l'Université de Pau et des Pays de l'Adour qui a facilité l'achèvement de cette thèse en m'octroyant dix mois de congé de formation professionnelle. Je suis consciente de l'effort ainsi consenti par mes collègues du Département Informatique de l'I.U.T. de Bayonne, qu'ils en soient tous individuellement et collectivement remerciés, enseignant ou non enseignant. Je voudrais souligner l'aide apportée par Pierre Cédras à qui je dois des pistes précieuses aussi bien en mathématiques qu'en micro-économie.

J'envoie un grand merci pour leur gentillesse et leur soutien à Emmanuel Bouix et Marion Latapy, futurs collègues, je l'espère.

Je n'oublie pas le Conseil Régional d'Aquitaine qui apporte une aide financière à notre équipe de recherche ainsi que le Laboratoire d'Informatique de l'U.P.P.A. qui l'héberge.

Enfin, je remercie les membres de mon équipe de baby sitter pour leur aide quotidienne — Maddy Lagrave, François et Françoise Voisin, Mayi et Jean-Paul Julien — ainsi qu'Emmanuel Bouix pour sa participation exceptionnelle.

Sans votre aide à tous, cette aventure humaine aurait été beaucoup plus difficile à mener voire impossible à terminer…

Résumé

La principale qualité d'Internet est d'être accessible partout, à tout moment et sur des supports de plus en plus variés. Son principal inconvénient est qu'il ne garantit aucune qualité de service si bien que l'utilisateur peut accéder à ce réseau dans des conditions tellement mauvaises qu'il ne peut pas l'utiliser. Une solution classique à ce problème réside dans la réservation des ressources lorsque les réseaux le permettent. Toutefois les supports et réseaux informatiques actuels sont très hétérogènes. Ainsi un utilisateur peut accéder à des pages Internet depuis son téléphone portable en passant par le réseau de téléphonie ainsi que par le réseau Internet et ceci afin d'utiliser un logiciel implanté sur un ordinateur personnel. Dans ces conditions, il est difficile d'envisager une réservation de ressources de bout en bout qui englobe tous les supports et systèmes existants. De plus, Internet ne propose aucun réel mécanisme de réservation de ressources si bien que cette solution n'est plus envisageable dès qu'il est utilisé à un endroit ou l'autre de la chaîne de transmission.

D'autre part, l'une des utilisations les plus répandues d'Internet est la manipulation de données multimédias. Or les applications utilisant ces ressources sont particulièrement sensibles à la qualité du service fourni par les réseaux de communication. En effet, pour être utilisables, elles doivent respecter des contraintes temporelles que le volume de leurs données rend problématique sur les réseaux actuels. Leurs performances sont alors dépendantes des variations du contexte d'exécution qui sont imprévisibles en particulier lorsque Internet est utilisé. De plus, l'une des caractéristiques des applications multimédias est l'importance de la perception qu'a l'utilisateur du service rendu ce qui n'est pas le cas d'applications de calcul scientifique, par exemple. C'est pourquoi le contexte d'exécution non informatique influe sur leurs performances. Il englobe aussi bien l'utilisateur que son environnement et est, de ce fait, caractérisé par son importante imprédictibilité.

Dans ces conditions, nous proposons d'assurer à l'utilisateur d'applications multimédias réparties une qualité du service optimale grâce à l'adaptation dynamique de l'application à son contexte d'exécution, informatique et non informatique. Cette solution, imposée par l'utilisation d'Internet et les caractéristiques du multimédia, répond à l'impossibilité d'une adaptation du contexte à l'application comme celle réalisée par les mécanismes de réservation de ressources. Elle nécessite une prise en compte de l'adaptation et de la qualité de service dès la conception et la réalisation de l'application. Ce constat nous amène à proposer, dans cette thèse, une méthode de conception d'application et un support d'exécution qui permettent d'adapter dynamiquement une application aux variations du contexte de manière à fournir et à maintenir la meilleure qualité de service possible aux utilisateurs, qualité définie individuellement par chacun. Cette adaptation est réalisée

dynamiquement par un intergiciel réparti qui ajoute ou supprime des composants, reconfigure et restructure les assemblages de composants formant l'application.

Dans ce but, nous avons défini un modèle d'application à base de composants logiciels qui peut être utilisé aussi bien pour décrire les aspects fonctionnels et structurels que pour évaluer la qualité de service. Pour atteindre cet objectif, notre modèle d'architecture d'application utilise deux niveaux structurels correspondant aux points de vue de l'utilisateur : le "groupe" qui représente le service global fourni à l'utilisateur et le "sous-groupe" qui exprime les fonctionnalités constituant ce service. En parallèle de ce modèle d'application et en cohérence avec lui, nous avons défini une manière originale de prendre en compte la satisfaction de l'utilisateur et non pas uniquement la qualité de service de type réseau puisque la qualité de service s'entend ici comme l'adéquation entre le service fourni et celui désiré par l'utilisateur. Elle regroupe alors aussi bien les aspects de synchronisation, d'ergonomie, de sécurité que de sémantique des données. Le modèle de qualité de service que nous proposons est alors basé sur deux critères, l'un appelé "contextuel " regroupant les caractéristiques variant en fonction du contexte et l'autre appelé "intrinsèque" regroupant les caractéristiques ne dépendant pas du contexte. Ces deux critères sont évalués en fonction des désirs de l'utilisateur.

De plus, que ce soit pour l'évaluation de la qualité de service ou pour la conception de l'application, nous avons utilisé un modèle et une représentation à base de graphes de flux orientés. En effet, l'une des principales caractéristiques des applications multimédias est l'existence de contraintes temporelles dans et entre les flux. Comme cette qualité de service doit être évaluée en permanence, il est nécessaire de trouver un moyen d'évaluation efficace et rapide. Nous proposons une évaluation répartie qui utilise les principes des machines à flots de données afin d'optimiser le parallélisme or, dans ce domaine, la représentation par graphes de flux est la plus efficace.

Ces différents modèles nous ont alors permis de proposer une méthode de conception d'application multimédia intégrant la qualité de service et un modèle de plate-forme qui supervise les applications lors de leur exécution. Cependant, nos travaux ont montré que la recherche d'une configuration optimale du point de vue de la qualité de service revient à une combinaison de problèmes d'ordonnancement et de routage connus pour être NP-complets. Par conséquent, nous nous sommes tournés vers une méthode itérative de recherche qui converge vers la meilleure configuration dans un contexte donné. Cette recherche est guidée à chaque étape par le souci de perturber le moins possible l'utilisateur et d'assurer ainsi la plasticité de l'application. De plus, la combinatoire est réduite par la prise en compte des vœux des utilisateurs qui permettent de classer les différentes structures en grandes familles de service. Nous proposons ainsi une heuristique adaptée aux contraintes du multimédia.

Ces modèles de plate-forme, d'application et de qualité de service, ainsi que les méthodes associées, graphes de flux et heuristique, ont été validés par un prototype développé avec LabVIEW de National Instruments.

En proposant de guider la conception, la réalisation et l'exécution des applications multimédias réparties vers l'obtention de la meilleure qualité de service possible pour les utilisateurs, nous espérons que nos travaux apportent une réponse aux problèmes de gestion de la qualité de service que les méthodes habituelles ne peuvent résoudre en particulier dans des environnement dépourvus de mécanismes de réservation de ressources. Ainsi nous avons tout d'abord élaboré des modèles cohérents de qualité de service et d'application, puis une méthode de conception reposant sur ces modèles et enfin un modèle de plate-forme d'exécution que nous avons validé par un prototype. L'objectif est en permanence de fournir le service le plus adéquat possible aux utilisateurs.

Abstract

The main quality of the Internet is to be available at any time, anywhere, and through an increasing variety of supports. However, its main drawback is that it does not guarantee any quality of service, which implies that the user may sometimes access this network in so bad conditions that he cannot use it. Resource reservation may help to solve this problem, when networks enable it. Today, however, support and computer networks are very heterogeneous. Indeed, a user can access web pages from his mobile phone via telephony networks and via the Internet network. He can then use a software installed on a personal computer. In these conditions, it is difficult to design end to end resource reservation, which handles all supports and existing systems. Besides, the Internet does not offer any resource reservation mechanism, which means that this solution cannot be considered as long as the Internet is used at one point or another of the transfer chain.

On the other hand, one of the most widespread use of the Internet is the handling of multimedia data. However, applications using these resources are particularly sensitive to the quality of the service provided by communication networks. Indeed, to be used, they have to respect temporal constraints, which is difficult on current networks because of the volume of their data. Their performance are then dependent on the variations of the run-time context, which are not predictable, in particular, when the Internet is used. Moreover, the importance of user perception of a provided service is one of the multimedia application features, in contrast with scientific computing applications for instance. This explains why external conditions have an impact on their performances. It includes the user as well as his environment, and is also therefore featured by its impredictability.

In these conditions, we would like to guarantee the user of a distributed multimedia application to benefit from an optimum quality of service in a given context, thanks to a dynamic adaptation of the application to its run-time context, hardware, software, network and external environment. This solution, imposed by the use of the Internet and multimedia features, is the counterpart of an adaptation of the context to the application, achieved by resource reservation. It requires taking into account adaptation and quality of service from design to implementation of the application. This situation leads us to suggest, in this thesis, a design method for the application and a support for the execution capable to dynamically adapt an application to the variations of the context. It will therefore provide and maintain the best quality of service that users can individually define. This adaptation is carried out dynamically by a distributed middleware which adds or removes components, reconfigures and restructures the set of components, that form the application.

To do so, we define an application model based on software components that can be used to describe the functional and structural aspects, as well as to assess quality of service. To achieve this goal, our application architecture model uses two structural levels corresponding to the view points of the user. Indeed, the "group" represents the service as a whole provided to the user and the "sub-group" expresses the functionalities constituting this service. In parallel with this application model and in coherence with it, we define an original way of taking into account quality of service: quality of service means here adequacy between the service provided and the one expected by the user. It therefore includes synchronisation, ergonomics and safety aspects as well as data semantics. The quality of service model that we suggest is then based on two criteria. The first one is called "contextual" and gathers different features depending on the context. The second one is called "intrinsic" and gathers features that do not depend on the context. These two criteria are assessed according to user's wishes.

Moreover, as long as evaluation of quality of service or application design are concerned, we use a model and a representation based on directed flow charts. Indeed, the existence of temporal constraints in and between flows is one of the main multimedia application features. However this quality of service must be permanently assessed. It is thus necessary to find ways for a fast and effective examination. We suggest a distributed evaluation which uses the principles of data flow systems, in order to optimise parallelism and, in this field, flow charts representation is the most effective.

These various models allowed us to suggest a method for multimedia application design integrating quality of service and a platform model which monitors applications during their run-time. However, our work point out that the search for an optimum configuration from the point of view of quality of service can be assimilated to a combination of scheduling problems and routing problems known as NP-complete. We thus suggest an iterative method for searching a better configuration which converges to the best configuration in a given context. At each stage, this research is aimed at disturbing the least possible user, in order to ensure application plasticity. Moreover, the combinatory is reduced since it takes into account users'wishes, which enables to classify the various structures into large categories of service. We thus suggest an heuristics adapted to the constraints of multimedia.

These models of platform, application and quality of service, as well as associated methods, flow charts and heuristics, were validated by a prototype developed with LabVIEW of National Instruments.

With the monitoring of the design, the realisation and the execution of distributed multimedia applications in order to obtain the best possible quality of service for users, we hope to provide a solution to the problems of quality of service management which cannot be solved through usual methods, particularly in environments without resources reservation mechanisms. Thus we first of all designed coherent models of quality of service and application, then we worked out a design method based on these models, and finally we developed a model of execution platform, which we validated by a prototype. The objective being permanently to provide the most adequate possible service to users.

Mots Clés

Qualité de service

Architecture logicielle

Applications adaptatives

Applications multimédias réparties

Composants logiciels

Modélisation de la qualité de service

Key Words

Quality of service

Software architecture

Adaptive application

Distributed multimedia application

Software component

Quality of service modelisation

Table des matières

Liste des figures

Liste des tableaux

Introduction

Il est tout à fait possible d'utiliser l'Internet pour regarder la télévision. Mais la qualité obtenue est tellement variable et peut être tellement médiocre que les industriels de la téléphonie mobile sont obligés de développer de nouvelles technologies pour proposer ce service à leurs utilisateurs. En effet, les terminaux informatiques que sont devenus les téléphones ont accès actuellement à plusieurs réseaux dont les plus performants en terme de bande passante sont l'Internet et le réseau U.M.T.S. — *Universal Mobile Telecommunication System* — support de la troisième génération de téléphones portables. Or aucune de ces deux technologies ne peut proposer une diffusion simultanée de programmes à des dizaines de millions d'abonnés en direct. Le problème sous-jacent est celui de la transmission d'un grand volume de données soumis à des contraintes temporelles alors que les bandes passantes aujourd'hui disponibles ne sont pas suffisantes. C'est pourquoi la Corée, seul pays où la télévision mobile est un réel succès, et les gouvernements européens, qui encouragent les industriels du secteur à tester différentes normes depuis ce printemps, tentent de se positionner sur un marché qui n'attend que la quatrième génération de téléphonie portable pour se démocratiser.

Or, si les résultats de la transmission télévisuelle sur l'Internet grand public et sur l'U.M.T.S. sont actuellement sensiblement les mêmes, ces deux réseaux sont fondamentalement différents par la garantie du service qu'ils fournissent. En effet, les réseaux téléphoniques permettent une réservation et une négociation de ressources informatiques alors que l'Internet assure seulement une garantie de type meilleur effort qui constitue en fait une absence de garantie. Cette impossibilité à prédire les temps de transmission et la bande passante disponibles sont rédhibitoires aussi bien pour la télévision que dans le cas plus général des applications multimédias qui manipulent un volume important de données devant respecter des contraintes en particulier de délais et de synchronisation. Puisque les applications multimédias réparties ne peuvent maîtriser les ressources auxquelles elles auront accès, concevoir, fournir et maintenir un service de qualité satisfaisante sur Internet semble une gageure.

D'autre part, le développement de l'informatique mobile, téléphone, agenda ou ordinateur portable, en parallèle avec la vulgarisation des ordinateurs personnels crée un nouveau besoin pour les utilisateurs qui ressentent la nécessité d'utiliser les mêmes logiciels sur des supports fixes et mobiles. Cette ubiquité des applications introduit de nouvelles fluctuations de service d'autant plus qu'apparaissent alors des discontinuités lors des changements de support ou de contexte d'utilisation.

Les caractéristiques du réseau Internet et les attentes des utilisateurs imposent donc aux concepteurs d'applications réparties sur ce réseau de tenir compte des variations de la qualité de service. Le problème est alors de réaliser et d'exploiter des applications capables de fonctionner dans un environnement mouvant et imprévisible. Une solution générique à cette problématique ne peut passer que par l'architecture logicielle. C'est pourquoi nous proposons dans ce mémoire de thèse une conception d'architecture logicielle originale pour intégrer la qualité de service dans les applications multimédias réparties. En focalisant notre étude sur les applications multimédias réparties sur Internet, nous avons choisi le domaine qui nous semble le plus représentatif de notre problématique. En effet, il est caractérisé à la fois par de grandes exigences de qualité, par une grande sensibilité au contexte et par la forte variabilité de ce dernier.

L'étude de ces spécificités nous a convaincus de la nécessité d'adapter les applications à leur contexte d'exécution qu'il s'agisse du contexte matériel, informatique — réseau, terminal etc. — ou environnemental — luminosité etc. — ou du contexte humain — habitudes, handicaps, etc. **Nous proposons donc une méthode de conception d'applications et un support d'exécution qui permettent d'adapter dynamiquement une application multimédia répartie sur Internet aux variations du contexte de manière à fournir et à maintenir la meilleure qualité de service possible aux utilisateurs, qualité définie individuellement**.

En traitant le cas des applications multimédias sur Internet, nous nous interdisons d'utiliser les méthodes de gestion de la qualité de service à base d'allocation et de négociation de ressources. Seule l'adaptation de l'application elle-même peut donc fournir une réponse aux exigences de qualité de service des utilisateurs. Cependant, nous puiserons dans toutes ces méthodes les éléments qui nous permettront de proposer un modèle original car nous ne souhaitons pas nous limiter à la qualité de service du réseau mais englober les aspects de synchronisation, d'ergonomie, de sécurité ainsi que la sémantique des données. Or une grande partie des systèmes et intergiciels existant pour gérer la qualité de service des applications multimédias réparties ne tient pas compte de ces aspects de qualité pour l'utilisateur.

D'autre part, la principale caractéristique des applications multimédias réside dans la manipulation d'un grand volume de données contraintes par des exigences temporelles. Or plus le volume des données à transmettre est grand, plus leur temps de transmission est élevé. Différents travaux ont montré dans ce cas l'efficacité des intergiciels en tant que supports d'exécution des applications multimédias réparties. A notre tour, nous proposons un modèle d'application multimédia répartie à base de composants logiciels, original par son architecture qui reflète la vision qu'a l'utilisateur du service qui lui est fourni, puis nous présentons un modèle de plate-forme qui autorise l'adaptation dynamique de l'application au contexte. Son originalité réside non seulement dans la définition entièrement dynamique des politiques d'adaptation mais aussi dans le souci permanent de ne pas perturber la qualité de service par ces adaptations.

Enfin, cette adaptation repose sur l'évaluation de la qualité du service fourni par l'application dans un contexte donné et sur la prédiction de la qualité des services que l'application pourrait fournir toujours dans ce même contexte mais avec une autre configuration. Conscients des fortes contraintes temporelles des applications étudiées, nous nous sommes inspirés des méthodes d'évaluation des systèmes complexes et répartis pour proposer un modèle d'évaluation de la qualité de service achevant ainsi de définir notre modèle de plate-forme.

Ces modèles ont été validés par un prototype qui met en évidence le bien fondé de centrer l'intégralité de notre approche sur l'utilisateur et l'efficacité de cette démarche pour obtenir une amélioration de la qualité du service fourni.

Afin d'exposer notre méthode de conception et notre plate-forme, nous présenterons dans une première partie les travaux traitant de la qualité de service, des applications multimédias et de l'évaluation des systèmes complexes qui ont inspiré nos recherches. Puis, dans une deuxième partie, nous proposerons nos modèles de qualité de service et d'applications multimédias. Enfin, la troisième partie permettra de présenter notre plate-forme d'exécution intégrant la qualité de service et de la valider par un prototype.

Partie 1: Etat de l'art

La démarche qualité en informatique concerne aussi bien la qualité matérielle qui peut être contradictoire avec la portabilité, que la qualité logicielle qui est difficile à obtenir et donc coûteuse. Elle englobe également la qualité de service qui est souvent négligée bien qu'elle soit la seule à refléter ce que l'utilisateur perçoit directement. A l'origine, ce terme a été utilisé principalement dans le domaine des réseaux pour décrire la qualité du service fourni aux applications par les systèmes de communications. Aujourd'hui, son acception s'étend. Elle quitte l'aspect purement technique pour rejoindre des préoccupations proches de l'utilisateur. Cependant, quels que soient le domaine et la définition utilisée, la mise au point de systèmes de gestion de la qualité de service s'articule autour de deux axes principaux [MAR 99] : l'expression de la qualité de service et les mécanismes de réalisation. Cette constatation guide notre présentation de l'état de l'art.

Dans une première partie, il nous faudra donc définir ce qu'est la qualité de service dans le cadre de notre étude avant de présenter différents modèles de qualité de service et de son évaluation. Nous y puiserons les éléments qui nous permettront de bâtir un modèle adapté aux applications multimédias réparties.

Puis, nous étudierons dans une deuxième partie les différentes manières de gérer la qualité de service en donnant tout d'abord une description générale puis en présentant différents travaux. Nous essayerons en particulier de mettre en exergue ce qui est utilisable dans le contexte particulier de l'Internet grand public et ce que l'absence de réservation de ressources exclut définitivement.

Une troisième partie nous permettra de spécifier les caractéristiques des applications multimédias réparties, de donner un exemple représentatif de leur modélisation et de présenter différents systèmes, plates-formes, intergiciels permettant de respecter les contraintes du multimédia et en particulier les contraintes temporelles. Leur étude nous permettra de proposer un intergiciel adapté au multimédia.

Enfin, il nous paraît nécessaire d'évaluer la qualité de service de manière dynamique. De plus, l'évaluation dans un système réparti complexe devant respecter des contraintes temporelles est un problème connu dans des domaines tels que l'informatique embarquée ou le temps réel. C'est pourquoi nous présenterons, dans une quatrième partie, les méthodes utilisées dans ces domaines qui nous permettront de proposer une évaluation dynamique et répartie. Ce faisant, nous aurons à résoudre le même type de problèmes de complexité algorithmique.

1.1. Qualité de service

L'étude de la qualité de service que nous présentons ici a pour objectif de délimiter le cadre de nos travaux et de proposer une définition, un modèle et une représentation adaptés à notre problématique.

1.1.1. Définition

La qualité de service — QdS, *Quality of Service*, *QoS*, en anglais — ne dispose pas d'une définition consensuelle qui conviendrait à tous les domaines auxquels elle s'applique. Cependant quel que soit le contexte, elle recouvre des propriétés non fonctionnelles d'une entité informatique, réseau, intergiciel ou application telles que les performances ou la sécurité. La norme ISO — *International Standard Organization* — la définit comme un ensemble de caractéristiques se rapportant au comportement collectif d'un ou de plusieurs objets [ISO 95].

C'est initialement pour les réseaux que cette notion fut utilisée. Elle est alors généralement décrite par des critères quantitatifs tels le délai, la gigue, *i.e.* la variation de délai, et la disponibilité liée au taux d'erreurs et donc au rapport signal sur bruit. La fonction d'un réseau étant la transmission, il s'agit bien là de propriétés non fonctionnelles. La qualité de service décrit ici les performances du réseau c'est-à-dire la qualité du service qu'il rend aux applications qui l'utilisent.

En revanche, dans le cas d'Internet, l'AFNOR — Association Française de Normalisation — propose un référentiel qui se place du point de vue de l'utilisateur du réseau. Ce référentiel de bonnes pratiques pour les fournisseurs d'accès [AFN 04] 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.

Or ce dernier critère ne constitue pas une propriété du réseau mais une propriété du flux transmis par le réseau [HAF 98]. Il ne s'agit donc plus de la qualité du service fourni par le réseau aux applications qu'il supporte mais de la qualité du service fourni à l'utilisateur final de ces mêmes applications.

Dans la même logique, il est donc opportun de définir la qualité de service d'une application vis-à-vis de l'utilisateur final. Dans cette perspective, il existe deux points de vue principaux qui mènent à deux définitions voisines mais cependant distinctes pour la qualité de service.

Le premier est le point de vue du système et la qualité de service est définie de la manière suivante pour les applications multimédias :

« *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.* » [VOG 95]

que nous traduirons par « La qualité de service représente l'ensemble des caractéristiques quantitatives et qualitatives d'un système multimédia distribué qui permettent d'atteindre la fonctionnalité requise pour une application. ».

Nous remarquons ici l'introduction de critères qualitatifs et donc subjectifs que l'AFNOR ne peut considérer puisque son objectif est de fournir des éléments objectifs de comparaison entre les différents fournisseurs d'accès Internet étant donné les enjeux commerciaux existant dans ce domaine. Dans le cas d'une application, cet aspect qualitatif ne peut être mis de côté car il décrit l'attente d'un utilisateur et permet d'évaluer la personnalisation toujours plus performante des services.

Le second point de vue est justement celui de l'utilisateur :

« *Quality of service is the collective effect of service performance which determines the degree of satisfaction of a user of the service* » [CCI 89]

que nous traduirons par « La qualité de service est l'effet collectif des performances du service qui détermine le degré de satisfaction de l'utilisateur du service. ».

Cette définition rejoint l'affirmation de [FIR 03] sur la notion plus générale de qualité selon laquelle elle est liée au degré avec lequel une entité « possède une combinaison de caractéristiques désirables ». La notion de désir comme de satisfaction nécessite de connaître l'opinion de l'utilisateur et donc que celle-ci soit récoltée avant l'évaluation de cette même qualité.

Selon le point de vue adopté, la qualité de service regroupe donc les caractéristiques d'un système ou le résultat de ces caractéristiques sur les performances du système vis-à-vis du service fourni à l'utilisateur. Nous choisirons de nommer qualité de service les performances du service fourni à l'utilisateur. Nous l'abrégerons en **QdS** pour la distinguer de la qualité de service des réseaux habituellement notée QoS. 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 les 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. Cependant, la première définition souligne à juste titre l'importance de définir les paramètres caractérisant la qualité de service et donc de définir un modèle de qualité de service comme ceux présentés dans le paragraphe suivant.

1.1.2. Modélisation

Les différentes définitions de la qualité de service induisent différents modèles. Nous considérons qu'un modèle inclut à la fois des caractéristiques et des mesures de qualité. Ainsi il fournit une base structurée sur laquelle il est possible d'identifier, d'analyser et de spécifier des exigences sur la qualité de service [FIR 03]. Ce modèle constitue donc le socle du système de gestion puisqu'il en donne les grandes orientations : ce qui ne sera pas décrit

par le modèle de qualité de service ne pourra être ni pris en compte ni, à plus forte raison, évalué.

Nous regroupons ici quelques modèles importants soit parce qu'ils sont dédiés aux applications multimédias et soulignent par conséquent les éléments que nous ne pourrons pas éviter de considérer, soit parce que le formalisme et les outils de modélisation qu'ils proposent pourront être adaptés à notre problématique.

1.1.2.1. Caractéristiques de qualité de service

La qualité de service est décrite par différents paramètres qu'il ne suffit pas de définir mais qu'il faut également structurer et dont il faut appréhender les articulations pour pouvoir définir la qualité d'une application à partir de leur évaluation.

a. Paramètres objectifs et subjectifs

De manière générale, il existe deux types de paramètres de qualité de service [HAF 98]. Les premiers sont les paramètres objectifs, mesurables et observables, tels ceux que nous avons cités pour les réseaux — bande passante, délai, gigue, taux de perte etc. Les seconds dépendent de l'équipement de l'utilisateur ou de son opinion et ne sont donc pas mesurables. Comme ils sont liés à l'utilisateur, l'AFNOR ne propose pas de les évaluer en raison de leur nature subjective. Dans un souci de généralité, elle n'utilise pas non plus des paramètres spécifiques à certains domaines tels que les informations psycho-visuelles ou psycho-auditives permettant de définir la qualité de la perception d'une vidéo ou d'une séquence sonore de façon objective.

En fonction de leur conception, les protocoles et services de communication offrent des indications sur les paramètres objectifs de leur qualité de service. Celles-ci sont limitées pour les protocoles de transport traditionnels comme T.C.P. et plus nombreuses pour les protocoles à haut débit tels A.T.M. — *Asynchronous Transfer Mode* — et X.T.P. — *eXpress Transport Protocol* — qui peuvent ainsi fournir des indications de sémantique plus élevée en prévenant l'utilisateur s'il ne peut pas disposer de la qualité de service demandée.

Cependant, la qualité de service d'une application grand public ne peut se limiter aux critères objectifs. En effet, sa description nécessite des notions de sémantique plus complexes comme la visibilité d'une image en fonction du contexte ou la compréhension d'une langue utilisée par un conférencier. Les modèles de qualité de service doivent donc représenter et structurer ces aspects en utilisant par exemple une hiérarchie comme dans les travaux que nous présentons dans le paragraphe suivant.

b. Hiérarchies

Pour [FIR 03], le modèle décompose le concept général de qualité pour créer une taxonomie des facteurs de qualité. Pour cet auteur, un facteur de qualité est un aspect, un attribut ou une caractéristique importante de haut niveau de la qualité d'un ou de plusieurs produits finis. De plus, le modèle doit fournir des critères de qualité et des mesures spécifiques pour « transformer les facteurs de qualité de haut niveau en des descriptions mesurables ».

Son modèle de qualité consiste en une hiérarchie de différentes composantes (cf. Figure 1). Le plus haut niveau est décrit par des groupes de facteurs de qualité qui peuvent, par exemple, être orientés concepteur s'ils concernent le développement ou la maintenance, ou être orientés utilisateur s'ils concernent l'exécution. Puis, viennent les facteurs de qualité qui concernent encore des aspects de haut niveau tels la réutilisabilité, la sécurité ou les performances. Ils sont décomposés en sous-facteurs décrivant des aspects de bas niveaux comme les temps de traitement ou la gigue pour les performances. Les critères de qualité sont ensuite définis. Ce sont des descriptions spécifiques fournissant des preuves sur ou contre l'existence d'un facteur ou sous-facteur de qualité : ils sont liés au domaine donc moins réutilisables que les facteurs. Enfin, les mesures de qualité quantifient les critères de qualité et les rendent mesurables. Elles doivent être objectives et non ambiguës. Cette hiérarchisation permet donc de relier une mesure à un facteur de qualité perceptible par l'utilisateur sans toutefois présumer des mécanismes qui permettent de dériver l'évaluation de l'un à partir de l'autre.

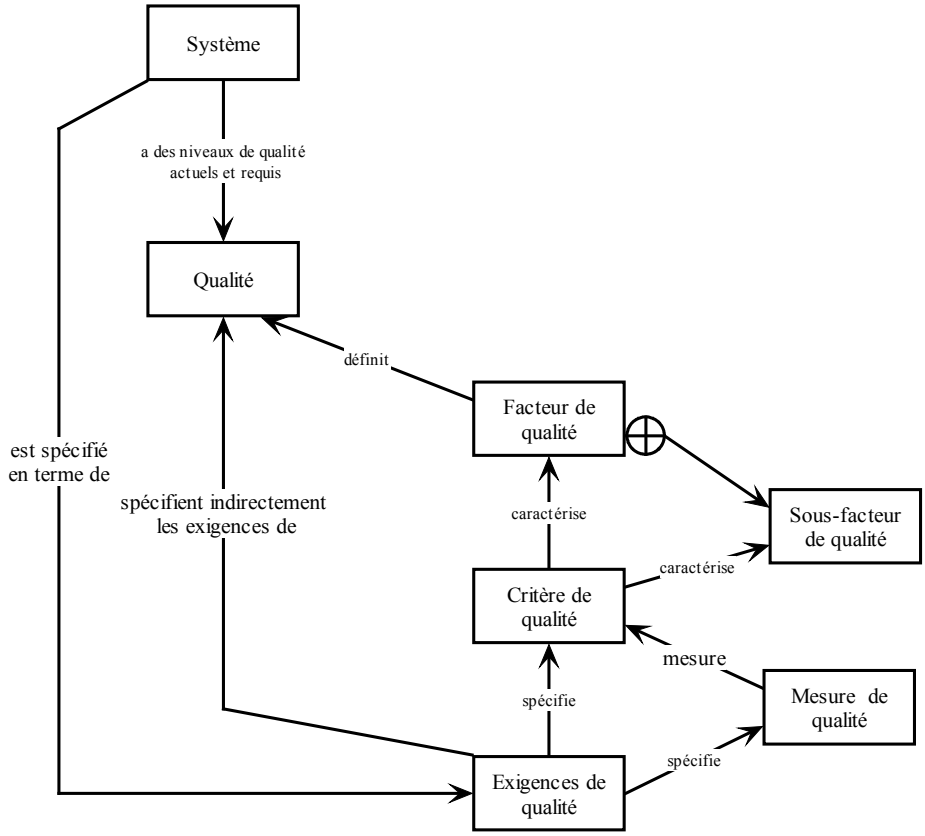


Figure 1 : Structure du modèle de qualité de service dans [FIR 03],

Un autre exemple de hiérarchisation des critères est donné par [GUX 02] comme support au langage HQML — *Hierarchical QoS Markup Language* — permettant l'évaluation de la qualité de service. La qualité de service est définie ici dans le cas particulier des applications multimédias réparties sur Internet et concerne donc la spécification et l'obtention d'une synchronisation et d'une intégration correctes des flux multimédias ainsi que la spécification et l'obtention de prédictabilité, de continuité et d'accessibilité. Les trois niveaux de spécification sont les suivants :

- le niveau utilisateur qui permet en particulier de spécifier des critères qualitatifs de qualité de service ainsi que les centres d'intérêt des utilisateurs ;
- le niveau application concernant par exemple la taille de l'écran et la résolution ;
- le niveau ressource système qui permet de spécifier, entre autres, la mémoire et la bande passante.

Cette hiérarchisation se distingue donc de la précédente par sa philosophie. En effet, elle est guidée par la structure d'exécution des systèmes multimédias répartis — un utilisateur, une application, un système sous-jacent — et non pas par la logique d'évaluation de la qualité de service — une mesure, un critère, un facteur. Cette différence formelle aboutit cependant à des structures similaires dans le sens où les informations sur le système sont les seules disponibles pour évaluer la qualité fournie aux utilisateurs.

Nous retiendrons donc de ces propositions la nécessité de structurer le modèle de qualité en une démarche descendante, en partant du plus haut niveau, l'utilisateur, et en aboutissant aux mesures et aux ressources du système. De plus, nous pensons qu'il est nécessaire de lier la définition du modèle à la définition et à la mise en œuvre des mesures. Le système de gestion de la qualité de service doit être intimement lié au modèle de qualité qu'il supportera. Cette idée justifie à elle seule le fait de proposer conjointement un modèle de qualité de service et un modèle de plate-forme qui permette sa mise en œuvre. Préalablement, il est toutefois nécessaire de définir comment passer d'une mesure à l'évaluation d'un critère ou facteur de qualité c'est-à-dire comment évaluer la qualité globale d'une application à partir des différentes mesures proposées par le modèle.

c. Composition de critères

Bien qu'il ne concerne qu'une qualité de type réseau, le routage sur la qualité de service — *QoS routing* — utilisé dans les télécommunications propose une solution intéressante pour déduire la qualité globale du service de la description de la qualité des différentes entités du système. En particulier, le routage à partir d'une métrique unique [AND 98] permet de traduire les besoins de qualité de service de l'utilisateur en une seule mesure. Différents paramètres sont définis puis évalués pour chaque entité du réseau. Il est alors nécessaire de définir des règles de composition distinctes pour chaque paramètre de manière à évaluer la mesure globale de ce paramètre de qualité. Les trois règles de base sont alors les règles additives, somme des chemins, utilisées pour le délai, la gigue ou le coût, les règles multiplicatives, multiplication de toutes les valeurs, utilisées pour la probabilité de transmission réussie et les règles concaves, le minimum, utilisées pour la bande passante [WAN 96]. Puis, le paramètre obtenu après composition se voit attribuer un poids de manière à définir la qualité globale à l'aide d'une moyenne.

Cette approche a retenu notre attention car elle peut s'appliquer à tout type de critère objectif ou subjectif. Elle souligne l'importance des règles de composition qui sont clairement mathématiques pour les critères objectifs mais qui devront être définies plus précisément pour les critères subjectifs. Néanmoins, le principal inconvénient de cette méthode réside dans le fait que l'optimisation de la qualité de service globale ne garantit pas chaque besoin en qualité de service. Si cet aspect est parfois critique pour les réseaux, il ne saurait l'être pour un utilisateur qui exprime des préférences de service et non des exigences dont le non-respect empêche le fonctionnement de l'application. De plus, nous allons voir que certains formalismes, tels les graphes et la représentation vectorielle, sont particulièrement adaptés à ce type de caractérisation de la qualité de service.

1.1.2.2. Formalismes

La notion de qualité de service est issue de l'étude des réseaux où la représentation à l'aide de graphes est omniprésente. Or un graphe peut se décrire par une grammaire et donc un langage. Nous pensons que cette filiation est à l'origine des deux formalismes principaux utilisés pour la qualité de service dont nous présentons ici quelques exemples : les langages et les graphes. D'autre part, il nous semble important d'exposer un exemple de formalisme permettant d'exprimer les contraintes temporelles de qualité de service car celles-ci sont une des caractéristiques fortes des applications multimédias.

a. Langage

Différents langages sont utilisés pour spécifier la qualité de service. Nous en avons retenu deux de manière à illustrer les approches basées exclusivement sur la qualité de service de type réseau et celles permettant éventuellement de prendre en compte la qualité fournie à l'utilisateur.

Le premier langage est le langage QDL — *QoS Description Language* [ZIN 97]— de l'infrastructure *QuO — Quality of Service for CORBA Objects*. Il décrit les besoins d'applications réparties en spécifiant à l'aide de trois sous langages :

- les contrats entre un client et un objet CORBA — *Common Object Request Broker Architecture* — composant l'application i.e. les différentes qualités de service qu'un objet fournit en fonction des ressources auxquelles il a accès ;
- les structures regroupant les objets et en particulier l'application ;
- les ressources du système.

QDL définit des régions de qualité de service reflétant l'état de l'accord de qualité de service relatif à une connexion à un objet lors de l'exécution de l'application [PAL 00]. Grâce à ces définitions, l'architecture QuO permet de passer dynamiquement d'une région à l'autre et permet ainsi l'adaptation dynamique des objets en fonction des ressources disponibles. Nous retenons tout particulièrement de ce langage sa capacité à exprimer formellement les différentes qualités de service fournies par un composant même s'il s'agit ici uniquement de caractéristiques techniques et s'il ne permet pas de décrire le comportement des objets mais uniquement le résultat. Ce langage met ainsi en œuvre un modèle hiérarchisé de qualité de service qui diffère uniquement de celui proposé par [GUX 02] (cf. §1.1.2.1.b p.35) par l'absence de représentation explicite du niveau utilisateur.

En revanche, l'Université de l'Illinois met en œuvre ce modèle grâce au langage HQML. Celui-ci permet une spécification hiérarchisée de la qualité de service en utilisant XML —*eXtensible Markup Language*. Ainsi une application multimédia peut spécifier toutes sortes de politiques ou d'exigences de qualité qui lui sont propres. Ce formalisme se veut universel dans le sens où il n'est pas couplé à un langage de programmation et qu'il est extensible à de nouveaux mécanismes de qualité qui n'auraient pas été pris en compte par ses concepteurs. Dans le projet initial [GUX 02], il interagit avec les systèmes de gestion de qualité génériques — des *proxies*— pour réaliser des garanties et des adaptations de qualité de service. L'objectif des auteurs est de définir un ensemble minimum de balises XML pour permettre aux concepteurs des applications d'exprimer les politiques et les exigences de qualité de service.

Malheureusement, bien qu'utilisable pour décrire la qualité perçue par l'utilisateur, il a été initialement conçu pour des mécanismes de réservation de ressources que nous ne considérons pas. En revanche, il propose comme support de modélisation des graphes et des vecteurs que nous pourrons adapter à notre problématique.

b. Graphes et vecteurs

L'utilisation de graphes pour représenter les réseaux et de vecteurs pour représenter les caractéristiques de ces réseaux est très répandue. Nous choisissons ici de présenter le formalisme utilisé autour du langage de qualité de service HQML et de l'environnement de programmation *QoSTalk* [GUX 02] car cette équipe partage avec nous l'objectif d'optimiser la qualité de service dans les applications multimédias réparties sur Internet.

Les applications considérées sont construites à base de composants logiciels (cf. Figure 2). Chaque composant a des entrées avec un niveau de qualité de service noté *Qin* et génère des sorties avec un niveau noté *Qout* [LIB 01]. Ces niveaux *Qin* et *Qout* sont des vecteurs composés des paramètres de qualité de service du niveau application tels que la taille de l'écran et sa résolution. Les auteurs utilisent également un vecteur *R* représentant les ressources nécessaires à chaque composant car leur système s'appuie sur un mécanisme de réservation de ressources.

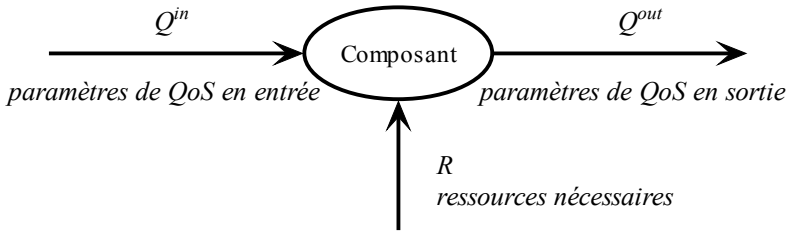


Figure 2 : Vecteurs de qualité de service par [LIB 01]

L'application étant représentée par un graphe, les spécifications de qualité de service sont représentées, elles aussi, par un graphe. De cette manière, la grammaire formelle *ConfigG* vérifie la logique des spécifications de qualité de service et en particulier les problèmes de compatibilité de formats entre les composants. Pour cela, elle définit la qualité de service d'un assemblage de composants à partir des vecteurs de ces derniers en utilisant des relations de type min ou max qui ne sont rien d'autre que des règles de composition concaves [WAN 96] identiques à celles utilisées dans les réseaux.

Ce modèle permet donc de représenter la qualité de service fournie par une application répartie construite à base de composants logiciels. Nous remarquons qu'il ne s'intéresse, à l'origine, qu'à l'étude de caractéristiques de type réseau mais que rien dans le formalisme n'interdit d'introduire des paramètres subjectifs dès lors qu'ils peuvent être représentés par les attributs d'un vecteur et que des règles de composition peuvent être définies.

c. Équations

Enfin, nous ne pouvons passer sous silence les modélisations spécifiques à l'un des aspects principaux de la qualité de service : les contraintes temporelles. Ainsi le projet POLKA — PrOcessus Légers et KAlité de service — utilise des équations et inéquations de qualité de service exprimées dans la logique temporelle Q.L. — *QoS Language*

[STE 93] — pour spécifier les contraintes temporelles d'applications multimédias. Ces contraintes sont dynamiquement modifiables par l'utilisateur. Par exemple, elles peuvent exprimer les exigences de la synchronisation voix-lèvres entre un flux audio et un flux vidéo. Pour cela, sont définis deux événements *e, e'* et $\tau(x,n)$ l'opérateur qui fournit la date de l'occurrence *n* de l'événement *x*. L'inéquation suivante [DEM 99] impose à la n-ième occurrence de *e'* d'être séparée d'au moins ε1 et d'au plus ε2 unités de temps de l'occurrence *n+k* de *e*. Quel que soit *n*, nous avons

$$\varepsilon 1 \leq \tau(e,n+k) - \tau(e',n) \leq \varepsilon 2 .$$

La plate-forme sous-jacente utilise tout d'abord une qualité de service utilisateur qui spécifie les contraintes que le système doit respecter en sortie, telles la synchronisation audio-vidéo, puis elle spécifie la qualité de service requise par le système multimédia pour satisfaire celle de l'utilisateur et ceci à l'aide des équations citées précédemment.

Ce formalisme est dédié aux applications temps réel c'est-à-dire aux systèmes devant respecter strictement des contraintes temporelles explicites. Il ne représente pas tant la qualité de service fournie par une application que les conditions nécessaires pour que l'application fournisse la qualité de service attendue par l'utilisateur dans la limite des ressources disponibles dans le système. Il se rapproche donc plus d'une spécification de comportement de l'application et n'est utilisable que si l'on peut intervenir de manière intrusive dans l'application par l'ordonnancement des tâches. Cependant, nous pensons qu'il peut être adapté pour représenter les performances temporelles des applications. En particulier, notons que l'utilisation d'équations à la place d'inéquations permet d'évaluer le respect de contraintes temporelles et donc, au final, certains aspects de qualité de service.

1.1.3. Qualité de service et utilité

De prime abord, il peut paraître surprenant de rapprocher la qualité de service, concept technique, de l'utilité, notion économique. Nous allons donc voir comment s'articulent ces deux aspects pour faire le lien entre la technique et la satisfaction de l'utilisateur.

1.1.3.1. Problématique des réseaux

La problématique du coût de la qualité de service dans les réseaux se traite de façon évidente à partir de notions utilisées en économie. Son objectif est de permettre aux fournisseurs d'accès de définir des politiques de prix capables d'inciter chaque utilisateur à choisir le service le plus adapté à ses besoins. Ces entreprises peuvent ainsi réduire la sur-allocation de ressources afin d'optimiser leurs bénéfices. Pour cela, il est nécessaire de considérer la fonction utilité [DAS 00]. Elle permet de modéliser les préférences des utilisateurs de réseau en définissant le prix qu'un utilisateur est prêt à payer pour certaines garanties de qualité de service. Cette fonction peut être définie à partir de paramètres de qualité de service comme le délai et le taux de pertes ou à partir des ressources allouées. Chaque type d'application peut être associé à une fonction utilité dans la mesure où il est possible de prédire de façon précise les attentes des utilisateurs.

Nous remarquons que cette prédiction est plus délicate à mettre en œuvre dans le cas des réseaux que dans celui des applications car les utilisateurs sont habitués à paramétrer

celles-ci. En revanche, nous allons montrer comment l'utilisation de la théorie de l'utilité et de celle relative aux préférences des utilisateurs peut enrichir la définition et la modélisation de la qualité de service. Nous allons, par conséquent, présenter succinctement ces notions.

1.1.3.2. Utilité, préférences et indifférences

En micro-économie, l'utilité est « l'avantage ou la satisfaction qu'une personne retire de la consommation d'un bien ou d'un service » [PAR 92]. Cette définition est donc très proche de la problématique de la qualité de service : évaluer la qualité de service consiste à estimer la satisfaction de l'utilisateur [CCI 89]. 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. D'autre part, l'utilité dépend de la quantité de biens consommés par le client comme la qualité de service dépend des valeurs, en terme de qualité, des paramètres ou critères fournis par l'application. La théorie de l'utilité va donc nous donner des pistes pour évaluer la qualité de service à partir de l'évaluation de critères de qualité. En d'autres termes, elle propose en économie une sorte de composition de critères (cf. §1.1.2.1.c p.37).

Le fondement de la théorie de l'utilité marginale — Alfred Marshall, 1890 — est la définition des préférences des consommateurs ou des utilisateurs en terme de quantité de biens consommés sachant qu'il ne peut influer sur leurs prix. Il y a une analogie entre cette impossibilité à agir sur les prix et l'impossibilité pour un utilisateur d'application répartie à agir sur les ressources informatiques nécessaires à l'obtention d'un critère de qualité. Pour cela, la micro-économie utilise des courbes de niveau d'utilité appelées courbes d'indifférence (cf. Figure 3) : elles sont 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. Ces courbes délimitent une région préférée et une région non préférée. 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 réseau mais dans une acception élargie à l'utilisateur.

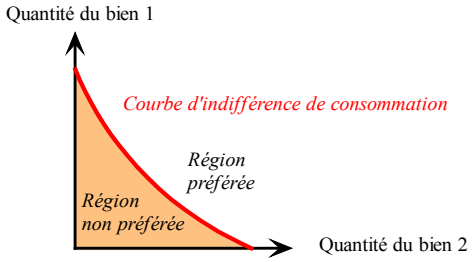


Figure 3 : Courbes d'indifférence en micro-économie

La théorie des préférences repose sur trois hypothèses fondamentales. La première est que les préférences ne dépendent pas du prix des produits. La seconde énonce que les préférences ne dépendent pas du revenu. Enfin, la dernière pose qu'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. Nous vérifions que la qualité de service respecte ces conditions puisque, d'une part, les préférences de l'utilisateur ne dépendent ni des

ressources nécessaires à un critère de qualité, ni de celles disponibles pour l'ensemble de l'application et d'autre part, l'utilisateur cherche son entière satisfaction. La carte des préférences est alors définie par une série de courbes d'indifférence qui, en tant que courbes de niveau, ne se coupent jamais ce qui signifie qu'il n'y a pas d'arbitrage à faire lorsque l'une des quantités est connue.

Enfin, l'allure de ces courbes dépend du degré de substituabilité des biens. Elle représente la définition de la fonction utilité à partir de la quantité de biens consommés. Pour la qualité de service, ces courbes représenteront la fonction de composition permettant de définir la mesure ou note de qualité de service de l'application en fonction de la mesure de qualité de service de ses critères : 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. Les courbes obtenues sont représentées par la Figure 4.

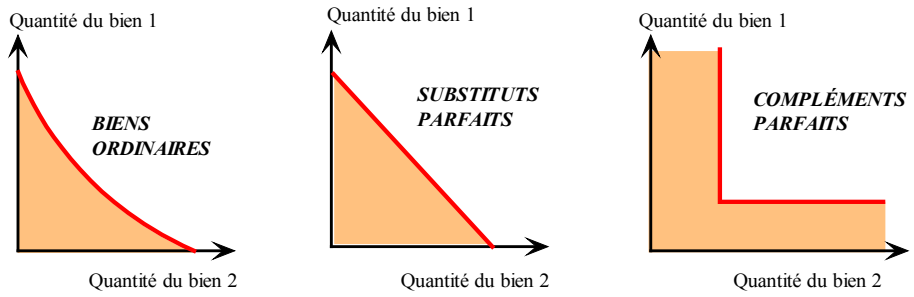


Figure 4 : Satisfaction de l'utilisateur en micro-économie

Cette approche économique nous inspirera pour définir la qualité de service d'une application à partir de la qualité de ses critères dans la mesure où il existe une relation de substituabilité entre ceux-ci. Une fois cette évaluation obtenue, elle servira au système pour gérer la qualité de service. Nous pensons qu'il s'agit d'une idée originale permettant une approche ascendante en passant de l'évaluation de critères de bas niveau à l'évaluation d'une qualité de haut niveau conceptuel réalisant une transcription *QoS-QoS* c'est-à-dire une transcription entre des qualités de service de différents niveaux [HAF 98]. Nous pensons ainsi définir une mesure de qualité traduisant fidèlement les attentes des utilisateurs.

1.1.4. Synthèse

Nous avons vu que la qualité de service peut être définie de différents points de vue allant d'une spécification très technique du réseau à l'explicitation des préférences de l'utilisateur. Nous veillerons à proposer une définition étendue de ce concept de manière à recouvrir aussi bien les paramètres objectifs que subjectifs intervenant dans l'évaluation d'un service. Il nous faudra considérer ceux qui sont mesurables, concrets, et d'autres qui ne le sont pas directement. Deux causes rendent un paramètre non mesurable. La première est sa complexité. Dans ce cas, il peut cependant être déterminé à partir d'autres critères par composition. La seconde est la subjectivité de certains paramètres qui ne dépendent pas de mesures mais d'une estimation utilisant les préférences des utilisateurs.

A partir de cette définition, nous nous efforcerons d'établir un modèle de qualité qui soit hiérarchisé d'une part selon la structure de l'application et d'autre part selon les services fournis par celle-ci. Nous choisirons alors un outil de modélisation adapté à notre qualité de service et aux applications multimédias réparties à base de composants. Ce modèle devra tenir compte de la perception que l'application a du contexte et des exigences de personnalisation des services exprimées par les utilisateurs. Nous pourrons alors utiliser l'analogie entre la qualité de service et l'utilité de manière à formaliser les préférences des utilisateurs en fonction des différents paramètres de qualité. Ainsi nous définirons des règles de composition spécifiques. Nous pourrons enfin proposer un système original de gestion de la qualité de service qui se démarquera des travaux existants dont nous allons maintenant présenter un aperçu.

1.2. Gestion de la qualité de service

Les modèles que nous avons présentés fournissent chacun une base structurée pour spécifier et évaluer la qualité de service. Ils sont utilisés dans des systèmes de gestion dédiés à un contexte ou à une problématique particulière. La gestion de la qualité de service décrit l'ensemble des activités mises en place dans un système pour surveiller, contrôler et administrer cet aspect [ISO 97]. Ses trois principales phases sont :

- la prédiction qui évalue l'environnement ;
- la définition qui permet la mise en œuvre des conditions nécessaires à l'obtention d'une qualité de service avant le début de l'application ;
- la phase opérationnelle chargée de la gestion en cours d'exécution de l'application.

Tous les systèmes ne mettent pas également en œuvre toutes ces étapes et tous ne le font pas de la même manière. Avant de présenter un échantillon représentatif, il est donc important de recenser les différents modes de gestion de la qualité de service.

1.2.1. Caractéristiques des modes de gestion

La principale difficulté de la gestion de la qualité de service dans les réseaux réside dans la traduction des caractéristiques de qualité offertes par le réseau en la qualité de service demandée par les applications. Cette gestion nécessite également une évaluation et une prise en compte de l'utilisateur. Quelle que soit l'approche, un consensus existe actuellement sur la nécessité de séparer la gestion de la qualité de la partie fonctionnelle en respectant ainsi les recommandations de la programmation par aspects — A.O.P., *Aspect Oriented Programming* [BOU 01] — qui prône la séparation des parties fonctionnelles et non fonctionnelles de la programmation. En revanche, que ce soit pour le niveau d'intervention ou le mode d'intervention plusieurs solutions sont proposées comme nous allons le voir.

1.2.1.1. Niveau d'intervention

La gestion de la qualité de service peut être mise en œuvre aux différents niveaux structurels des systèmes informatiques [GUX 02] :

- Le premier niveau concerne les réseaux et les systèmes d'exploitation. Si leur conception le permet, il leur est possible d'allouer des ressources spécifiques aux différentes applications qu'ils supportent de manière à respecter des contraintes de qualité explicites.
- Le deuxième niveau est celui de l'application qui peut être modifiée soit dans sa structure soit dans son fonctionnement. La gestion de la qualité doit alors veiller à ne pas rendre les applications trop complexes à mettre en œuvre. Elle doit

également prévoir des mécanismes d'adaptation de l'application à de nouveaux matériels ou réseaux. Par conséquent, la portabilité et les aspects matériels doivent être pris en compte dès le début de l'élaboration du système de gestion de la qualité de service.

- Enfin, le dernier niveau est celui des intergiciels. En fonction des réseaux et des systèmes d'exploitation utilisés, ils permettent soit de gérer les mécanismes de qualité de service de ceux-ci soit d'adapter ces services lorsqu'ils proposent seulement une garantie de type meilleur effort.

Un ou plusieurs de ces trois niveaux peuvent gérer l'adaptation d'un ou plusieurs niveaux [MAR 99]. Pour cela, l'acteur doit posséder une représentation explicite du niveau à adapter. Ainsi une application auto-adaptable doit être réflexive [ARC 00] dans le sens où elle a connaissance de son comportement et de sa structure.

1.2.1.2. Mode d'intervention

Quel que soit le niveau, il existe deux types principaux de mode d'intervention : la réservation de ressources et l'adaptation.

a. Réservation de ressources

Le premier mode d'intervention est la réservation et l'allocation de ressources qui sont souvent mises en œuvre à l'aide d'un intergiciel. Son principe est le suivant :

- l'application spécifie la qualité de service qu'elle requiert sous la forme d'exigences de ressources du système ;
- elle réserve les ressources correspondant à ses besoins si celles-ci sont disponibles ou alors négocie avec le système un compromis entre ce qu'elle nécessite et ce que celui-ci peut fournir. Cette négociation peut consister à choisir parmi des contrats prédéfinis ou non qui spécifient les différentes possibilités offertes par le système.
- le système lui alloue les ressources en fonction de l'accord trouvé.

Comme l'application peut s'exécuter en sachant totalement ce dont elle dispose, la garantie de service du réseau permet d'assurer à l'utilisateur une qualité de service fixe. Ceci est en général obtenu en utilisant des contrats où chaque composant stipule la qualité qu'il fournit en fonction des ressources allouées. Les classes de qualité de service permettent alors de définir les différentes qualités disponibles.

Malheureusement, cette méthode ne peut être mise en œuvre dans le cas d'Internet. En effet, ce réseau ne propose pas de mécanisme de réservation ni de garantie de ressources. C'est pourquoi nous développerons plus significativement le second mode d'intervention constitué par l'adaptation car c'est le seul que nous pourrons utiliser dans le cas général des applications réparties.

b. Adaptation au contexte

Dans les environnements où il n'est pas possible d'agir sur les services de communication, la qualité de service ne peut reposer que sur l'adaptation des applications au contexte.

Une première définition de cette notion consiste à considérer l'adaptation au contexte comme l'adaptation aux caractéristiques détectables et pertinentes de l'environnement informatique d'exécution et en particulier l'état du réseau ou la charge du terminal. En rendant les applications capables d'appréhender les caractéristiques de leur environnement, l'informatique nomade encore appelée diffuse ou ubiquitaire — *pervasive, ubiquitous computing* — cherche alors à rendre le support et le réseau de communication transparents à l'utilisateur. Celui-ci peut changer de support et se déplacer tout en bénéficiant des mêmes services.

Une seconde définition est issue d'une évolution similaire à celle connue par la notion de qualité de service et prend en compte l'utilisateur. Le contexte regroupe alors l'environnement informatique et non informatique comme la luminosité d'une pièce ou les préférences de l'utilisateur [RAK 01] [STE 02] [AYE 04]. Il peut être défini comme l'ensemble de toutes les informations environnementales qui sont pertinentes pour l'interaction entre l'utilisateur et l'application et qui peuvent être perçues par l'application [SAL 00].

Ces informations [SAL 00] peuvent être classées à partir de différents critères. Un de ceux-ci peut être le fournisseur : personnes, lieux, objets physiques, objets informatiques [SAL 00]. Un second est constitué par leur nature [ASP 03] :

- le contexte de traitement : processeur, mémoire, système d'exploitation, bande passante, niveau de batterie du terminal …
- le contexte lié à l'utilisateur : profil, localisation…
- le contexte physique : conditions climatiques, niveau sonore du bruit ambiant dans une pièce…
- le contexte temporel : date, historique d'activité…

Un système sensible au contexte ou adaptable au contexte — *context-aware computing* — est alors « un système qui peut découvrir et utiliser des informations contextuelles telles que la localisation de l'utilisateur, la date et l'heure, la proximité d'autres utilisateurs et d'autres dispositifs informatiques, les possibilités de connexion à un ou plusieurs réseaux, la bande passante, le réseau disponible, le niveau de bruit ambiant… » [ASP 03]. Ce système devra alors posséder un modèle, des outils de découverte du contexte et des moyens d'adaptation. Or, comme le contexte est changeant, l'adaptation devra être la plus dynamique possible.

1.2.1.3. Différents types d'adaptation

L'adaptation se divise en deux grandes étapes : la spécification des actions d'adaptation et leur exécution. Elle peut être statique ou dynamique à chacune de ces étapes. Une étape sera statique si elle est effectuée avant ou pendant le déploiement de l'application et dynamique si elle a lieu en cours d'exécution. Une solution mixte telle que spécifier statiquement l'adaptation et l'effectuer dynamiquement est possible. Elle peut cependant se heurter à l'impossibilité de prévoir toutes les situations contextuelles. C'est pourquoi une adaptation entièrement dynamique est préférable. D'autre part, l'évaluation du résultat peut, elle aussi, être statique ou dynamique selon que les variations de l'environnement sont prévues ou mesurées. Nous obtenons donc différents types d'adaptation comme résumé dans le Tableau 1.

		Adaptation de la QoS	
		Statique	Dynamique
Evaluation de la QoS	Statique	Tout est fait avant l'exécution (application paramétrable)	Toutes les variations de l'environnement sont prévues d'avance ainsi que les adaptations associées
	Dynamique	Reconfigurations manuelles de l 'application	**Application réflexive capable de s'auto évaluer et de s'auto adapter**

Tableau 1 : Possibilités d'adaptation

Les différents types d'adaptation peuvent également se distinguer par l'entité adaptée. Cette dernière peut être l'application mais aussi l'intergiciel ou le système d'exploitation ou plusieurs de ces éléments à la fois. Dans le cas d'applications réparties sur Internet, il n'est pas possible de proposer des mécanismes adaptés à tous les systèmes d'exploitation dont les utilisateurs pourraient disposer. C'est pourquoi une adaptation à ce niveau n'est pas envisageable. Seront alors adaptables uniquement l'application et l'intergiciel.

D'autre part, si ces deux entités sont construites à partir de composants, l'adaptation peut consister en une modification [LED 01] :

- des composants par paramétrage ou configuration ;
- de la nature des liaisons entre composants ;
- de l'architecture par suppression ou ajout d'assemblages de composants ou de liaisons.

Enfin, la stratégie ou politique d'adaptation peut être implicite [MAR 99] lorsque l'application ne la maîtrise pas. Ce sont alors, en général, des protocoles spécialisés qui la gèrent. Dans ce cas, l'adaptation peut avoir lieu uniquement pour des évolutions prédictibles et prévues du contexte puisque sa spécification est statique. A l'inverse, elle est explicite lorsque l'application la contrôle d'une manière procédurale si la gestion de l'adaptation est programmée dans l'application ou dans une entité spécialement conçue pour cela. Elle est déclarative si elle est seulement spécifiée. Dans les deux cas, le code gérant la qualité de service peut être soit mêlé au code fonctionnel de façon intrusive ce qui le rend difficile à réutiliser, à réadapter et à maintenir soit défini à part et la difficulté réside alors dans la traduction entre les spécifications de qualité de service et leur mise en œuvre, ce que [HAF 98] appelle la transcription QoS-QoS. Cette difficulté est typique de la composition des aspects nécessaire à la programmation par aspects qui guide ce type de solution [BOU 01] (cf. §1.2.1 p.45).

Leur caractère dynamique, leur niveau et leur mode d'intervention, ainsi que la manière dont ils sont implantés permettent de classer les différents systèmes de gestion de la qualité de service. Nous en présentons ici quelques-uns dans l'objectif d'illustrer les différentes méthodes proposées et d'identifier les éléments de réponse qui seront réutilisables dans le cadre de notre problématique. C'est pourquoi nous avons exclu les systèmes reposant uniquement sur une réservation de ressources.

1.2.2. Réservation de ressources et adaptation

La notion de qualité de service étant héritée des systèmes de télécommunication, nous présentons tout d'abord des solutions de ce domaine alliant la réservation et l'allocation de ressources à l'adaptation.

1.2.2.1. CAliF multimédia

La plate-forme CAliF Multimédia — CAliF pour *Cooperative AppLIcation Framework* — a été développée au sein du Laboratoire d'Informatique de l'Université de Franche-Comté L.I.F.C. Sa thématique générale concerne le travail coopératif à distance même si cette plate-forme traite plus particulièrement le partage de ressources distribuées, la diffusion de flux continus et la qualité de service [GAC 01] [GAC 01b]. C'est une plate-forme de développement pour les applications coopératives multimédias. Elle permet de créer des applications coopératives en faisant abstraction des problèmes de gestion de communication, de cohérence, de synchronisation et de flux multimédias. Elle utilise un bus CORBA qui permet de faire communiquer les objets répartis et gère ainsi l'hétérogénéité, l'interopérabilité, la portabilité et l'accès aux ressources.

Sa structure reflète ses quatre objectifs principaux qui correspondent chacun à un service traitant soit de la coopération soit du transport des médias continus.

La coopération est réalisée par le service Communication de groupe et le service Cohérence. Le premier de ces services diffuse les informations produites par les membres de l'application et gère les demandes d'adhésion ou de départ des membres du groupe. Il permet également la diffusion de messages tels que les contrôles des médias continus. Le second service assure la cohérence de la mémoire partagée répartie appelée aussi synchronisation de groupe [HAF 98] c'est-à-dire qu'il garantit que les objets de l'application répartie sont identiques pour chaque utilisateur.

Le transport des médias continus est, quant à lui, assuré par le service Audio/Vidéo et le service QoS. Le service Audio/Vidéo assure le transport de flux continus et, en particulier, la synchronisation inter-flux. Les commandes de contrôle le concernant transitent par le service Communication alors que les flux passent en dehors du bus CORBA pour des raisons de performance et de facilité de gestion de la synchronisation entre vidéo et son. Enfin, le service QoS traduit les requêtes de qualité de service et la réservation physique des ressources. Cette structure est décrite par la Figure 5.

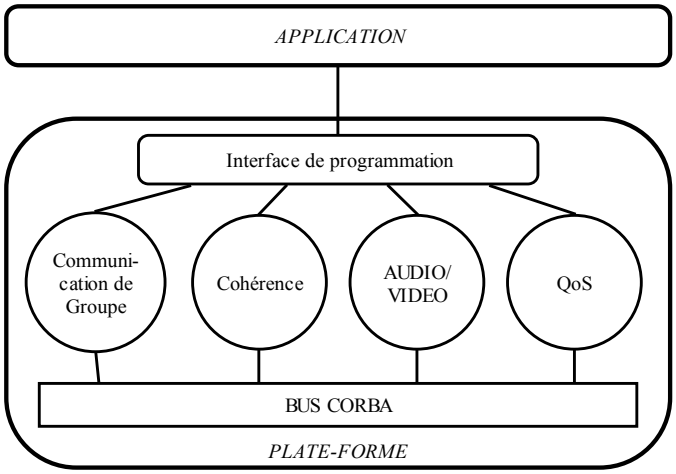


Figure 5 : Structure de la plate-forme CAliF multimédia d'après [GAC 01b]

La plate-forme CAliF Multimédia permet ainsi aux applications de réserver des ressources du réseau tout en adaptant leurs besoins aux ressources disponibles. Elle leur fournit un système de communication pouvant garantir un traitement particulier c'est-à-dire qu'elle fournit une réelle garantie de service. Par ailleurs, l'adaptation des ressources est obtenue par le paramétrage des routeurs. C'est pourquoi la solution proposée ne s'applique qu'à des réseaux privés — V.P.N., *Virtual Private Network*. De plus, la gestion globale de la qualité de service sur le réseau est basée sur un contrôle d'admission selon la priorité attribuée à chaque application.

Cette plate-forme utilise donc une adaptation du système d'exploitation et de l'application. Cette adaptation est spécifiée statiquement et réalisée dynamiquement. Les résultats de tests spécifiant les comportements de quelques réseaux sont proposés [GAC 01b] afin de permettre aux concepteurs d'applications de définir les règles dirigeant l'adaptation des routeurs. A cette occasion, les auteurs soulignent le nombre infini de situations contextuelles pouvant se présenter en fonction des utilisateurs, des caractéristiques de qualité de service et de l'état des systèmes.

Nous nous sommes intéressés à ces travaux car ils sont représentatifs d'une première approche de la qualité de service limitée aux caractéristiques des réseaux. La simplicité de cette définition alliée à la maîtrise des services de communication permet ici de fournir une réelle garantie de service aux applications et, indirectement, aux utilisateurs.

1.2.2.2. QuO

Parmi les systèmes à base de réservation de ressources, nous présentons QuO qui a la particularité d'adjoindre à chaque composant un mécanisme de qualité de service.

Quality Objects — QuO [ZIN 97] — est une infrastructure permettant de fournir de la qualité de service dans les applications distribuées sur les réseaux développée *par BBN Technologies*. Ces domaines d'application vont actuellement [SCA 03] des applications embarquées et donc temps réel aux applications réparties sur des grands réseaux — W.A.N., *Wide Area Network*.

QuO propose un intergiciel à base d'objets CORBA pour construire les applications. Son principe fondateur est d'étendre les interfaces fonctionnelles des objets CORBA pour spécifier et gérer la qualité de service. Issue de la programmation par aspects (cf. §1.2.1 p.45), la modélisation du système est scindée en deux chemins : un chemin fonctionnel capturant les flots d'informations et de contrôle et un chemin d'attributs de qualité de service permettant de déterminer si une interaction respecte les propriétés du système qu'elles soient temporelles ou fonctionnelles.

La gestion de la qualité de service dans QuO repose tout d'abord sur le contrôle du système d'exploitation et du réseau. En effet, les comportements de ceux-ci sont modifiés en fonction des informations contextuelles fournies de manière à obtenir la meilleure qualité de service possible. Puis l'application peut être modifiée à l'aide de contrats qui permettent non seulement de l'organiser et la déployer mais aussi de l'adapter.

Dans une première phase du projet, trois objets principaux étaient utilisés [LOY 98] :

- les contrats — *QoS contract objects* — qui organisent et contrôlent l'exécution de l'application ;
- les conditions du système — *system condition objects* — qui mesurent et récupèrent les informations sur les caractéristiques courantes du système ;
- les délégués — *delegate objects* — qui modifient le comportement du système en cours d'exécution.

Ces outils ne permettaient alors qu'une définition statique de l'adaptation réalisée ensuite dynamiquement. De plus, les contrats étaient séparés du code fonctionnel mais y faisaient référence si bien qu'ils n'étaient pas réutilisables. En outre, ils nécessitaient une connaissance de tous les états possibles de l'application ce qui est très difficile à mettre en œuvre [SCA 02].

Le projet a donc évolué en permettant une spécification dynamique de l'adaptation. Pour cela, il utilise des éléments de l'intergiciel appelés *Qoskets* [WAG 04] qui encapsulent le comportement adaptatif permettant la sélection dynamique de serveur. Ces extensions permettent de définir, pour chaque objet, la qualité de service qu'il peut fournir en fonction des ressources disponibles. En cours d'exécution, l'objet peut passer dynamiquement d'un état de qualité à l'autre. L'infrastructure permet d'ajouter une négociation des ressources. Ainsi, tant le système que l'application sont adaptés. La spécification de la qualité de service se fait à l'aide du langage QDL présenté précédemment (cf. §1.1.2.2.a p.38).

QuO propose un mécanisme dynamique d'adaptation des éléments de l'application et du réseau qui est à la fois portable et réutilisable. Cependant, il nécessite de modifier le code initial de l'application et des composants. Cette approche ne permet donc pas d'utiliser directement des composants déjà réalisés ni des composants de type COTS, *Component-off-the-shelf*.

1.2.3. Adaptation

Seule l'adaptation peut permettre de maintenir la qualité de service dans des réseaux dénués de véritable garantie de service tels ceux que nous comptons utiliser. C'est pourquoi notre intérêt s'est tourné plutôt vers les travaux d'adaptation que vers ceux de réservation

de ressources. Nous en présentons quelques-uns uns de manière à illustrer les différentes entités qui peuvent être adaptées.

1.2.3.1. POLKA

Le projet POLKA propose une méthode de développement et un ensemble d'outils pour la spécification, le développement et l'exécution d'applications réparties adaptables présentant des contraintes temporelles [DEM 02]. Cette étude a été menée au sein du Département Informatique et Réseaux de l'École Nationale Supérieure des Télécommunications de 1994 à 2001.

En utilisant un modèle fonctionnel de l'application, des spécifications des contraintes temporelles et des spécifications du système, une plate-forme d'objets CORBA supervise l'exécution de l'application. Elle permet de respecter les contraintes temporelles spécifiées dynamiquement par l'utilisateur grâce à un ordonnancement automatique [DEM 99] réalisé dans la limite des ressources disponibles. Si la qualité requise ne peut être atteinte, POLKA propose alors une adaptation de l'application soit par l'utilisateur soit par le système définie à partir d'une représentation de l'application sous forme de graphes.

Pour cela, la plate-forme gère la qualité de service à partir de deux critères, la qualité de service utilisateur et la qualité de service requise. La qualité de service utilisateur est la qualité du service souhaitée par l'utilisateur et s'exprime, par exemple, en termes de variation tolérée des délais d'affichage entre deux images — synchronisation intra-flux — ou de synchronisation audio-vidéo [DEM 99] — synchronisation inter-flux. Pour respecter cette qualité de service utilisateur, le système multimédia a besoin que les flux traités respectent des contraintes de qualité de service qui constituent la qualité de service requise. Elles s'expriment par les équations de qualité de service présentées en 1.1.2.2.c. Le comportement temporel d'un composant est alors spécifié par un contrat qui assure qu'il fournira la qualité de service désirée s'il dispose de la qualité de service requise en entrée.

En fonction de ces deux critères, la plate-forme POLKA supervise l'application en ordonnançant les invocations de méthodes. Lorsque les contraintes temporelles ne sont pas respectées, l'application doit adapter son comportement aux ressources disponibles. Ceci se traduira donc par une modification des équations de qualité de service en cours d'exécution. La rétroaction ainsi obtenue est essentielle lorsque la charge du système n'est pas prédictible et que la spécification de la qualité de service doit être dynamique.

Ces travaux soulignent l'importance du caractère dynamique de la définition de la qualité de service et de l'apport d'un système auto-adaptatif et rétroactif en particulier lorsque les besoins ne sont pas prédictibles hors ligne [SIN 98]. Nous remarquons cependant que l'objectif des auteurs est de traiter uniquement les caractéristiques temporelles de qualité de service même s'ils fournissent des méthodes de gestion — rétroaction — pouvant être utilisées pour assurer d'autres critères de qualité.

1.2.3.2. JQoS : adaptation des sources multimédias

JQoS est un système de vidéoconférence via Internet basé sur la qualité de service développé par le laboratoire *Multimedia Communications Research Laboratory* de l'Université d'Ottawa. Il a pour objectif d'être portable au niveau réseau comme au niveau plate-forme.

Le système JQoS propose une solution aux problèmes de qualité de service sur Internet en utilisant un mécanisme de contrôle dynamique de la qualité de service au niveau application [ZHU 01]. Celui-ci correspond à un algorithme qui reçoit les requêtes de qualité de service et ajuste la transmission des sources de manière à satisfaire le plus grand nombre de requêtes des utilisateurs de la vidéoconférence. Pour cela, il utilise des techniques de compression des médias continus, l'A.P.I. — *Application Programming Interface* —de Java J.M.F. — *Java Media Framework* —, R.M.I. — *Remote Method Invocation* — ainsi que le protocole de transport temps réel R.T.P./R.T.C.P. — *Real-time Transport Protocol/Real-Time Control Protocol* — utilisé par J.M.F.

Ces contrôles sont intéressants car ni le filtrage ni la distribution dégradée des flux multimédias ne peuvent être utilisés pour des vidéoconférences sur Internet. En effet, le filtrage de qualité de service, qui permet aux flux multimédias d'être transmis à des clients multiples avec différentes requêtes de QoS après une phase de négociation, se révèle trop lent pour les vidéoconférences. D'autre part, la distribution dégradée impose de connaître préalablement le codage des données ce qui n'est pas toujours réalisable dans ce contexte. Elle est obtenue par un encodage multi niveaux *i.e.* l'envoi simultané de flux dégradés hiérarchiquement à des groupes distribués. Chaque couche représente une certaine qualité de manière à ce qu'en choisissant le nombre de couches qu'il reçoit, le récepteur augmente la qualité de la présentation.

Le système JQoS tente d'éviter les oscillations de qualité de service c'est-à-dire la succession d'améliorations et de dégradations que peut connaître l'application lorsqu'elle n'atteint pas une position d'équilibre stable. Pour cela, une plus grande priorité d'utilisation des ressources du réseau et du système a été octroyée aux flux audio par rapport aux flux vidéos. Ainsi JQoS évite d'améliorer alternativement l'un des médias puis l'autre si cette amélioration provoque une dégradation du média restant ce qui constituerait des oscillations. De plus, l'auto-adaptation ne s'effectue que pour une dégradation de la qualité de service et non pour une amélioration de manière à éviter les oscillations. Une amélioration ne peut être obtenue qu'à partir de requêtes des récepteurs. Cependant, si ces requêtes sont incompatibles avec l'état du système, la source de transmission peut être séparée en différentes sessions correspondant aux niveaux de qualité de service de manière à satisfaire les différentes requêtes.

De plus, les objectifs de JQoS sont de respecter les trois caractéristiques principales des vidéoconférences via Internet basées sur la qualité de service [ZHU 01]. La première est la capacité d'auto-adaptation dynamique de la qualité de service en fonction des performances de transmission ponctuelles du réseau. La deuxième est l'existence d'un algorithme intelligent d'administration des requêtes de QoS tenant compte de l'état d'adaptation de la QoS du système. La dernière est la capacité du récepteur à adapter la QoS selon son intérêt personnel pour les flux reçus.

JQoS propose donc des solutions efficaces mais restreintes car elles concernent uniquement les vidéoconférences et elles ne traitent ni la synchronisation audio-vidéo — pourtant essentielle aux vidéoconférences —ni la planification de la qualité de service. En revanche, il permet de gérer la qualité de service sans utiliser de réservation de ressources. De plus, même si la qualité de service concernée est très spécifique, la méthode de contrôle utilisée présente une solution concrète pour certaines applications multimédias. Nous retiendrons la possibilité d'utiliser J.M.F. et R.T.P. pour gérer la qualité de service tout en assurant la portabilité des applications ainsi que la capacité d'auto-adaptation dynamique de la qualité de service en fonction des performances du réseau. En revanche, nous remarquons que la gestion de la qualité de service est dissymétrique : l'amélioration est provoquée par les récepteurs alors que la dégradation est automatique. Enfin, l'adaptation

est définie de manière statique pour certains paramètres tels la priorité plus grande donnée au flux audio vis-à-vis du flux vidéo. Or ce type de stratégie interdit la personnalisation des services nécessaire au cas particulier où un utilisateur présente un handicap comme une mauvaise audition. Dans ce cas, nous ne pourrons nous contenter d'une définition statique des paramètres de qualité de service subjectifs.

1.2.3.3. Agilos : adaptation de la structure de l'application et des ressources

Agilos — *Agile QoS* [LIB 01] — est un intergiciel pour la gestion de la qualité de service dédié aux applications multimédias sur Internet. Il réalise une adaptation par reconfiguration des applications et par allocation des ressources disponibles aux différentes applications concurrentes. Agilos coordonne et contrôle le déroulement de l'adaptation et en sélectionne les mécanismes.

Pour cela, les applications exportent une interface de contrôle (cf. Figure 6) qui permet de les adapter à l'aide de réglages spécifiques à chacune. En effet, ces interfaces activent des mécanismes adaptatifs qui agissent sur des paramètres réglables de qualité de service pour obtenir l'optimisation et la stabilisation de paramètres critiques. La qualité de service finale est donc définie par ces derniers paramètres de niveau application alors que les premiers englobent les demandes en ressources. Son obtention est réalisée en deux étapes semblables à celles mises en œuvre dans les systèmes asservis en automatique : l'identification et le contrôle.

L'identification permet la définition du modèle de l'application. Elle est réalisée avant l'exécution grâce à des tests permettant de spécifier les relations entre les paramètres réglables et les paramètres critiques. Celles-ci sont représentées par des graphes. Des composants spécifiques — *Qual Probe* — mettent en œuvre un algorithme de test dont les résultats guident la définition des stratégies d'adaptation exprimées à l'aide de règles et spécifiques à chaque application.

Le contrôle en cours d'exécution permet l'adaptation de l'application grâce à des mécanismes de prise de décision qui sont génériques contrairement aux règles. L'adaptation peut concerner deux niveaux, les composants et les services multimédias, puisqu'elle consiste soit en un réglage des paramètres d'un composant soit en une reconfiguration de l'application.

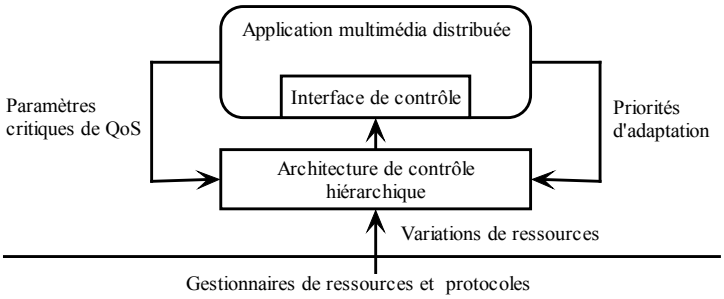


Figure 6 : Architecture globale d'Agilos

La gestion de la qualité de service se déroule en plusieurs phases lors de l'exécution. Tout d'abord, l'architecture de contrôle d'Agilos détecte les variations de ressources puis

décide quand et comment adapter les composants ou les groupes de composants, en particulier ceux fournissant un service multimédia. Cet intergiciel utilise alors la même technique d'adaptation des sources multimédias que le système JQoS (cf. §1.2.3.2 p.52). C'est pourquoi il propose une boucle de contrôle [LIB 01] du même type inspirée par les asservissements linéaires puisqu'elle réalise un correcteur de type P.I.D. — Proportionnel Intégral Dérivé — utilisé pour stabiliser les systèmes, optimiser les temps de réponse et annuler l'erreur constituée ici par une utilisation non optimisée des ressources. Enfin, une allocation de ressources est réalisée en fonction des poids alloués aux différentes applications et en fonction de leur sensibilité aux variations du contexte.

Nous retiendrons de ces travaux l'importance d'une adaptation intervenant à différents niveaux structurels de l'application ce qui nécessite une architecture guidée par la gestion de la qualité de service. L'adaptation est ici réalisée de manière dynamique mais spécifiée statiquement ce qui la rend inefficace dans des environnements non prédictibles. De plus, nous notons l'utilisation de poids pour réaliser l'allocation des ressources entre les différentes applications à la manière des priorités mises en œuvre dans CAliF multimédia (cf. §1.2.2.1 p.49). Nous reprendrons cette idée non pas pour l'allocation de ressources mais pour discerner l'importance des différents services que rend une application dans l'évaluation de sa qualité de service. Enfin, Agilos souligne l'intérêt de maintenir une qualité de service optimale et non pas seulement de l'obtenir. Pour atteindre cet objectif, il fait appel aux techniques de l'automatique qui permettent de s'affranchir de réaliser l'adaptation optimale en une seule intervention mais autorisent d'utiliser la convergence d'un système vers une qualité voulue. Dans des contextes dont la complexité empêche de définir la qualité optimale de l'application de façon analytique, ce principe propose alors une solution itérative intéressante.

Nous proposons dans la suite des travaux focalisés volontairement sur certains aspects de l'adaptation dont l'approche nous aidera à construire notre système de gestion.

1.2.3.4. CADeComp : adaptation du déploiement

CADeComp [AYE 04] [AYE 05] est un intergiciel sensible au contexte dédié à l'informatique mobile où les ressources sont limitées. Il propose un déploiement automatique à la volée et sensible au contexte : une application est installée au moment de son accès et désinstallée juste après la fin de son utilisation.

Pour cela, les applications sont distribuées sur plusieurs nœuds du réseau et formées par des composants réutilisables connectés entre eux via des ports. D'autre part, la représentation du contexte utilise un modèle de méta-données d'informations pour adapter le déploiement et en particulier décrire les instances des composants. Grâce à cela, les informations de contexte permettent de choisir les paramètres de déploiement et en particulier de spécifier la structure de l'application. En effet, celle-ci est décrite en précisant quels composants et quelles connexions sont obligatoires ou optionnels. Puis CADeComp propose un service de déploiement constitué d'un ensemble de composants adaptatifs placés au-dessus des outils de déploiement classique.

Ces travaux ciblés sur l'adaptation du déploiement présentent les mêmes inconvénients que tous les systèmes utilisant une adaptation spécifiée statiquement. En effet, la configuration déployée risque d'être rapidement obsolète en particulier pour des applications mobiles dont le contexte est très variable. Cependant, cette approche est intéressante car elle propose de modifier la structure de l'application en identifiant explicitement les composants et les connexions optionnels.

1.2.3.5. PLASMA : adaptation par agent mobile

Nous choisissons de présenter dans la partie adaptation les travaux sur PLASMA, un intergiciel à composants permettant de construire une application multimédia auto-reconfigurable même si l'exemple principal présenté par les auteurs utilise une adaptation des flux multimédias par un agent mobile déployé sur des *proxies* [LAY 04] [HAG 02]. En effet, PLASMA autorise une adaptation des paramètres fonctionnels et structurels de l'application par modification des liens entre composants et par modification des politiques d'adaptation.

Ce système permet d'implanter différents traitements sur les données multimédias à l'aide de composants séparés. Il propose une configuration et une reconfiguration dynamique puisque l'application est déployée automatiquement à partir des propriétés externes et reconfigurée par des composants spécifiques. Pour cela, les spécifications concernant la structure et le modèle de reconfiguration sont séparées de l'implantation de l'application respectant ainsi la séparation des aspects (cf. §1.2.1 p.45).

Le canevas logiciel proposé est structuré en trois niveaux :

- les spécifications qui utilisent le langage A.P.S.L. — *Adaptation Proxy Specification Language* [ATA 03] ayant évolué en *AdaPtive media Streaming Language* [LAY 04] — pour décrire la structure de l'application et les modèles de reconfiguration à l'aide de graphes exprimés en X.M.L. ;
- le contrôle qui traduit les spécifications APSL au niveau exécution ;
- l'exécution de l'application à partir de composants composites gérant l'envoi des flux multimédias selon le modèle client/serveur.

Les politiques d'adaptation sont exprimées à l'aide de règles proches des règles utilisées dans les bases de données actives ayant des contraintes temporelles [DAY 88]. Elles reposent sur des sondes — *Probes* — regroupant des événements qui définissent les paramètres observables, des *Conditions*, associant les événements aux actions de reconfigurations et enfin des *Actions* modifiant soit les paramètres fonctionnels du processus d'adaptation, soit la structure des traitements multimédias soit les politiques d'adaptation. Notons également que APSL fournit une vérification syntaxique et sémantique permettant principalement de vérifier la compatibilité des flux échangés entre différentes entités.

Ces travaux ont retenu notre attention d'une part pour la dynamicité totale de l'adaptation, d'autre part pour l'utilisation de graphes comme support d'expression aussi bien de l'application que de la gestion de la qualité de service.

1.2.4. Synthèse

La gestion de la qualité de service peut se faire à différents niveaux structurels — réseaux, systèmes d'exploitation, intergiciel ou application — et de différentes manières — réservation de ressources et adaptation. Dans le meilleur des cas, elle doit être dynamique, portable et réutilisable. Elle peut agir sur la structure, la composition et l'ordonnancement de l'application de façon différente selon qu'il s'agisse d'applications multimédias réparties ou d'applications temps réel embarquées. Son objectif est l'obtention et le maintien de la

meilleure qualité de service possible pour un contexte donné. Comme celui-ci évolue, elle devra suivre ses évolutions au plus près.

D'autre part, nous avons vu que, dans un souci de réutilisabilité, de nombreuses solutions proposées utilisent des composants logiciels en tant qu'unité qui peut être déployée de façon indépendante et composée [SPY 02]. Pour les mêmes raisons, nous utiliserons des composants ce qui nous interdira d'intervenir sur le code de l'application mais nous permettra d'introduire des composants de type COTS et surtout de séparer les aspects traitant des fonctionnalités et de la qualité de service.

De plus, notre problématique liée à Internet nous interdit de faire appel à un système basé sur la réservation de ressources, c'est pourquoi nous proposerons un mécanisme d'adaptation. Celui-ci concernera le niveau application car il est exclu de modifier les services fournis par le réseau Internet ainsi que de proposer l'utilisation de mécanismes du système d'exploitation dans une application distribuée sur des ressources hétérogènes.

Pour définir les conditions de cette adaptation, il est nécessaire d'étudier les spécificités des applications multimédias réparties et de leur qualité de service.

1.3. Applications multimédias réparties

Selon le domaine d'application, nous avons vu qu'il existe différents modes de gestion de la qualité de service. Il est donc nécessaire de définir précisément ce que sont les applications multimédias réparties pour concevoir un système de gestion de la qualité de service qui soit adapté aux spécificités de leur problématique. Dans cet objectif, nous présentons divers travaux proposant des modèles d'applications multimédias ou des architectures pour gérer la qualité de service de ces applications. Nous essayerons d'en tirer les grands principes que notre gestion de la qualité de service devra respecter.

1.3.1. Caractéristiques des applications multimédias

Contrairement à la qualité de service, la notion de multimédia est définie par différentes normes permettant de classer ces applications selon différents critères. Nous en donnons un aperçu avant de préciser leurs contraintes spécifiques concernant aussi bien les données que les services.

1.3.1.1. Définition des applications multimédias

Une application est dite multimédia si elle possède la « propriété de traiter différents types de média de représentation où le média de représentation est le type de donnée qui définit la nature de l'information comme décrite dans son format de codage. » [ISO 91]. Les données peuvent être discrètes telles des textes, des images fixes et des données numériques, ou continues comme le son et la vidéo.

Un média est dit continu s'il constitue une séquence continue d'échantillons de taille finie liés par des dépendances temporelles strictes [HAF 98]. Cette séquence est appelée flux. L'application multimédia doit donc respecter des contraintes temporelles pour ne pas dégrader voire détruire la sémantique des données continues [SUN 99] : en effet, elles ont une durée de validité au-delà de laquelle elles sont obsolètes.

En revanche, ces applications ne sont pas temps réel au sens strict du terme — temps réel dur ou *hard real time* — c'est-à-dire qu'elles n'ont pas à respecter strictement des contraintes temporelles explicites. En effet, elles peuvent accepter la perte de quelques informations ou leur accumulation éphémère dans la mesure où cela n'est pas perçu par l'utilisateur. C'est pourquoi certains les définissent comme temps réel souple ou mou — *soft real time* — puisque le temps est plus considéré comme un facteur de qualité que comme un facteur critique à l'inverse des systèmes temps réel durs. Les applications multimédias ne sont pas des applications critiques et leur caractère temps réel dépend des délais que l'utilisateur considère comme rédhibitoires. Or ceux-ci ne seront pas les mêmes pour une application de vidéosurveillance qui a des préoccupations proches du temps réel

que pour une application de vidéoconférence. Il nous a donc semblé opportun d'étudier les différentes classifications proposées pour les applications multimédias réparties.

1.3.1.2. Classification des applications multimédias

Les applications multimédias peuvent être classées selon leurs fonctionnalités ce qui permet d'en distinguer trois types [HAF 98]. Le premier regroupe les applications de présentation qui permettent la consultation d'informations multimédias préalablement stockées telles que les systèmes de vidéo à la demande. La difficulté est alors de respecter les contraintes temporelles tout en assurant un grand débit de données. Le deuxième type concerne les applications conversationnelles qui permettent la communication en temps réel entre personnes situées sur des sites distincts. Le dernier type regroupe les applications qui proposent des présentations tout en étant conversationnelles telles que le *E-learning* — l'enseignement à distance via Internet ou télé-enseignement — permettant l'accès à des ressources communes et la communication en temps réel entre les apprenants et les enseignants.

Le modèle que nous allons proposer concerne l'ensemble de ces types dans la mesure où ces applications ne sont pas temps réel dur car nous ne souhaitons pas restreindre notre étude plus que nécessaire. Cependant, cette première classification tient compte uniquement de ce que réalise l'application et non de la façon dont elle le fait. Il existe une autre classification fondée sur la prédictibilité de leurs comportements.

Classer selon la prédictibilité consiste à différencier les applications dont le comportement, et donc les besoins en ligne, peuvent être déterminés hors ligne, de celles dont le comportement est difficilement prédictible avant leur exécution [SIN 98]. Dans le premier cas, les applications multimédias peuvent être conçues en utilisant les solutions étudiées pour les systèmes temps réel fortement contraints. Dans le second cas, leur forte dynamicité ne permet pas d'appliquer directement des solutions préexistantes. Il s'agit des systèmes permettant à l'utilisateur de modifier dynamiquement la charge en lançant des applications ou de redéfinir la qualité de service désirée en cours d'exécution. Se pose alors, en outre, le problème des débits variables des flux multimédias.

Notre problématique de recherche englobe ces deux types d'application. Cependant, c'est dans le cas des applications non prédictibles que les solutions que nous proposerons seront les plus bénéfiques. En effet, notre principal objectif est de proposer une architecture maintenant la qualité de service désirée par l'utilisateur grâce au respect des contraintes imposées par le multimédia et ceci malgré les évolutions possibles de l'environnement d'exécution. Notons d'ailleurs que les variations du contexte d'exécution sont, elles aussi, imprédictibles. En effet, si l'évolution du trafic sur Internet suit à grande échelle des oscillations identifiables en fonction des heures de la journée, des jours de la semaine, de la période dans l'année et des habitudes des différents pays, à l'échelle d'un utilisateur, il n'est pas possible de prédire son évolution. Or cet aspect imprédictible tant des applications que du contexte, nous imposera une gestion de la qualité de service entièrement dynamique puisque nous ne pourrons pas définir les politiques d'adaptation préalablement à l'exécution (cf. §1.2.1.2.b p.46).

1.3.1.3. Exigences des applications multimédias

Les spécificités des applications multimédias sont principalement liées à la manipulation de flux continus dont l'utilisateur perçoit directement la qualité. C'est

pourquoi elles imposent aux systèmes des exigences concernant principalement les données et les services.

a. Données

Les applications multimédias génèrent de grands débits de données sur le réseau lorsqu'elles sont réparties. Le débit est exprimé le plus souvent en nombre de bits transmis pendant une certaine période de temps. Il s'agit par exemple de 2 Mbps pour le codage vidéo MPEG. Ce débit, qui illustre la bande passante utilisée, peut être réduit par compression des données mais cette dernière réduit la flexibilité et introduit des temps de latence additionnels nécessaires aux compressions et décompressions [HAF 98].

Ce débit élevé est d'autant plus problématique que ces données doivent respecter des contraintes temporelles. En effet, les flux doivent être synchronisés de manière intra et inter-flux [HAF 98].

La synchronisation intra-flux concerne la synchronisation des données entre elles. Ainsi les applications multimédias étant des applications contraintes temporellement, une donnée reçue trop tard est une donnée perdue. D'autre part, l'ergonomie et la sémantique des flux imposent une continuité de la présentation du média qu'il soit audio, vidéo ou plus classique comme des textes. Ces deux contraintes s'expriment par des relations temporelles entre les données d'un même flux.

D'autre part, les flux doivent généralement être synchronisés entre eux et en particulier le son et l'image d'une vidéo : c'est la synchronisation inter-flux. Elle peut aussi concerner la synchronisation entre un flux vidéo et un média discret dans le cas de la restitution en temps réel de commentaires sur un document composé.

Enfin, la dernière synchronisation concerne la synchronisation de groupe dans les applications réparties. Son objectif est d'assurer que tous les utilisateurs aient accès en même temps aux mêmes données c'est-à-dire que la contrainte WYSIWIS — "*What You See Is What I See*", "je vois ce que tu vois" — soit respectée.

b. Services

Comme l'utilisateur perçoit directement les flux multimédias, les applications multimédias doivent proposer une garantie de service. A l'image de celle reflétant la qualité des systèmes de communication, elle peut être spécifiée de différentes manières. Une garantie déterministe assure, par exemple, un délai maximum entre deux données. Pour caractériser ce même délai, une garantie statistique assure qu'un certain pourcentage de données aura un retard inférieur à une valeur fixe. Enfin, une garantie de type "meilleur effort" est utilisée actuellement dans des réseaux comme Internet et ne donne en fait aucune assurance sur le résultat. En outre, dans le cas d'applications réparties, ces diverses contraintes imposent de réduire le temps et les ressources requises afin de délivrer les mêmes données aux différents utilisateurs puisque les ressources globalement disponibles sont incompressibles.

Construire ces systèmes de garantie de service ne peut se faire qu'en concevant et utilisant un modèle d'application adapté à la gestion de la qualité de service dans le cas particulier des applications multimédias.

1.3.2. Exemple de modélisation : Agilos

Les applications multimédias sont dirigées par le transfert de flux continus d'informations. Or les systèmes dominés par les données sont généralement représentés par des graphes orientés où les nœuds représentent des traitements dénués d'effet de bord et les arcs indiquent l'ordre de réalisation de ces traitements [COR 99]. De plus, la problématique du multimédia réparti est liée à celle du temps réel réparti où la représentation par graphe est omniprésente. Ces considérations nous amènent à choisir de modéliser les applications à l'aide de graphes.

Parmi les différents travaux que nous avons déjà cités et qui les utilisent, POLKA, QuO, Agilos ou PLASMA (cf. §1.2 p.45), nous choisissons de développer plus particulièrement les recherches concernant l'intergiciel Agilos (cf. §1.2.3.3 p.54) car il propose un environnement visuel très complet de programmation appelé *QoSTalk* [GUX 00] où le dessin des graphes de configuration et la spécification de qualité de service associée (cf. §1.1.2.2.b p.39) sont réalisés grâce à un outil nommé *Visual Hierarchical QoS Editor* [GUX 00b].

De plus, ces propositions sont représentatives non seulement des applications multimédias réparties mais également de celles devant respecter des contraintes temporelles. Leur modélisation utilise des graphes et les grammaires formelles associées.

1.3.2.1. Graphes

Le modèle d'application utilisé dans l'intergiciel Agilos adapte au multimédia les travaux du projet EPIQ qui proposait l'utilisation de graphes acycliques orientés [HUL 97] dans le cadre de l'étude de la qualité de service des systèmes temps réel.

Le modèle définit des composants et des liens entre composants. Trois niveaux de composants sont utilisés [GUX 01] : un composant atomique — que d'autres nomment primitif — qui réalise une fonction multimédia basique, un composant composite regroupant un ensemble de composants atomiques sur un seul site hôte et enfin un groupe de composants composites équivalant, par exemple, à un client ou un serveur sur les hôtes finals. D'autres part, trois sortes de lien monocanal ou multicanal sont utilisés [GUX 02] : un lien fixe représentant un canal de communication filaire qui ne peut pas être interrompu ou déplacé en cours d'exécution, un lien d'hôte mobile pour un canal de communication sans fil entre deux composants et un lien d'utilisateur mobile lorsque celui-ci se déplace d'une machine à une autre.

A partir de ces différentes entités, Agilos propose une modélisation structurée en deux niveaux hiérarchiques [LIB 01]. Le premier niveau repose sur les composants atomiques. L'application est alors divisée en tâches qui forment un graphe orienté acyclique appelé *graphe de configuration* de l'application (cf. Figure 7 p.63). C'est un graphe des flux sur lequel les données multimédias circulent entre les fournisseurs de service et l'utilisateur final. Les nœuds représentent les composants et les arcs les flux multimédias. Le second niveau est construit à partir de composants composites définis comme des assemblages de composants atomiques tels les clients ou les serveurs, chacun étant sur un seul hôte.

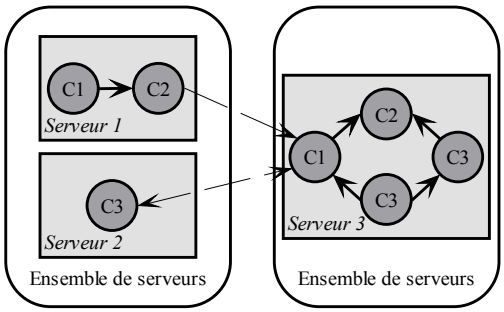


Figure 7 : Graphe de configuration générique [GUX 01]

De plus, les dépendances des composants vis-à-vis des autres composants ou des ressources sont représentées par des graphes [CUI 01]. Le premier, le *graphe des dépendances fonctionnelles*, utilise les nœuds pour représenter les composants logiciels, un composant de l'intergiciel ou un module du système et les arcs pour indiquer qu'une entité a besoin des services d'une autre pour fonctionner. Ainsi la Figure 8 indique que la tâche C1 a besoin des composants de l'intergiciel C3 et de C4 pour s'exécuter. Le second graphe représente les exigences estimées de ressources de chaque composant en un *graphe des dépend*ances *vis-à-vis des ressources*. Ces exigences peuvent être exprimées, par exemple, par le pourcentage d'occupation de l'unité centrale ou par le débit utilisé par un composant ou une tâche (cf. Figure 9 p.64).

Enfin notons l'utilisation d'un *Super-Graphe des configurations* qui permet de définir tous les ensembles de configurations aux différents niveaux hiérarchiques et en particulier les différentes implantations possibles.

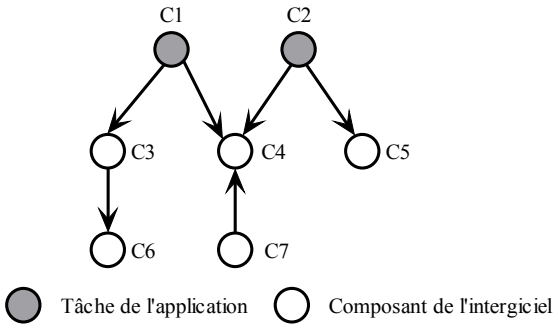


Figure 8 : Graphe de dépendance fonctionnelle [CUI 01]

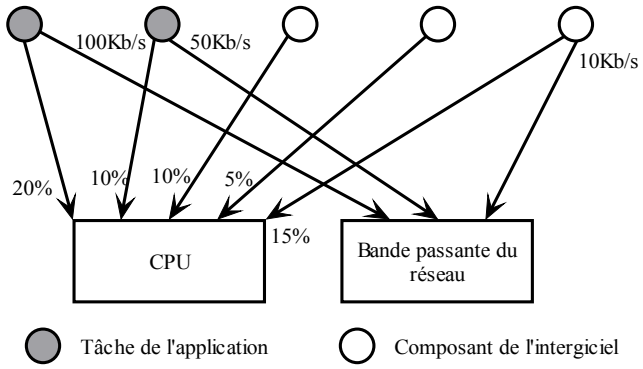


Figure 9 : Graphe des dépendances vis-à-vis des ressources [CUI 01]

1.3.2.2. Grammaire

La représentation de l'application par un graphe permet de représenter les configurations à l'aide d'une grammaire formelle. Cette représentation mêle alors les deux niveaux structurels de l'application puisque les symboles non terminaux représentent les groupes de composants composites, les composants composites et d'autres symboles temporaires alors que les symboles terminaux représentent les composants atomiques. Une configuration de l'application est alors représentée par un mot de la grammaire

L'intérêt de cette grammaire appelée ConfigG est de vérifier la logique des spécifications de qualité de service à l'aide de deux analyseurs — parseurs. Le premier est syntaxique et vérifie que la configuration correspond bien à un mot du langage et qu'elle est donc licite. Le second est sémantique et vérifie les différentes étapes de dérivation c'est-à-dire la compatibilité entre les composants.

Ainsi Agilos propose un exemple représentatif de la modélisation des applications multimédias en tant qu'applications dominées par les flux d'informations. Il propose un formalisme qui permet de représenter différents niveaux de l'application allant des contraintes fonctionnelles à l'échange d'information entre ces constituants. De plus, l'utilisation de graphes sous-entend celle de grammaires formelles qui servent de support à des vérifications de validité. Ces principes serviront de base à notre modélisation des applications multimédias réparties même si nous laissons de côté la dépendance vis-à-vis des ressources.

1.3.3. Structures des systèmes multimédias

Nous nous intéressons maintenant à la structure des systèmes multimédias qui mettent en œuvre les différentes méthodes de gestion de la qualité de service que nous avons présentées. Ces systèmes doivent permettre à la qualité de service d'être configurable, prédictible et maintenable tout au long du cycle de vie du système. Dans cet objectif, leur structure doit respecter différents principes [AUR 98] parmi lesquels nous retenons tout particulièrement :

- la transparence qui impose que l'application ne soit pas en charge de la spécification et de la gestion de la qualité de service et qui peut être obtenue par l'utilisation d'un intergiciel ;
- l'intégration qui énonce que la qualité de service doit être gérée dans toutes les couches du système ;
- la séparation des aspects (cf. §1.2.1 p.45) qui conseille d'implanter sur des éléments architecturaux séparés les activités distinctes que sont le transfert des médias, le contrôle et la gestion.

De nombreuses solutions sont proposées à partir de plates-formes logicielles parmi lesquelles nous avons retenu les architectures suivantes.

a. CAliF Multimédia

Au sein de la plate-forme CAliF Multimédia évoquée précédemment (cf. §1.2.2.1 p.49), la gestion de la qualité de service est structurée en deux éléments principaux : une base de données de qualité de service et un gestionnaire (cf. Figure 10). Rappelons tout d'abord que son objectif est de permettre aux applications de réserver les ressources du réseau tout en adaptant leurs besoins aux ressources disponibles. La base de données reflète l'état global du système grâce à un annuaire des requêtes de qualité de service, à une carte du réseau permettant de connaître les réservations effectives sur l'ensemble des routeurs connus, à un serveur d'authentification tenant à jour la liste des permissions et les quotas pour les réservations et enfin à un annuaire de règles qui permet de définir la politique d'adaptation. D'autre part, le gestionnaire de qualité de service reçoit les requêtes de qualité de service. Il est constitué de trois modules réalisant trois fonctions principales :

- le module de traitement des requêtes choisit la politique de qualité et en informe le module d'adaptation ;
- le module d'adaptation négocie les besoins d'une application si les ressources du réseau viennent à changer et surveille l'état du réseau ;
- le module d'application de la politique de qualité de service réalise le paramétrage des mécanismes de qualité de service sur les routeurs en fonction de règles définies avant l'exécution.

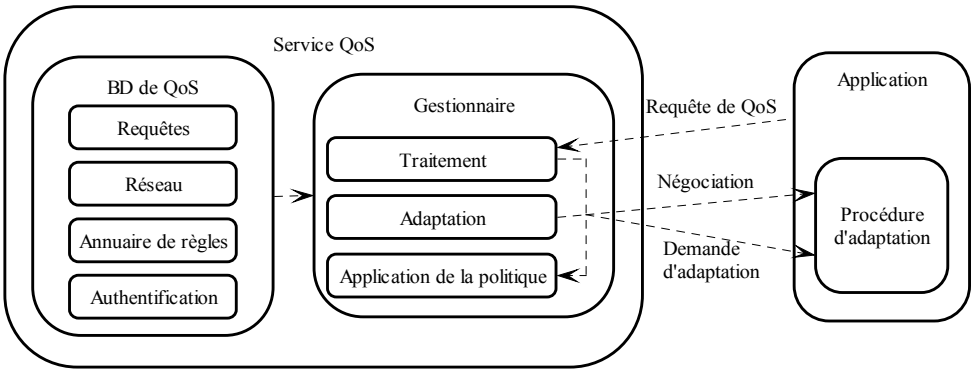


Figure 10 : Architecture de gestion de la QoS dans CAliF Multimédia [GAC 01b]

Nous retiendrons en particulier la nécessité d'utiliser une base de données reflétant l'état du système rendant ainsi celui-ci réflexif.

b. JQoS

L'architecture de gestion de la qualité de service du système JQoS [ZHU 01] (cf. §1.2.3.2 p.52) pour les vidéoconférences via Internet basées sur la qualité de service est constituée de trois éléments principaux (cf. Figure 11). Tout d'abord, une source établit une session de vidéoconférence en envoyant des données via des canaux R.T.P. audio et vidéo séparés. Les participants à la vidéoconférence correspondent à des récepteurs qui s'abonnent aux canaux pour participer à la session et envoient des rapports pour indiquer l'état ou qualité de réception et les requêtes de qualité de service. Enfin, le *Gestionnaire de session* surveille les performances de réception, gère chaque état de récepteur, administre les requêtes de qualité de service et décide de l'adaptation de la qualité de service dont il informe la source.

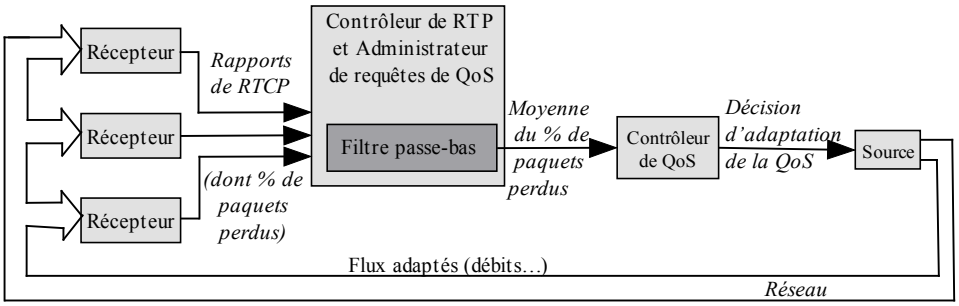


Figure 11 : Gestion de la qualité de service dans JQoS d'après [ZHU 01]

Le principe retenu pour la gestion de la qualité de service peut être représenté par un système asservi d'automatique linéaire (cf. Figure 12). Un récepteur envoie au gestionnaire de session une requête de qualité de service qui constitue la consigne en automatique. Ce dernier récupère l'état de réception du récepteur de manière à déterminer si la requête peut être acceptée définissant ainsi la variable de retour. En effet, la qualité de service ne peut pas être augmentée dans le cas où un récepteur subit beaucoup de pertes de transmission. Lorsque le nombre de requêtes similaires est suffisant, le gestionnaire peut prendre une décision d'adaptation de la qualité de service en fonction de l'état courant du système et émet ainsi la commande du système asservi. Si le service est adapté, les flux multimédias sont adaptés et les états des récepteurs modifiés. La boucle du système asservi est fermée. Cette architecture permet donc une auto-adaptation dynamique de la qualité de service.

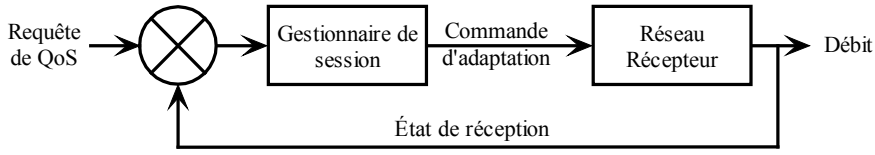


Figure 12 : Principe de gestion de la qualité de service dans JQoS

Le *Gestionnaire de session* sert ainsi de passerelle entre les récepteurs et la source. Il est composé d'un *Contrôleur de R.T.P.* qui suit les performances de transmission et gère chaque état de récepteur, d'un *Administrateur de Requêtes de qualité de service* traitant ces

requêtes de dégradation ou d'augmentation de la qualité de service et d'un *Contrôleur de qualité de service* prenant la décision d'adaptation de la qualité de service en fonction des informations fournies par les deux entités précédentes et l'envoyant à la source.

Cette adaptation des flux au contexte ne répond cependant pas au besoin de personnalisation des services. Ce système permet de fournir une qualité de service du même type que celles obtenues dans les réseaux. Il ne permet pas d'adapter l'intégralité du service à l'utilisateur en tant qu'élément du contexte d'exécution. Cependant, il propose un mécanisme intéressant de contrôle par rétro-action assurant la dynamicité de l'adaptation et la convergence de la qualité de service vers une qualité optimale pour un contexte donné. Cette méthode permet ainsi de fournir une garantie de service non pas déterministe mais que nous pourrions appeler "meilleur effort optimisé".

c. Agilos

La gestion de la qualité de service implantée sur l'intergiciel Agilos [LIB 01] (cf. §1.2.3.3 p.54) est structurée en trois niveaux (cf. Figure 13). Le premier niveau est réalisé par des composants *Adaptateurs* et des composants *Observateurs* qui gèrent le système distribué en surveillant le gestionnaire de ressources et les applications. Ils déclenchent des événements signifiant un changement de ressource et provoquent ainsi l'adaptation à ces variations en fournissant des informations qui ne sont pas spécifiques à une application mais spécifiques aux types de ressources — bande passante, CPU etc. (cf. Figure 9 p.64). Le deuxième niveau utilise ces mêmes informations pour que des *Configurateurs de composants* décident quand et comment doivent être invoqués les mécanismes d'adaptation à l'aide de règles d'adaptation spécifiques à l'application. Enfin, le troisième niveau est constitué par un *Configurateur de service* qui contrôle et coordonne les décisions d'adaptation entraînant des changements topologiques dans la fourniture des services multimédias — clients, filtres, etc. L'adaptation de l'application aux variations des ressources est alors obtenue par l'utilisation d'une boucle de contrôle de retour dans laquelle un algorithme de correction P.I.D. dédié [LIB 99] permet de régler la qualité de service.

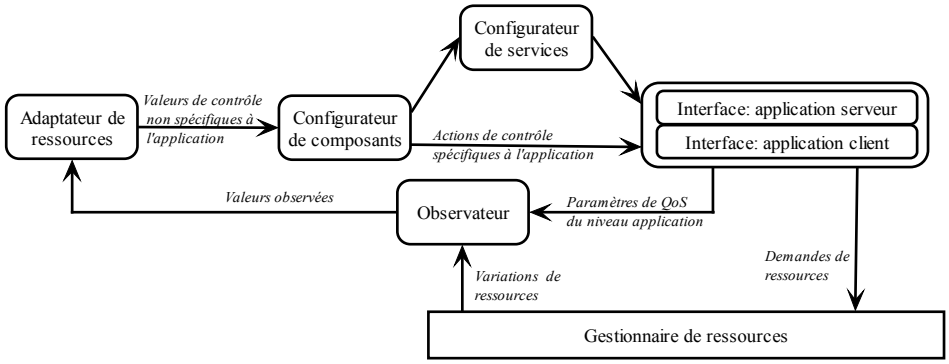


Figure 13 : Gestion de la qualité de service dans Agilos d'après [LIB 99]

La modélisation de l'adaptation de l'application utilise deux modèles :

- un modèle de contrôle par tâche — *Task Control Model* — qui vérifie que les décisions d'adaptation permettent de satisfaire les propriétés recherchées ;

- un modèle de contrôle flou — *Fuzzy Control Model* — qui transcrit à l'aide de règles les décisions d'adaptation du niveau système en choix d'adaptation par reconfiguration ou paramétrage.

Chaque modèle est réifié par un composant : l'Adaptateur qui adapte et configure les tâches pour le contrôle par tâche et le Configurateur pour le contrôle flou. L'utilisation de la logique floue permet d'étendre le modèle linéaire de base de l'adaptation fourni par le contrôle par tâche à des choix non linéaires. En d'autres termes, la stratégie d'adaptation est une approximation des comportements non linéaires du système grâce à un contrôle linéaire par morceaux en temps normal — *Task Control Model* —et un contrôle flou aux points de discontinuité — *Fuzzy Control Model*.

Cette architecture est originale par l'algorithme qui permet de prévoir la qualité de service d'une potentielle configuration en fonction des ressources disponibles. A l'aide de tests, il définit ainsi les paramètres de l'application sur lesquels il faut agir — les paramètres réglables — pour obtenir la qualité de service désirée décrite par des paramètres critiques. Remarquons que ce problème est non polynomial et que l'algorithme propose cependant une méthode permettant de relier un paramètre critique à un nombre fini de paramètres réglables à l'aide de composants de test appelés Q*ualprobe.*

Cependant les auteurs eux-mêmes ont souligné le principal inconvénient d'Agilos : la spécification statique de l'adaptation. C'est pourquoi ils ont enrichi cet intergiciel dédié à l'adaptation [LIB 02] à l'aide de l'intergiciel reconfigurable $2K^Q$ [WIC 01] qui gère la qualité de service grâce à des réservations de ressources et de l'intergiciel évolutif SMART [CUI 01b]. Nous nous sommes volontairement focalisés sur la partie adaptation, Agilos, mais il est intéressant de noter que la synergie des trois projets permet une mise à jour de la représentation réflexive de l'application grâce à la propagation des informations de changement uniquement sur quelques sites hôtes pris au hasard grâce à un algorithme original de propagation aléatoire [LIB 02]. De plus, la gestion de la qualité de service est répartie et utilise des messages XML structurés et portables. Enfin, des mécanismes d'intelligence artificielle permettent de prévoir les préférences des utilisateurs.

L'architecture de ce système complet comprend les éléments [GUX 01b] représentés sur la Figure 14 page 69:

- le *Contrôleur de QoS* qui réalise différentes fonctions de gestion de la qualité de service par ordonnancement, reconfiguration ou adaptation, et qui invoque les fonctions de gestion de ressources ;
- le *Gestionnaire de ressource* ;
- le *Gestionnaire d'événements* qui garde en mémoire les informations de contexte, déclenche les actions du *Contrôleur de QoS* et dirige la gestion de la qualité de service ;
- l'analyseur ou *Parseur de QoS* qui traduit les politiques spécifiques aux applications en structures de données exploitables par le *Gestionnaire d'événements* ;
- le *Profiler* qui définit automatiquement les préférences de l'utilisateur.

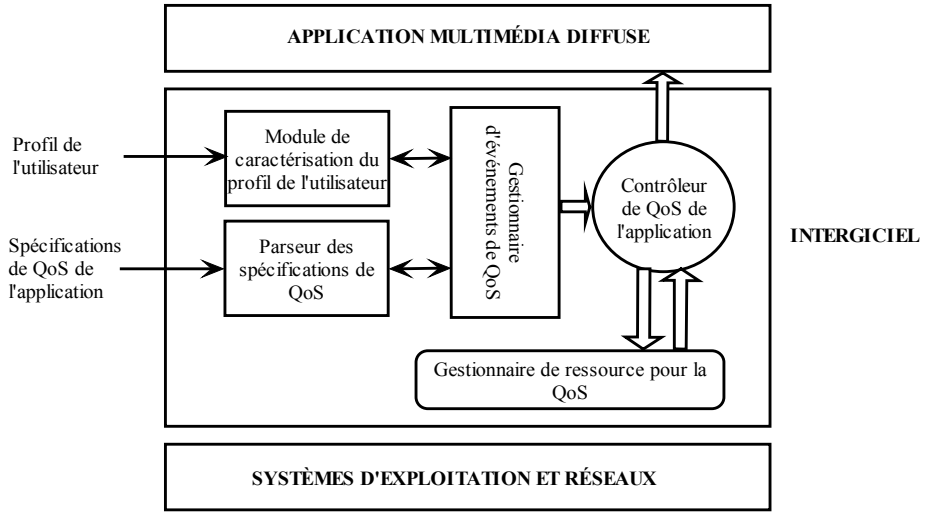


Figure 14 : Architecture exploitant Agilos, 2K^{Q} et SMART [GUX 01b]

Le système complet souligne l'importance des événements pour gérer les applications réparties possédant des contraintes temporelles et l'intérêt de la prévision des qualités de service potentielles. De plus, il propose un système réparti de gestion et surtout un mécanisme original de synchronisation des données représentant l'application.

1.3.4. Synthèse

Les applications multimédias réparties sur Internet sont caractérisées par de grands débits de données devant respecter des contraintes temporelles et ceci en l'absence de toute possibilité de réservation de ressources. La qualité de service peut donc être obtenue uniquement par l'adaptation de l'application. Or, il n'est possible de prévoir ni les besoins de ces applications, ni l'évolution du réseau et donc du contexte. Ce caractère imprédictible impose alors à l'adaptation d'être dynamique. D'autre part, les applications multimédias sont des applications guidées par les flux où la perception que l'utilisateur a de ces flux est primordiale. C'est pourquoi nous pensons qu'il est essentiel de considérer non seulement les caractéristiques objectives de qualité de service dépendant principalement de l'état du réseau mais encore les caractéristiques subjectives évaluées par l'utilisateur.

L'étude des systèmes existants pour gérer la qualité de service dans les applications multimédias, que nous résumons dans le Tableau 2, nous amène à penser qu'il est nécessaire de définir une gestion dynamique dans ces deux étapes, spécification et exécution, qui tienne compte explicitement de la qualité obtenue par l'utilisateur. Or parmi les travaux présentés aucun ne propose une spécification dynamique en l'absence de réservation de ressources pour une qualité de ce type là. L'objectif de cette thèse est donc de combler ce manque en proposant un modèle de gestion de la qualité de service au sens large utilisant l'adaptation dynamique des applications.

Projet	Applications		QoS		Adaptation				Réservation de ressources
	modèle	*techno-logie*	type	*modèle*	entité adaptée	spécification	*exécution*	*mode*	
CAliF	*client/ serveur*	*Corba*	réseau		**application**/ routeur	statique	*dynamique*	*reconfiguration de l'application*	oui
QuO	*graphe*	*Corba*	réseau	*langage*	**application**	statique (**dynamique** Qoskets)	*dynamique*	*paramétrage des objets*	oui
POLKA	*graphe*	*Corba*	réseau	*langage*	**application**	**dynamique**	*dynamique*	*ordonnancement*	**NON**
JQoS	*client/ serveur*	*JMF*	réseau		sources des flux multimédias	**dynamique**	*dynamique*	*modification des formats de compression*	**NON**
Agilos	*graphe*	*C++*	**réseau**/ **utilisateur**	*graphe*	**application**	statique (Agilos seul)	*dynamique*	*paramétrage des composants reconfiguration*	**NON**

Tableau 2 : Comparaison de différents projets de gestion de la qualité de service dans les applications multimédias réparties

Pour mener à bien ce projet, nous avons noté que les approches modulaires, en particulier à base de composants logiciels, sont de plus en plus souvent utilisées. Leur modélisation utilise alors des graphes qui mettent en exergue les flux multimédias échangés entre les composants des applications réparties. Lorsque les applications évoluent en fonction du contexte, les grammaires liées à ces graphes permettent de vérifier la validité des configurations envisagées.

Des systèmes gérant la qualité de service dans les applications multimédias réparties, nous retiendrons également les étapes utiles à l'adaptation dynamique des applications. Celle-ci débute par une phase de perception du contexte afin de détecter les variations qui déclencheront une adaptation. Puis, le système de gestion de la qualité de service doit savoir prévoir les différentes qualités de service qu'il est possible d'obtenir en fonction du contexte. Il est alors important d'avoir la possibilité d'agir à différents niveaux structurels sur l'application en fonction des effets escomptés sur le service rendu à l'utilisateur.

Enfin, l'une des principales difficultés soulevées par les travaux que nous avons présentés est la complexité du choix de la configuration de l'application correspondant à un contexte donné. Or ce choix repose sur l'optimisation et donc l'évaluation de la qualité de service que les différentes configurations peuvent fournir. Cette évaluation doit être réalisée dans un contexte contraint temporellement. C'est pourquoi nous allons étudier l'optimisation et l'évaluation dans les systèmes complexes répartis.

1.4. Optimisation et répartition

Une gestion efficace de la qualité de service nécessite une étape d'évaluation qui permette d'en estimer les résultats et éventuellement de réajuster les choix effectués. Dans les applications multimédias réparties, cette évaluation doit être réalisée en cours d'exécution. Elle est donc soumise à des contraintes temporelles à l'instar des applications qu'elle évalue. Or l'optimisation et l'évaluation dans les systèmes complexes répartis présentent une problématique très proche. C'est pourquoi nous présentons tout d'abord quelques aspects de programmation parallèle car ces systèmes l'utilisent fréquemment. Puis nous exposons une méthode d'évaluation des temps de traitement dans les systèmes répartis qui inspirera notre méthode.

Enfin, nous avons noté les problèmes de complexité rencontrés dans la gestion de la qualité de service dans les applications multimédias. En particulier, nous avons vu dans CAlif Multimédia (cf. §1.2.2.1 p.49) que le nombre de situations contextuelles possibles est infini et dans Agilos (cf. §1.3.3.c p.67) que la définition du lien entre un paramètre de qualité de service de haut niveau conceptuel de type utilisateur et un paramètre de plus bas niveau de type réseau est un problème non polynomial. De plus, l'évaluation dans les systèmes complexes répartis se heurte également à des problèmes de complexité algorithmique. C'est pourquoi nous définirons cette notion avant de présenter quelques solutions utilisées pour des problématiques proches de la nôtre.

1.4.1. Flots de données

Un modèle d'exécution par flots de données «consiste à traiter les opérations au fur et à mesure que leurs arguments deviennent calculés » [BER 80]. Il permet l'optimisation du temps de calcul dans la programmation parallèle car l'asynchronisme de son exécution permet l'explicitation du parallélisme inhérent au programme [BER 80] [VER 00] et la synchronisation implicite des activités parallèles [LEE 94]. De plus, il est naturellement auto-ordonnancé — *self-scheduling* [LEE 94]. Cependant seule l'assignation unique des variables garantit le déterminisme final du programme en n'autorisant le calcul d'une variable qu'une seule fois dans la vie du programme [PLA 76] [BER 80].

La modélisation de ce type d'applications est dérivée des réseaux de Pétri. Elle est dominée par les graphes de flots de données : ce sont des graphes orientés comme dans [COR 99] [VER 00] dont les nœuds représentent des instructions ou des traitements dénués d'effet de bord et dont les arcs expriment les dépendances des données entre ces instructions et, en particulier, l'ordre de réalisation des traitements. Lors de l'exécution, les données se propagent le long des arcs sous la forme de jetons. Lorsqu'un nœud dispose d'un jeton à chacune de ses entrées, il est exécuté et produit d'autres jetons. A ces graphes sont souvent ajoutés des graphes de précédence ou graphes des tâches comme dans [GAL 99] : ils décrivent le flot de contrôle c'est-à-dire que les nœuds représentent les tâches et les arcs les contraintes de précédence entre les tâches [SYR 77]. L'objectif de la représentation des programmes sous ces formes est l'optimisation de la rapidité d'exécution. Leur principal inconvénient est qu'une exécution séquentielle y est lente [AMA 95].

Deux modèles primitifs ou historiques de machines supportent ces applications. Le premier est constitué par les machines statiques qui exploitent le principe d'assignation unique et de flux de données. Elles sont apparues vers 1975 avec le *MIT Static Dataflow Machine* [DEN 75]. Le second regroupe les machines dynamiques à jetons contextuels apparues à la fin des années 1970 avec Arvind au MIT et Gurd et Watson à l'Université de Manchester. Elles permettent d'exécuter plusieurs fois et simultanément le même bloc d'instructions en ajoutant un identificateur de contexte au jeton pour distinguer les jetons relatifs aux mêmes opérations mais à des données contextuellement différentes. Le modèle statique présente l'avantage de la simplicité de la détection des nœuds disponibles grâce à des bits de présence mais est pénalisé par les constructions itératives ou réentrantes qui provoquent des baisses de performance. En revanche, le modèle dynamique se distingue par de meilleures performances obtenues en attribuant de multiples jetons à un arc. Cependant il est difficile à mettre en œuvre.

Afin d'optimiser son temps de réalisation, nous avons choisi d'utiliser le modèle d'exécution à flots de données pour l'évaluation dynamique de la qualité de service. Pour sa réalisation, nous nous inspirerons du modèle des machines dynamiques car, pour un contexte donné, il sera nécessaire d'évaluer simultanément et rapidement différentes qualités de service possibles. Nous n'utiliserons pas des jetons mais des événements pour identifier les traitements à effectuer car ils nous semblent plus efficaces pour améliorer le parallélisme dans la mesure où ils peuvent être directement pris en compte par les langages de programmation récents.

D'autre part, l'estimation des qualités de service que l'application pourrait fournir en fonction de sa configuration fera appel à la détermination préalable de plusieurs caractéristiques dont le temps d'exécution si besoin. Or le calcul du temps d'exécution et son optimisation constituent une problématique classique de l'informatique embarquée temps réel. Nous allons donc présenter maintenant un type de graphe des flux utilisé dans ce contexte pour y puiser les idées qui nous permettront de concevoir un graphe des flux adapté à notre problématique.

1.4.2. Optimisation du temps d'exécution dans les systèmes embarqués

Nous présentons ici le Graphe Conditionnel des Processus — *Conditional Process Graph* — proposé dans [ELE 00] pour optimiser les temps d'exécution lors de la synthèse des systèmes embarqués distribués composés de processeurs programmables — *C.P.U* soit. *Central Processing Unit,* microprocesseur, microcontrôleur — et de processeurs dédiés à l'application — A.S.I.C. soit *Application Specific Integrated Circuit*. Ce graphe abstrait capture au niveau processus le flot de données et le flot de contrôle et évalue le temps le plus long que met un système à s'exécuter. Il permet de minimiser ce temps en ordonnançant les processus de façon déterminée et en optimisant les paramètres de communication. Les systèmes embarqués sont ici définis comme des processus interagissant qui ont été répartis sur une architecture composée de plusieurs processeurs programmables et de plusieurs composants matériels dédiés, interconnectés par des bus partagés.

Nous voyons immédiatement les préoccupations communes entre cette problématique et la nôtre. Il s'agit d'optimiser un critère de qualité de service, le temps d'exécution. Les

systèmes sont répartis sous forme de processus, hôtes de machines hétérogènes. La représentation utilisée pourra donc être adaptée à la qualité de service dans les applications multimédias réparties

Le Graphe Conditionnel des Processus est défini à partir du Graphe des Processus que nous allons présenter tout d'abord.

1.4.2.1. Graphe des Processus

Le Graphe des Processus est une représentation abstraite d'une application spécifiée comme un ensemble de processus [ELE 98]. C'est un graphe orienté, acyclique et polaire $G(V, E_s, E_c)$ où :

- chaque nœud $P_i \in V$ représente un processus dédié à un élément de traitement *i.e.* processus, composant logiciel ou bus ;
- E est l'ensemble des arêtes où une arête $e_{ij} \in E$ de P_i à P_j indique que la sortie de P_i est l'entrée de P_j et E_s et E_c sont les ensembles des arêtes simples et des arêtes conditionnelles avec $E_s \cap E_C = \varnothing$, $E_s \cup E_C = E$.

A une arête conditionnelle est associée une condition c'est-à-dire que la transmission sur cette arête est réalisée seulement si la condition est vraie alors que dans le cas d'une arête simple, il suffit que le processus d'entrée soit activé. Le Graphe des Processus est un graphe polaire car deux nœuds factices appelés source et puits représentent la première et la dernière tâche. Ces nœuds sont factices car leur temps d'exécution est nul et ils n'utilisent pas de ressources. Tous les nœuds du graphe sont successeurs de la source et prédécesseurs du puits. Enfin, chaque traitement est caractérisé par un temps d'exécution ce qui permet l'évaluation du temps total à partir du Graphe Conditionnel des Processus.

1.4.2.2. Graphe Conditionnel des Processus

a. Construction

Le Graphe Conditionnel des Processus, *Conditional Process Graph*, est le nom donné par [ELE 00] au graphe élaboré des processus, *mapped process graph*, $\Gamma(V^*, E^*_s, E^*_c, M)$. Il est généré à partir du Graphe des Processus $G(V, E_s, E_c)$ en insérant des processus additionnels, représentant des processus de communication, sur certaines arêtes et en associant chaque processus à un dispositif de traitement. Cet adjonction permet de représenter l'implantation en prenant en compte la spécificité des applications réparties via des réseaux.

Ce graphe permet de représenter l'association des processus $P_i \in V^*$ à des processeurs et des bus grâce à la fonction :

$M : V^* \rightarrow PE$ avec PE ensemble des dispositifs de traitement

$P_i \rightarrow M(P_i)$ avec $M(P_i)$ dispositif de traitement auquel P_i est affecté pour l'exécution

PE, ensemble des éléments de traitement, regroupe les processeurs programmables, les composants matériels dédiés et les bus alloués.

Puis chaque processus P_i, alloué à un processeur programmable ou matériel $M(P_i)$, est caractérisé par un temps d'exécution t_{Pi}. Les processus de communication sont introduits pour chaque connexion qui relie des processus alloués à des processeurs différents : leur temps d'exécution $t_{i,j}$, où P_i est l'émetteur et P_j le récepteur, correspond au temps de communication.

Enfin, un nœud de disjonction est défini comme un nœud ayant des arêtes conditionnelles en sortie. Le processus associé est alors appelé processus de disjonction. De même, on définit un nœud de conjonction et un processus de conjonction comme un nœud et le processus associé sur lesquels se retrouvent des chemins disjoints issus de nœuds de disjonction.

On obtient ainsi le Graphe Conditionnel des Processus dont un exemple est donné par la Figure 15.

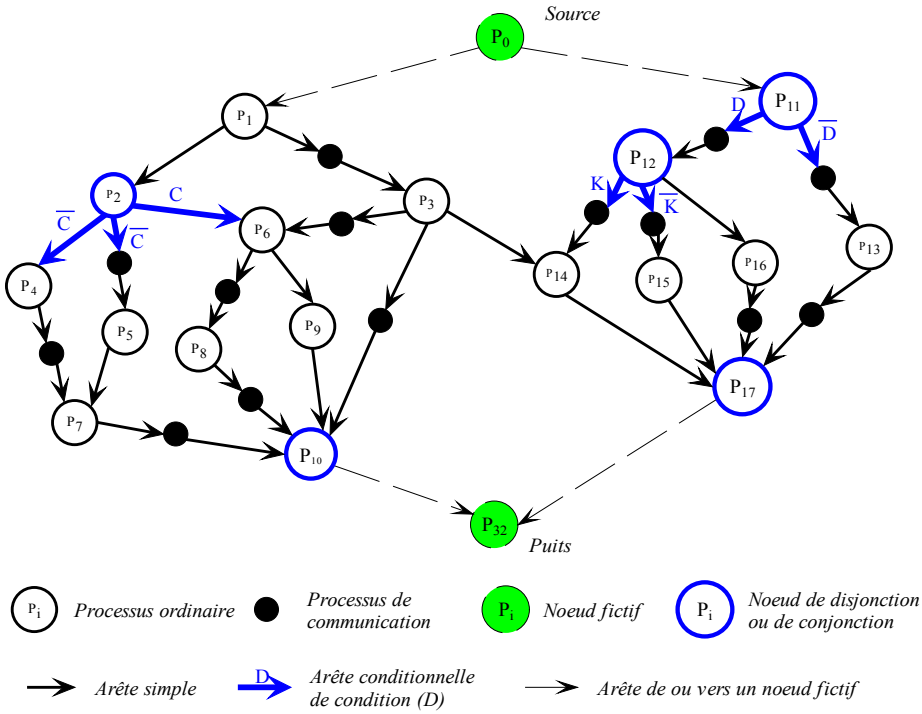


Figure 15 : Exemple de Graphe Conditionnel des Processus d'après [ELE 00]

b. Utilisation

Un processus, s'il n'est pas de conjonction, est activé uniquement si toutes ses entrées sont arrivées. Un processus de conjonction est activé après l'arrivée de messages sur l'un de ses chemins entrants. Un processus fournit ses sorties quand il a terminé ses traitements.

On définit une garde comme l'expression booléenne X_{Pi} associée à chaque nœud du graphe qui représente la condition nécessaire et suffisante pour que ce processus soit activé comme dans $\left(X_{P3} = true,\ X_{P14} = D \wedge K\right)$. Ces gardes permettent d'interpréter la

signification des arcs par la condition suivante : deux nœuds P_i et P_j — P_j n'étant pas un nœud de conjonction — sont reliés par une arête e_{ij} si et seulement si $X_{Pj} \Rightarrow X_{Pi}$ ce qui signifie que X_{Pi} est vrai quand X_{Pj} l'est. La sémantique d'exécution utilisée ici est appelée *single rate system* — système à débit unique — car chaque nœud est exécuté au moins une fois pour chaque activation du système.

Lorsque le système s'exécute, un des chemins possibles dans le graphe est exécuté. Chaque alternative correspond à un sous graphe $G_k \in \Gamma$ auquel est associée une expression logique appelée le label du chemin et notée L_k qui représente les conditions nécessaires à l'exécution de ce sous-graphe. Le sous graphe G_k contient les nœuds P_j du graphe conditionnel Γ pour lesquels $L_k \Rightarrow X_{Pj}$ où X_{Pj} est la garde du nœud P_j et elle doit être vraie quand L_k est vrai.

c. Intérêt de ces graphes pour les applications multimédias réparties

La problématique étudiée par [ELE 00] concerne, comme la nôtre, le traitement de flux d'informations, multimédias dans notre cas, et les systèmes distribués hétérogènes. Nous avons les mêmes préoccupations d'optimisation et de contraintes temporelles. Elles sont de type "temps réel dur" pour ces auteurs et de type "temps réel souple" pour nous (cf. §1.3.1.1 p.59).

La représentation qu'ils proposent présente différentes caractéristiques qui nous seront utiles. Tout d'abord, elle permet l'identification des composants communs aux configurations, c'est-à-dire aux évaluations ou aux résultats d'évaluation communs à différentes configurations, à l'aide des nœuds de conjonction et de disjonction. Elle permet d'obtenir les graphes des configurations par extraction depuis un graphe de l'application. De plus, elle représente non seulement la structure de l'entité, mais aussi son exécution. Enfin, en ajoutant des informations de localisation comme attributs des nœuds, il sera possible d'identifier les différentes possibilités de déploiement pour chaque composant.

D'autre part, le Graphe Conditionnel des Processus est utilisé pour construire un algorithme basé sur les heuristiques d'ordonnancement par listes qui constituent une réponse à la complexité algorithmique exponentielle à laquelle se heurte le problème d'optimisation du temps de traitement par ordonnancement. Or nous avons noté précédemment que ce n'est pas la seule problématique liée aux applications réparties à base de composants ou de processus qui soit d'une complexité exponentielle. C'est pourquoi nous proposons dans la suite un aperçu de cette problématique.

1.4.3. Complexité algorithmique

La méthode d'évaluation et d'optimisation du temps de traitement présentée précédemment soulève le problème de la complexité des choix dans des systèmes construits à base de briques atomiques que l'on peut déplacer et ordonnancer différemment en fonction des ressources. Ce problème est NP complet dans le cas général. Nous allons préciser ici ce que cela signifie et quelles en sont les conséquences pour notre problématique.

1.4.3.1. Définition

La fonction de complexité algorithmique mesure « les ressources utilisées par un algorithme » qui sont « caractérisées par la taille de l'instance du problème à traiter » [WOL 91]. Nous ferons l'hypothèse que « la frontière entre complexité acceptable et inacceptable se situe à la limite entre fonction polynomiale et non polynomiale. » [WOL 91]. Nous choisirons comme critère de complexité la complexité en temps notée O et nous utiliserons une analyse du pire cas où nous considérerons que le temps de calcul pour des données de taille n est le temps de calcul maximum pour des données de cette taille. Ainsi nous utiliserons la définition suivante et le critère suivant :

« Une fonction g(n) est dite O(f(n)) s'il existe des constantes c et n_0 telles que pour tout $n>n_0$ $g(n) \leq cf(n)$. »

« Nous considérerons comme acceptable toute fonction de complexité polynomiale, c'est-à-dire $O(n^k)$ pour un k fixé ». [WOL 91].

Les problèmes acceptables sont donc les problèmes représentés par un langage appartenant à la classe d'équivalence polynomiale P — classe des langages décidés par une machine de Turing polynomiale. On définit également les problèmes NP appartenant à la classe NP— classe des problèmes dont les langages sont acceptés par une machine de Turing non déterministe polynomiale. Enfin on définit la classe NPC des problèmes NP complets comme la classe des problèmes de NP décrits par un langage L tel que pour tout langage L' de NP il existe une transformation polynomiale permettant de passer de L' à L. On admet généralement que P≠NP ce qui implique qu'un problème NP-complet n'a pas de solution polynomiale.

Notre objectif est d'identifier les problèmes de décision ou d'optimisation susceptibles d'être NP complets parmi ceux auxquels nous allons être confrontés lorsque nous voudrons optimiser la qualité de service dans les applications multimédias réparties. En effet, dans ce cas, il faudra proposer des heuristiques permettant de réduire la complexité au cas polynomial.

Rappelons notre problématique : nous souhaitons proposer un système de gestion de la qualité de service qui optimise celle-ci dans le cadre d'applications multimédias réparties constituées de composants logiciels. L'application étant répartie, ces composants pourront être placés à différents endroits et, s'ils respectent les contraintes fonctionnelles de leur service, ils pourront être ordonnancés de différentes manières. Enfin, pour fournir différentes qualités et s'adapter au contexte, un composant devra être choisi pour chaque fonction atomique au sens de [GUX 01] (cf. §1.3.2.1 p.62). Il est donc nécessaire de distinguer parmi ces problématiques celles qui ont une complexité exponentielle.

1.4.3.2. Complexité algorithmique de cas simples

a. Choix de composants en fonction de la qualité de service

L'adaptation d'une application aux variations du contexte d'exécution peut être réalisée grâce au choix des composants les plus adéquats parmi ceux disponibles. Il s'agit alors de choisir k composants parmi un ensemble de n si l'on suppose disposer de plusieurs composants pour réaliser une fonctionnalité donnée, avec des caractéristiques de qualité de

service différentes. Dans ce cas, le nombre de sous-ensembles possibles de composants est donné par $C_k^n = \frac{n!}{k!(n-k)!}$.

Dans le pire des cas, la complexité peut donc être non polynomiale. En revanche, ces composants devront respecter des contraintes liées aux fonctionnalités et aux compatibilités ce qui élimine naturellement de nombreuses possibilités. De plus, il s'agira de déterminer l'assemblage qui fournira la meilleure qualité de service dans un contexte donné. Grâce à ces contraintes supplémentaires, on peut donc espérer qu'en pratique, la complexité sera plus faible.

Cependant si l'on adopte la modélisation des applications multimédias réparties à base de graphes (cf. §1.3.2.1 p.62), choisir les composants à utiliser en fonction de la qualité de service revient à choisir des chemins dans un graphe. Il s'agit donc de résoudre un problème de routage sur la qualité de service — *QoS Routing* — dont l'objectif est de définir un chemin entre un nœud source et un nœud puits qui satisfasse un ensemble de contraintes de QoS. Ce problème est appelé *Multi-Constrained Paths Problem* — MCP. Dans le cas d'une diffusion multicast, il s'agit de MCMWM — *Multiple Constrained Minimum Weight Multicast*. Ces problèmes ont été montrés NP complets par [WAN 96] dans le cas unicast pour un nombre de métriques supérieur à 2 et dans le cas multicast par [KUI 02] pour m métriques additives avec m quelconque. Le cas multicast est particulièrement intéressant car il correspond à fournir différents services puisqu'il y a différentes destinations dans le graphe représentant une application.

Ces démonstrations soulignent que choisir des composants est donc un problème de complexité non polynomiale dès que la qualité de service fait appel à plus de deux critères. Cependant il existe des solutions comme l'algorithme *MAMCRA* [KUI 02] qui définit de manière polynomiale un routage permettant d'optimiser la qualité de service. La qualité de service d'un flux et celle désirée y sont d'ailleurs modélisées à l'aide de vecteurs regroupant les différentes métriques d'une façon similaire à la représentation utilisée dans QoSTalk (cf. §1.1.2.2.b p.39). Cependant notons qu'aucune considération ergonomique n'est utilisée dans cet algorithme si bien que l'utilisateur peut être perturbé par les variations brutales de qualité.

b. Ordonnancement et répartition de composants sur des ressources hétérogènes

Une fois le choix des composants effectué, le problème de les ordonnancer et de les répartir sur des ressources hétérogènes est NP-complet. Ceci a été démontré dans [GAR 79] sous l'appellation d'Ordonnancement Multiprocesseur Minimum avec Facteur de Vitesse — *Minimum Multiprocessor Scheduling with Speed Factor*. Si la qualité de service est liée au temps de traitement, il ne sera pas possible d'optimiser ce critère de façon algorithmique sans proposer une heuristique. De plus, les contraintes de précédence que les composants d'une application multimédia doivent respecter du fait de leurs fonctionnalités ne suffisent pas à réduire la complexité du problème.

En effet, trouver un ordonnancement de composants en respectant des contraintes de précédence demeure NP complet. Ceci a été démontré dans [GAR 79] sous les termes d'Ordonnancement Minimum avec Contraintes de Précédence — *Minimum Precedence Constrained Scheduling*. Si l'on tient compte des contraintes de ressources au lieu des contraintes de précédence, le problème demeure NP complet comme prouvé par le même

auteur pour un Ordonnancement Minimum avec Contraintes de Ressources — *Minimum Resource Constrained Scheduling*.

Dans les cas où les deux types de contraintes sont réunis c'est-à-dire si l'on doit ordonnancer une application composée de composants de durée unitaire sur des processeurs identiques tout en leur faisant respecter un graphe de précédence, l'optimisation du temps de traitement est un problème appelé *UET-Scheduling* [RAY 87] qui a été démontré comme NP complet [FER 89] [LAG 90] dans le cas général. En particulier, sa complexité est non polynomiale dans le cas général s'il est possible de faire varier le graphe de précédence. En revanche, si ce graphe est un arbre, le problème devient polynomial.

1.4.4. Synthèse

L'optimisation de la qualité de service dans une application multimédia répartie à base de composants possède donc des points communs avec l'optimisation des systèmes répartis complexes. Tout d'abord, l'aspect réparti permet d'introduire un parallélisme opératoire dans la gestion de la qualité de service. Ainsi les temps de réaction aux variations du contexte seront optimisés à l'aide d'une programmation à base de flux de données. Nous aurons alors une application orientée flux de données par sa nature multimédia et un support de gestion de cette application orienté flux de données pour accroître sa réactivité. Nous pourrons donc établir un isomorphisme entre l'application et la gestion de la qualité de service. D'autre part, cet isomorphisme nous amène à puiser dans les représentations des systèmes répartis pour modéliser l'application et les choix de qualités possibles pour un contexte donné. Le graphe orienté devient alors incontournable non seulement pour représenter l'application (cf. §1.3.2.1 p.62) mais aussi pour représenter l'évaluation de l'application. Enfin, nous veillerons à proposer des solutions algorithmiques réalisables c'est-à-dire ayant une complexité polynomiale pour un problème que nous soupçonnons être NP complet, ce qu'il nous faudra démontrer.

1.5. Conclusion

Enrichissant la définition de la qualité de service utilisée dans les réseaux de télécommunication, nous considérons que la qualité de service est l'adéquation entre le service fourni et le service souhaité par l'utilisateur. Son modèle devra donc prendre en compte des critères matériels, mesurables et objectifs, et d'autres, non mesurables et subjectifs, en particulier ceux liés aux préférences des utilisateurs. Ainsi nous proposerons une réelle personnalisation de l'application. Ce modèle devra être hiérarchisé d'une part en fonction de la structure de l'application et d'autre part en fonction des services fournis par celle-ci. A la manière des études sur les politiques de coût de la qualité de service dans les réseaux, nous utiliserons la notion d'utilité pour guider notre méthode de définition de la qualité de service à partir des résultats de l'évaluation des différents critères. Le modèle de qualité de service et le modèle d'évaluation seront alors tous les deux centrés sur l'utilisateur.

D'autre part, l'Internet nous interdit des réservations de ressources ce qui nous conduit à faire le choix de l'adaptation. Or celle-ci ne peut concerner le réseau car il ne propose aucune véritable garantie de service. Elle ne peut pas non plus s'appliquer aux systèmes d'exploitation à cause de la répartition des applications sur des ressources hétérogènes imprédictibles. Nous choisissons donc d'adapter le plus complètement possible l'application en modifiant sa structure, sa composition ou son ordonnancement lorsque cela sera possible. De plus, la nécessité de modifier l'application nous encourage à utiliser des composants logiciels et matériels et ouvrira donc la porte à la technologie émergente des COTS. Ainsi, une restructuration de l'application consistera en une modification des composants utilisés et donc une adaptation du comportement et de l'ordonnancement de l'application.

Cette adaptation des applications ne peut cependant être réalisée qu'en respectant les contraintes des applications multimédias réparties et en particulier le débit des données et les exigences temporelles. De plus, sont imprédictibles aussi bien le comportement de la majorité des applications multimédias que celui d'Internet ou que le contexte général d'exécution. L'adaptation de l'application devra donc être la plus dynamique possible : nous définirons les politiques d'adaptation en cours d'exécution en fonction du contexte et des composants disponibles. Nous aurons alors une vision large du contexte en correspondance avec notre définition de la qualité de service : il recouvre le contexte informatique, le contexte matériel, l'environnement d'exécution et le contexte humain. Il sera donc nécessaire de posséder à tout instant un reflet de l'application répartie construite à base de composants.

L'adaptation repose sur une évaluation que nous voulons répartie de manière à optimiser les temps de calculs et à respecter les contraintes temporelles du multimédia. Pour cela, inspirés par la programmation par flots de donnés, nous proposerons l'utilisation de graphes. L'application multimédia et l'évaluation de sa qualité de service seront alors représentées par des graphes isomorphes. Cherchant à optimiser la qualité de service fournie aux utilisateurs à tout instant, nous veillerons à proposer des solutions dont la complexité algorithmique soit polynomiale pour que nos modèles puissent être mis en œuvre.

Partie 2: Modélisation de la qualité de service et des applications multimédias réparties

Au centre de notre étude se situe la qualité de service. Nous proposons une définition adaptée aux applications multimédias réparties et à leurs caractéristiques. Celles-ci se distinguent non seulement par l'importance de ce que perçoit l'utilisateur mais également par l'imprévisibilité du contexte. Notre détermination de la qualité de service permet d'englober ces aspects et ne se restreint donc pas aux seules caractéristiques des réseaux. A partir de cette définition, pierre angulaire de nos travaux, nous construisons notre modèle d'application. Ce modèle contient déjà les germes de notre méthode d'adaptation de l'application. Ses différents niveaux structurels sont utilisés pour adapter l'application. Puis, nous proposons un modèle d'évaluation de la qualité de service sous la forme de règles de composition des différents paramètres généraux de qualité de service. Enfin, nous présentons une méthode de conception mettant œuvre tous ces modèles et permettant d'exécuter l'application sur notre plate-forme. Dans tous ces travaux, notre réflexion est guidée par l'analogie précédemment citée entre la qualité de service et l'utilité.

2.1. Modélisation de la qualité de service

Les spécificités des applications multimédias nous amènent à proposer un modèle original de qualité de service dont nous précisons ici la définition, la structure hiérarchisée et la représentation. Nous présentons ensuite la méthode d'évaluation que nous mettons en œuvre pour ce modèle inspiré de la notion d'utilité.

2.1.1. Définition de la qualité de service

Les applications multimédias sont caractérisées par une forte interaction entre le logiciel et l'utilisateur. Ces systèmes orientés flux de données fournissent des informations en continu et leur qualité est liée à leur capacité à rendre ce service sous la forme attendue par l'utilisateur.

Nous définissons donc ***la qualité de service — QdS —*** *comme* ***l'adéquation entre le service souhaité par l'utilisateur et le service qui lui est fourni.***

Cette définition présente l'intérêt d'englober aussi bien la qualité de service telle qu'elle est définie dans les réseaux que différents aspects d'ergonomie et de personnalisation des services.

Définir cette qualité pour une application donnée consiste donc à spécifier ce que souhaite l'utilisateur, puis le service qui lui est fourni à un instant donné afin de comparer ces deux observations. La qualité de service est habituellement décomposée en critères. Ceux-ci ne sont donc pertinents qu'à condition qu'il soit possible de comparer l'état d'un critère pour le service fourni avec celui de ce même critère pour le service attendu. Or l'utilisateur ne peut pas décrire la QdS à l'aide d'une infinité de paramètres. Le nombre de ceux qu'il faut considérer devra donc être suffisamment faible pour que l'utilisateur puisse les définir et suffisamment élevé pour être représentatif. Ces différents critères sont structurés au sein de notre modèle de QdS défini, lui aussi, en fonction des caractéristiques des applications multimédias réparties et en particulier de leur contexte d'exécution.

2.1.2. Modèle de la qualité de service

Pour illustrer la présentation de notre modèle et de l'ensemble de nos travaux, nous utilisons un exemple d'application multimédia que nous présentons en préambule. Puis, nous exposons notre modèle de qualité de service structuré en deux niveaux avant d'exposer la représentation bidimentionnelle qui nous permet de proposer une méthode d'évaluation.

2.1.2.1. Exemple d'application multimédia répartie sur Internet

Nous avons choisi d'illustrer nos travaux par un type d'application multimédia répartie possédant à la fois les caractéristiques d'une application multimédia de présentation et d'une application conversationnelle [HAF 98] : la vidéoconférence sur Internet. Nous entendons par **vidéoconférence** — *Videoconferencing on Internet Protocol*, *VCoIP* — un système de communication interactif pouvant transmettre en simultané des images, du son et des données dans le but de réunir à distance et en temps réel un groupe de personnes situées dans des lieux distincts. Cette vidéoconférence est caractérisée par deux types d'utilisateurs. Le premier concerne les intervenants à la conférence que nous nommerons **locuteurs**. Le second décrit les auditeurs de cette conférence que nous appellerons **téléspectateurs.**

En fonction des prévisions du concepteur de l'application, les locuteurs peuvent utiliser différentes langues et transmettre des données sous forme de fichiers. Ils peuvent également se déplacer devant un tableau ou un écran de projection pour y désigner des informations. Parallèlement, les téléspectateurs peuvent recevoir les images des locuteurs et percevoir les discours soit directement grâce à la transmission du son soit par le biais de sous-titres.

La Figure 16 page 85 présente une réalisation possible d'une telle application dans le cas simple de la transmission synchronisée du son et de l'image lors d'une conférence donnée par deux locuteurs situés en deux endroits différents pour deux téléspectateurs distants. Ce schéma précise les localisations que nous appellerons **sites**, 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 pour les locuteurs. Chacun dispose d'un ordinateur personnel 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 considéré comme le poste P_3 et équipé d'un casque audio. Le second S_4 est constitué par l'ordinateur personnel multimédia P_4 équipé d'un haut-parleur. Nous supposons que tous les postes sont équipés d'une carte que nous appelons carte vidéo et qui est en fait 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.

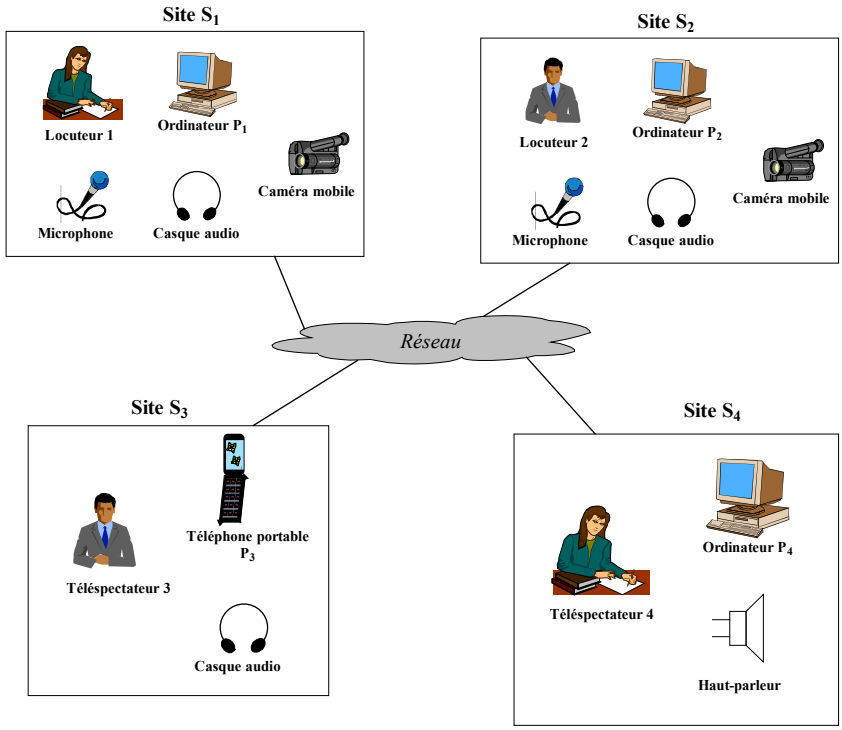


Figure 16 : Exemple de vidéoconférence

2.1.2.2. Structure du modèle

Nous construisons notre modèle de qualité de service à partir de deux niveaux hiérarchiques. Ainsi la qualité est décrite à l'aide de caractéristiques que nous regroupons en critères en fonction de leur dépendance vis-à-vis du contexte.

Une **caractéristique de QdS** est un élément descriptif simple de la qualité. Elle peut être liée à l'état du réseau ou du matériel mais elle peut également décrire des aspects plus ergonomiques. Elle peut être mesurable ou non mesurable. En revanche, elle est caractérisable soit à partir de mesures soit à partir des spécifications des composants de l'application. Ainsi dans la vidéoconférence, différentes caractéristiques définissent le son, en particulier le débit et le temps de transmission, qui sont liés à l'état du réseau. Le flux sonore peut également être caractérisé par la langue utilisée pour les conversations. Pour décrire les images, on peut utiliser la cadence vidéo — nombre d'images par seconde — ou le débit du flux qui dépend des composants de compression et de l'état du réseau. Les images sont également décrites par leur taille en pixel.

D'autre part, la propriété principale du contexte d'exécution des applications multimédias réparties est d'être imprévisible (cf. §1.2.1.2.b p.46). Ainsi 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 classement. 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 ou un assemblage de composants 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. Ce critère permet également de décrire l'adéquation du support matériel — comme les dimensions d'un écran — ou bien l'adéquation des performances d'un traitement — comme la vitesse d'un suivi de mouvement.

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 lié 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 amenées par l'environnement matériel telles le bruit, les erreurs de transmission, ou la luminosité lorsqu'il s'agit de détecter et suivre un objet dans une image. C'est sur ce second critère que se répercuteront les propriétés du réseau que sont le débit, les retards ainsi que leurs variations respectives. 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 qu'un assemblage de composants ou qu'une application. Elles seront définies à partir des souhaits de l'utilisateur. Les caractéristiques contextuelles et intrinsèques seront donc 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 [HAF 98] mais en partant du service et en adaptant l'application et non les ressources. Il est donc intéressant de confronter notre modèle à ceux proposés par d'autres auteurs.

2.1.2.3. Intérêt de la structure

a. Hiérarchie

Nous proposons une hiérarchisation du modèle de QdS à la manière de D.G. Firesmith [FIR 03] (cf. §1.1.2.1.b p.35). Notre QdS correspond à ce que cet auteur nomme "Groupe de facteur de qualité orienté utilisateur" (cf. Tableau 3 p.87). Nos caractéristiques de QdS regroupent à la fois les sous-facteurs, les critères et les mesures car nous nous attachons uniquement aux résultats qui peuvent être comparés aux demandes des utilisateurs. En revanche, cet auteur ne propose pas de notion équivalente à celle de nos critères intrinsèques et contextuels. En effet, ceux-ci sont directement hérités de l'importance d'un contexte non prédictible dans les applications multimédias réparties sur Internet et ne sont pas pertinents pour toutes les applications. De plus, Firesmith définit les critères en tant que description spécifique d'un sous-facteur dans un domaine particulier. Or nous n'étudions qu'un domaine, les applications multimédias réparties, donc nos sous-facteurs, ce que nous nommons caractéristiques, sont déjà spécifiques et il est inutile de discerner les deux

notions. Restreindre l'étude aux applications multimédias réparties permet donc de simplifier la hiérarchie du modèle de qualité de service.

Niveau	Définition	Exemples
Groupe de facteurs	Ensemble de facteurs	Orienté concepteur
		Orienté utilisateur
Facteur	Aspects de haut niveau d'un produit fini	Réutilisabilité
		Sécurité
		Performances
Sous-facteurs	Aspects de bas niveau	Délais
		Gigues
Critères	Description spécifique	Protection des données contre les erreurs de transmission
		Détection des erreurs de transmission
Mesures	Quantification des critères	Taux d'erreurs de transmission

Tableau 3 : Modèle de qualité de service par [FIR 03]

De plus, nous préférons utiliser les termes de critère en tant que « caractère, principe qui permet de distinguer une chose d'une autre, d'émettre un jugement, une estimation » [LAR 96] c'est-à-dire en tant que marque distinctive à la base de notre évaluation de la QdS et celui de caractéristiques « ce qui constitue la particularité, le caractère distinctif » [LAR 96]. Nous pensons que les caractéristiques — délai, format, etc. – permettent de définir les critères représentatifs, intrinsèque et contextuel, que nous utilisons pour l'évaluation de la QdS. Le terme de facteur « agent, élément qui concourt à un résultat » [LAR 96] nous semble moins adapté à notre problématique spécifique tandis qu'il est nécessaire dans le cas d'une définition très générale de la qualité de service telle celle donnée par [FIR 03].

b. Classement selon l'objectivité

Le classement des caractéristiques en fonction de leur méthode d'évaluation proposé par [HAF 98] permet de discerner des paramètres objectifs, c'est-à-dire mesurables et observables, et des paramètres subjectifs c'est-à-dire dépendant de l'équipement de l'utilisateur ou de son opinion et n'étant donc pas mesurables. Nous pensons qu'il ne peut y avoir de caractéristiques objectives puisque toutes les caractéristiques du service sont comparées aux désirs de l'utilisateur qui sont subjectifs. Il n'y a donc pas d'évaluation absolue mais relative à un utilisateur. Notre définition de la qualité de service utilise donc uniquement des paramètres subjectifs, mesurables ou non mesurables, qui sont observables par l'utilisateur.

Le Tableau 4 permet de préciser la relation entre notre classement intrinsèque/contextuel et le classement de [HAF 98]. Il précise si chaque critère utilise des caractéristiques mesurables et/ou des caractéristiques non mesurables pour décrire les principaux aspects du service. Ainsi, pour le critère intrinsèque, l'adéquation de la fonctionnalité avec les attentes des utilisateurs ne peut être mesurée. En revanche, les performances optimales du service sont mesurables quand elles concernent des

caractéristiques telles que la taille maximale des images ou le nombre maximal de couleurs. L'adéquation du support matériel, quant à elle, peut être décrite par des caractéristiques mesurables comme la taille d'un écran ou non mesurables comme sa mobilité. Le critère contextuel regroupe principalement des caractéristiques mesurables comme celles permettant d'évaluer le respect des contraintes temporelles. Cependant, le respect du service malgré les perturbations peut être décrit par des caractéristiques mesurables ou non mesurables comme la langue utilisée dans un dialogue.

Critère	Composantes du critère	Présence de	
		caractéristiques mesurables par nature	caractéristiques non mesurables par nature
Intrinsèque	Adéquation de la fonctionnalité		X
	Adéquation du support matériel	X	X
	Adéquation de ses performances optimales	X	
Contextuel	Respect des contraintes temporelles	X	
	Respect du service malgré les perturbations	X	X

Tableau 4 : Comparaison avec le classement proposé par [HAF 98]

D'autre part, nous ne définissons pas de QdS pour les composants logiciels car l'utilisateur perçoit uniquement les résultats d'un assemblage de composants réunis pour fournir une fonctionnalité et non ceux d'un composant isolé. En effet, dans le cas limite où une fonctionnalité est obtenue à partir d'un unique composant, la notion de QdS n'est pas attachée à l'entité composant mais à l'entité fonctionnalité. En d'autres termes, la QdS ne se définit pas à partir de l'implantation mais à partir du modèle fonctionnel. De plus, certaines caractéristiques sont mesurables par nature comme le nombre d'images par seconde alors que d'autres ne le sont pas comme l'adéquation avec la fonctionnalité attendue. C'est pourquoi il faudra proposer un système d'évaluation permettant de comparer les assemblages de composants entre eux et surtout de les confronter aux souhaits de l'utilisateur.

2.1.2.4. Représentation

Maintenant que sont déterminés les deux critères qui vont nous permettre de définir la QdS, il est important de choisir son mode de représentation. Dans notre modèle, la QdS est composée de deux critères, intrinsèque et contextuel, qui sont indépendants, *i.e.* orthogonaux, puisqu'une caractéristique de service dépend ou ne dépend pas du contexte d'exécution et participe à l'un ou l'autre des critères mais ne peut intervenir dans les deux. Il est intéressant de conserver les deux dimensions dans la représentation car, lors des comparaisons, les caractéristiques contextuelles peuvent varier en cours d'exécution alors que les caractéristiques intrinsèques sont fixées à la conception pour un composant ou pour une composition donnée d'un assemblage de composants : deux composants peuvent fournir la même qualité mais l'un étant à sa qualité intrinsèque optimale et l'autre étant limité par le contexte. Ceci peut être un argument de comparaison : si le contexte est défavorable, il sera ainsi plus porteur de choisir le second composant dont la qualité augmentera avec une amélioration du contexte. Nous utiliserons donc une première 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*)

Pour chaque dimension, intrinsèque ou contextuelle, la QdS traduit une proximité entre la qualité spécifiée par l'utilisateur considérée comme la qualité optimale car il n'est pas utile de faire mieux si l'utilisateur y est insensible. Inversement, elle traduit une distance par rapport à la pire qualité, celle définie comme rédhibitoire par l'utilisateur. 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 :

- par la note de 0 quand le service est rendu de manière rédhibitoire pour l'utilisateur ;
- par la note de 1 quand le service rendu correspond à celui attendu par l'utilisateur.

La QdS souhaitée par l'utilisateur est donc représentée par le couple (1,1). Comme les deux dimensions, intrinsèque et contextuelle, sont orthogonales, le plan de représentation de la QdS est lié à un repère orthogonal : il représente l'ensemble des qualités de service possibles. De plus, la QdS d'un composant ou d'un assemblage de composants dépend tout autant d'une dimension que de l'autre et exprime, à un instant donné, l'adéquation entre le service souhaité par l'utilisateur et le service qui lui est fourni. La QdS à un instant t_0 est donc représentée par un point du plan des qualités de service. Ce point a pour coordonnées les deux réels précédemment évoqués : Co pour le critère contextuel et In pour le critère intrinsèque comme représenté sur la Figure 17.

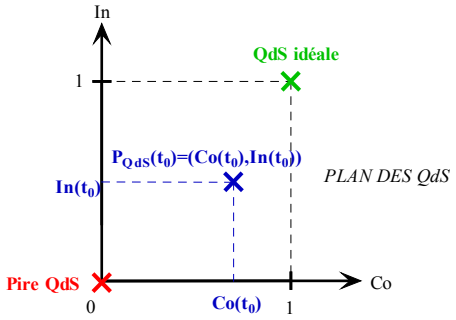


Figure 17 : Représentation de la QdS à un instant t

Un composant ou un assemblage de composants sera représenté selon les cas (cf. Figure 18 p.90) :

- par un point qui traduit la QdS à un instant précis ;
- par un ensemble de points 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.

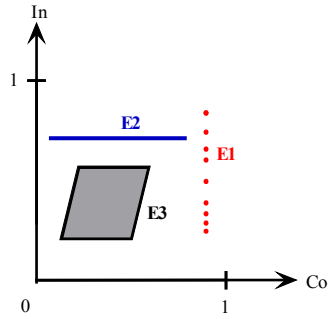


Figure 18 : Qualités de service de différentes entités

2.1.3. Évaluation de la qualité de service globale

Nous avons établi un premier mode de représentation simple de la QdS à partir de l'évaluation des deux types de caractéristiques, intrinsèques et contextuelles. Nous devons maintenant définir comment évaluer la QdS à partir des notes de ces deux critères. Nous cherchons donc une seconde représentation sous la forme d'une fonction *f* telle que : *QdS=f(Co, In)*. Nous allons tout d'abord identifier différentes solutions possibles. Puis nous déterminerons les propriétés mathématiques que la fonction recherchée doit respecter. Comme celles-ci ne suffisent pas à définir la fonction QdS de façon unique, nous utiliserons la notion d'utilité pour choisir comment évaluer la QdS à partir des deux critères de qualité.

2.1.3.1. Différentes solutions d'évaluation

Différents choix s'offrent à nous. Notre problème s'apparente à un problème de décision à deux critères pour lequel nous pourrions utiliser la théorie de la logique floue à l'instar de [LI 99] (cf. §1.3.3.c p.67). Cependant, dans le cadre de cette thèse, nous retiendrons uniquement les outils les plus simples.

Tout d'abord, la QdS peut être vue comme le meilleur compromis entre les caractéristiques contextuelles et les caractéristiques intrinsèques. Dans ce cas, la note de QdS est donnée par la distance euclidienne à la qualité optimale représentée par le point (1,1)

Formule 2 : $f_1(Co, In) = \sqrt{(Co-1)^2 + (In-1)^2}$.

Une autre solution consiste à définir la QdS comme égale à la note du pire critère. La seconde définition est donc

Formule 3 : $f_2(Co, In) = \min(Co, In)$ **.**

Nous allons vérifier que ces propositions donnent des résultats sensés. Pour cela, nous réalisons une étude empirique à partir des points définis par la Figure 19 de manière à couvrir toutes les zones de QdS possibles. Nous comparons les résultats de ces deux fonctions avec une définition intuitive de la QdS (cf. Tableau 5). L'élément de comparaison sera le classement obtenu des différentes qualités de service (cf. Tableau 6) car nous avons vu que la QdS n'est pas absolue mais qu'elle est au contraire une valeur relative aux désirs de l'utilisateur. Nous avons donc besoin uniquement d'ordonner les différentes qualités possibles pour pouvoir choisir la meilleure.

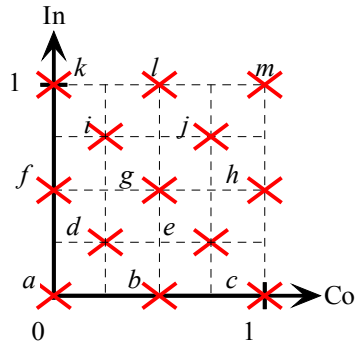


Figure 19 : Différentes qualités de service

Point		Critère		QdS		
nom	**coordonnées**	**contextuel**	**intrinsèque**	**intuitive**	f_1	f_2
a	(0 ;0)	rédhibitoire	rédhibitoire	rédhibitoire	0	0
b	(0,5 ;0)	moyen	rédhibitoire	rédhibitoire	0.5	0
c	(1;0)	optimal	rédhibitoire	rédhibitoire	1	0
d	(0,25 ;0,25)	plutôt mauvais	plutôt mauvais	plutôt mauvaise	0.35	0.25
e	(0,75 ;0,25)	plutôt bon	plutôt mauvais	moyenne	0.79	0.25
f	(0 ;0,5)	rédhibitoire	moyen	rédhibitoire	0.5	0
g	(0,5;0,5)	moyen	moyen	moyenne	0,71	0,5
h	(1 ;0,5)	optimal	moyen	plutôt bonne	1.12	0.5
i	(0,25 ;0,75)	plutôt mauvais	plutôt bon	moyenne	0.79	0.25
j	(0,75 ;0,75)	plutôt bon	plutôt bon	plutôt bonne	1,06	0,75
k	(0;1)	rédhibitoire	optimal	rédhibitoire	1	0
l	(0,5 ;1)	moyen	optimal	plutôt bonne	1.12	0.5
m	(1 ;1)	optimal	optimal	optimale	1.41	1

Tableau 5 : Évaluations de qualité de service

Evaluation	QdS		
	intuitive	f_1 - distance	f_2 - pire cas
optimale	m	c,h,j,k,l,m	m
plutôt bonne	h,j,l	e,g,i	j
moyenne	e,g,i	b,f	g,h,l
plutôt mauvaise	d	d	d,e,i
rédhibitoire	a,b,c,f,k	a	a,b,c,f,k

Tableau 6 : Comparaison des différentes méthodes d'évaluation

Aucune des deux propositions ne recoupe le classement obtenu par l'approche intuitive. En revanche, l'évaluation de la QdS par une distance euclidienne à la qualité optimale (f_1) donne des résultats très distincts pour les points b, c, f, k dont elle classe la QdS comme moyenne ou optimale alors que l'approche intuitive la considère rédhibitoire. Ce résultat est évidemment lié à la définition mathématique de la distance qui est une fonction continue et au fait que ces points correspondent à la note minimale pour l'un des critères. Les résultats obtenus par l'estimation du pire cas (f_2) sont semblables à ceux de l'approche intuitive. C'est donc cette définition qui serait la plus adaptée à notre problématique.

Cependant l'intuition ne constitue en aucun cas une justification c'est pourquoi nous allons définir les propriétés mathématiques que la fonction QdS doit respecter de manière à pouvoir guider notre choix.

2.1.3.2. 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 intrinsèques et contextuels.

Nous souhaitons que la QdS soit normée de manière à garder la même échelle de valeurs que les critères et ainsi simplifier les études mathématiques. 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 :

$$\textbf{P1}: \quad f:[0;1]\times[0;1]\rightarrow[0;1]$$

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 :

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

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 multimédias : 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 :

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

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 :

$$\textbf{P4 :} \qquad \forall x \in [0;1], f(x,1) = x \text{ et } \forall y \in [0;1], f(1,y) = y$$

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 :

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

L'ensemble des propriétés énoncées ici ne permet pas de définir la fonction f. Nous pouvons cependant noter que l'évaluation de la QdS par une distance à la qualité optimale (f_1) se distingue de l'approche intuitive et de l'estimation du pire cas (f_2) par les propriétés P1, P3, P4 c'est-à-dire que f_1 ne donne pas des valeurs entre 0 et 1, elle ne considère pas comme rédhibitoire une qualité où un des critères est rédhibitoire et elle considère comme optimale une qualité où l'un seul des critères est optimal. Nous sommes donc tentés de choisir la seconde définition — f_2 — pour la QdS non seulement pour sa similitude avec les résultats intuitifs mais également parce qu'elle respecte les propriétés mathématiques que nous avons déterminées. L'étude de l'analogie entre QdS et utilité devra nous confirmer dans ce choix.

2.1.3.3. Analogie avec l'utilité

L'utilité comme la QdS représentent « l'avantage ou la satisfaction qu'une personne retire de la consommation d'un bien ou d'un service » [PAR 92] (cf. §1.1.3.2 p.41). En économie, l'objectif de l'étude de l'utilité est de maximiser l'utilité totale pour un revenu donné mesurable et connu, alors que l'objectif de l'étude de la QdS est de maximiser celle-ci pour des ressources données non mesurables et inconnues. Nous proposons d'évaluer la QdS à partir des notes de deux critères intrinsèque et contextuel, In et Co. Il y a donc 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 7.

	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 7 : Analogie entre utilité et qualité de service

Cette analogie nous permet de définir des courbes d'indifférence pour la QdS (Figure 20) de la même manière que les courbes d'indifférence pour l'utilité (cf. §1.1.3.2 p.41).

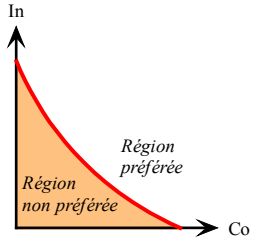


Figure 20 : Courbes d'indifférence pour la qualité de service

Or le degré de convexité d'une courbe d'indifférence 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. 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. 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 (cf. Figure 4 p.42). Une équipotentielle de la fonction *f* telle que $QoS=f(Co, In)$ est ainsi représentée sur la Figure 21.

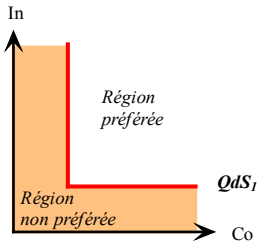


Figure 21 : Equipotentielle de qualité de service

La fonction qui permettra d'évaluer la QdS dans notre modèle devra donc posséder les équipotentielles que nous venons de définir en plus de toutes les propriétés établies au paragraphe précédent. Nous allons maintenant vérifier que l'estimation du pire cas (f_2) répond bien à tous ces critères.

2.1.3.4. Modèle d'évaluation de la qualité de service

L'étude précédente nous permet de définir l'évaluation de la QdS et de déterminer les propriétés de la comparaison de deux qualités de service à l'aide de notre modèle.

a. Définition de la fonction QdS

Nous avons établi que l'expression de la QdS en fonction des critères In et Co, *QoS=f*(*Co*, *In*), doit respecter les propriétés énoncées au paragraphe 2.1.3.2 p.92. Nous avons vu que la fonction $f_2(Co, In) = \min(Co, In)$ donnait des résultats cohérents (cf. §2.1.3.1 p.90). A partir de l'analogie avec la fonction d'utilité, nous avons ensuite défini les équipotentielles de la fonction f (cf. Figure 21). Or la fonction f_2 possède des courbes équipotentielles ayant la même allure :

$$\min(x, y) = A \in \Re \Rightarrow \begin{vmatrix} x = A \text{ et } y \geq A \\ \text{ou} \\ y = A \text{ et } x \geq A \end{vmatrix}.$$

Nous choisissons donc de définir la note de QdS d'une entité à partir des notes des critères intrinsèque, In, et contextuel, Co, par la relation :

Formule 4 : $\mathbf{QdS(Co, In) = \min(Co, In)}$

Le tracé des courbes de niveau de la QdS est alors celui de la Figure 22.

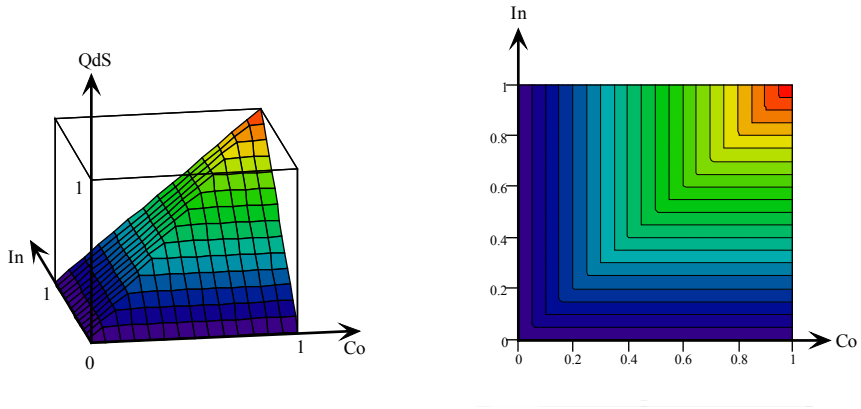


Figure 22 : Courbes de niveau de la fonction QdS

b. Propriétés de la fonction QdS : comparaison

La définition de la QdS à laquelle nous avons abouti nous permet de comparer la QdS de deux entités logicielles. Or la fonction obtenue simplifie cette comparaison en particulier dans le cas où les composants logiciels et les assemblages de ces composants ne proposeraient qu'une seule valeur du critère intrinsèque ce qui traduit le fait que les composants aient été conçus pour rendre le service d'une seule façon et donc avec une seule qualité.

En effet, supposons qu'une entité logicielle soit en cours d'exécution dans une configuration C_1 et fournisse une qualité de service QdS_1. Si tous les composants de cette configuration ne proposent qu'une seule QdS, cette configuration ne propose de par sa conception qu'une valeur In_1 du critère intrinsèque. La courbe parcourue par sa QdS lorsque le contexte varie est décrite par la Figure 23.

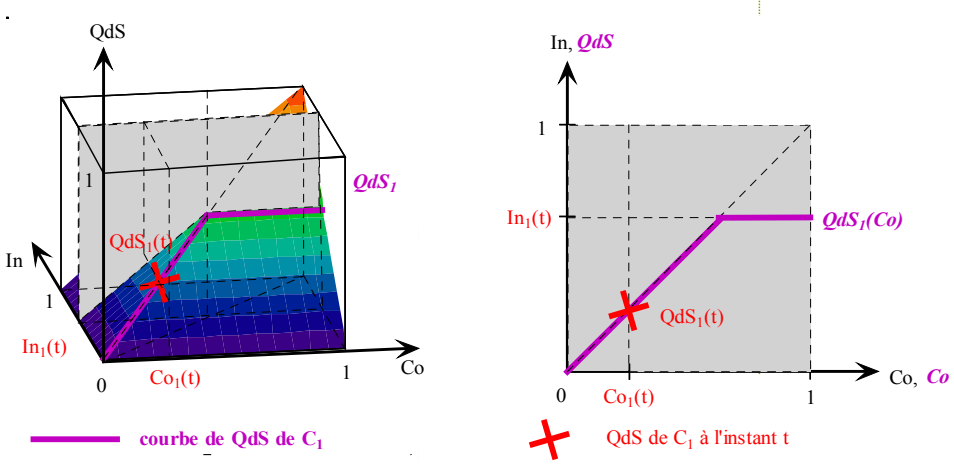


Figure 23 : Courbe de la QdS d'une entité à In constant

Supposons que l'entité étudiée soit mise en œuvre par 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 24a) 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 24b) 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.

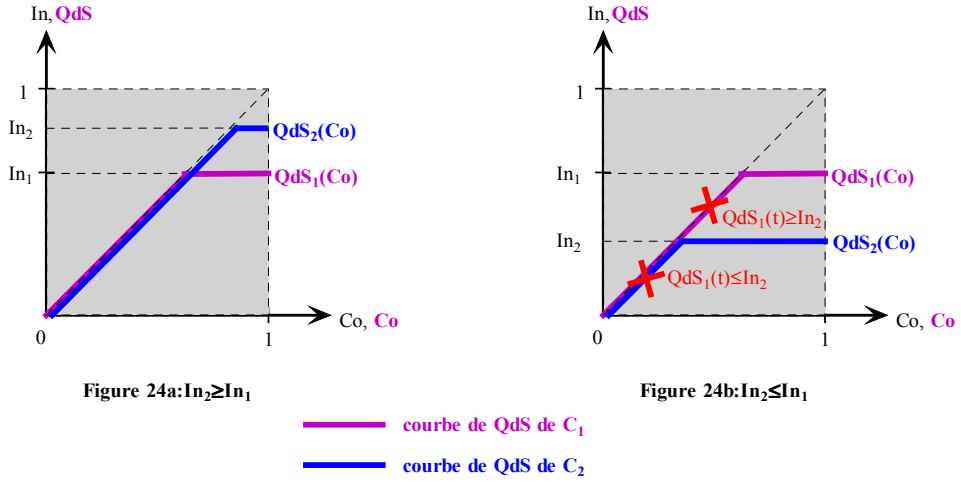


Figure 24a:$In_2 \geq In_1$ **Figure 24b:$In_2 \leq In_1$**

Figure 24 : Comparaison de deux configurations

2.1.4. Synthèse

Nous proposons un modèle original de la qualité de service des applications multimédias sur Internet. Cette qualité définie par l'adéquation entre le service souhaité par l'utilisateur et le service qui lui est fourni peut être modélisée de façon plus simple que dans le cas général des applications réparties car seules comptent les caractéristiques des logiciels que l'utilisateur peut directement percevoir dans une application multimédia. Ainsi nous n'utilisons que deux niveaux hiérarchiques. Le plus bas niveau permet de décrire des paramètres simples de QdS et est appelé caractéristique de QdS. Le second niveau regroupe dans deux critères de QdS d'une part les caractéristiques indépendantes du

contexte, c'est le critère intrinsèque, et d'autre part les caractéristiques dépendant du contexte, c'est le critère contextuel. De cette manière, notre modèle souligne l'importance du contexte et de ses variations dans les applications multimédias réparties sur Internet.

Pour représenter la QdS d'une entité, nous utilisons deux notes octroyées à chacun des critères. La QdS peut alors être représentée par le point du plan ayant ces deux valeurs pour coordonnées dans un repère orthonormé grâce à l'orthogonalité des deux critères et à la représentation des qualités extrêmes que nous avons choisie : 0 représente une qualité rédhibitoire pour l'utilisateur et 1 la qualité optimale c'est-à-dire celle désirée par l'utilisateur. Ainsi, nous conservons la richesse de représentation du modèle qui discerne l'influence du contexte. Puis l'analogie entre la QdS et la notion d'utilité en micro-économie, nous amène à choisir comme modèle d'évaluation de la QdS la fonction donnant la valeur du pire critère à cette qualité. Ce choix nous permettra de simplifier la comparaison entre deux configurations d'une même entité logicielle et simplifiera par là même la recherche d'une qualité et d'une configuration optimale. C'est notre modèle d'applications multimédias réparties, présenté dans le paragraphe à venir, qui fixe quels sont les assemblages de composants pour lesquels la QdS est définie.

2.2. Modélisation des applications multimédias réparties

Les spécificités des applications multimédias réparties sur Internet nous ont amenés à définir la qualité de service d'une manière originale, à en proposer un modèle hiérarchisé simple et un modèle d'évaluation spécifique. De la même façon, ce sont les caractéristiques de ces applications en matière de contraintes de développement logiciel sur Internet qui nous imposent de choisir l'utilisation d'une plate-forme comme support d'exécution de ces applications. Ce choix influe ensuite sur le modèle d'application que nous proposons, modèle qui reflète inévitablement le rôle prépondérant de ce que l'utilisateur peut percevoir et les contraintes temporelles du multimédia.

Après avoir justifié l'utilisation d'une plate-forme, nous allons définir le modèle d'application multimédia répartie que nous utilisons.

2.2.1. Nécessité d'une plate-forme

Les intergiciels, *middlewares,* plates-formes, sont très utilisés pour construire des systèmes à partir d'objets distribués — D.O.C. *Distributed Object Computing* [SCA 02]. En effet, comme nous l'avons déjà évoqué, ils autorisent la séparation des aspects [BOU 01] [ARC 00].

De plus, les applications ayant pour cible le grand public se doivent d'être bon marché et d'assurer un service satisfaisant. L'objectif de nos travaux est de garantir cette QdS à l'utilisateur. Pour que nos propositions soient utilisables, il faut qu'elles permettent de développer ce type d'application à faible coût. Pour cela, nous proposons d'utiliser des composants logiciels de manière à réduire le temps et le coût de conception. Ce temps sera également réduit si les architectures sont simples à construire et en particulier si le concepteur peut s'affranchir des contraintes autres que fonctionnelles.

Nous proposons donc d'utiliser une plate-forme logicielle qui sert de support d'exécution à l'application et qui permet d'optimiser la qualité du service fourni à l'utilisateur à tout instant en fonction des composants logiciels disponibles et du contexte d'exécution. Nous définissons une plate-forme d'exécution en tant que plate-forme matérielle munie de composants logiciels de base comme les protocoles réseaux et les noyaux des systèmes d'exploitation [SOL 97]. L'objectif de nos travaux est de proposer un modèle pour cette plate-forme d'exécution.

Notre plate-forme d'exécution sera conçue comme un superviseur réparti sur les différents postes de l'application. Elle reçoit des états reflétant le fonctionnement et la structure de l'application et envoie à celle-ci des commandes (cf. Figure 25 p.100) afin d'optimiser la QdS grâce à une adaptation dynamique de l'application réalisée en modifiant dynamiquement :

- le comportement des composants qui proposent différentes qualités de par leur conception ;

- la composition de l'application en supprimant, ajoutant ou remplaçant des composants ;
- l'ordonnancement des composants lorsque cela est possible ;
- le déploiement des composants.

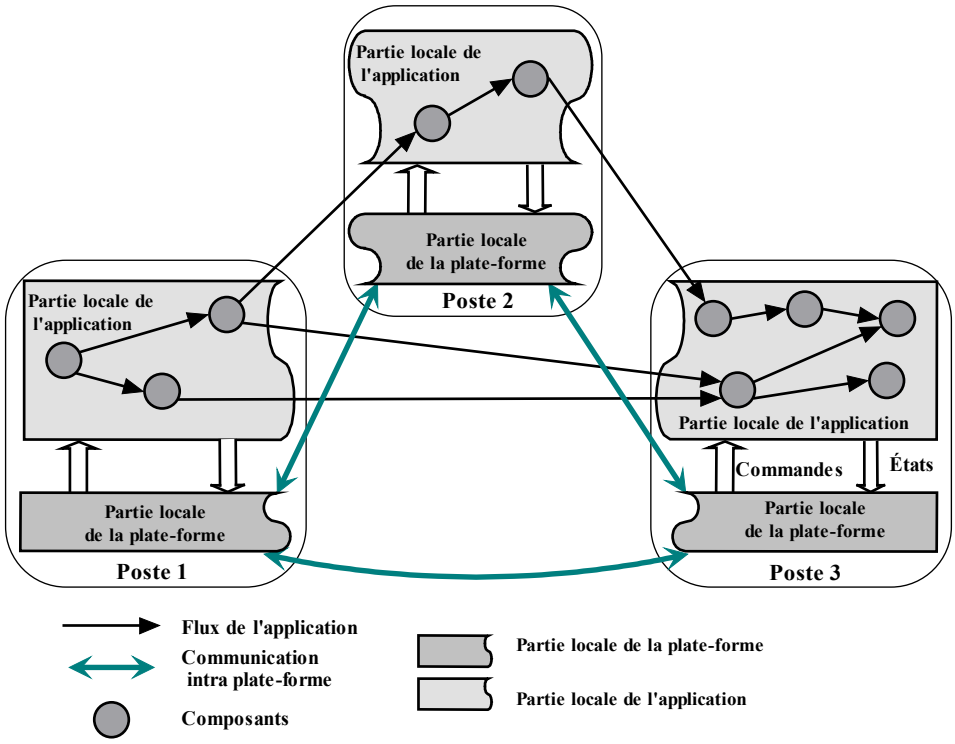


Figure 25 : Supervision de l'application par la plate-forme

La définition des politiques d'adaptation, quand et comment adapter, sera réalisée dynamiquement grâce aux informations contextuelles récoltées par la plate-forme qui décriront aussi bien les connexions réseau, que les terminaux informatiques ou que les préférences des utilisateurs. Lorsque la qualité fournie n'est pas satisfaisante soit parce que l'utilisateur n'a pas obtenu ce qu'il souhaitait soit parce que les ressources sont sous utilisées, la plate-forme évaluera les qualités de service que l'application peut potentiellement fournir pour savoir si une adaptation est nécessaire et déterminer laquelle. Notons que l'intervention de l'utilisateur à ce niveau est une des conséquences des caractéristiques des applications multimédias pour lesquelles le service fourni est directement perceptible. Une autre conséquence est la tolérance vis-à-vis de la perte d'information qui permet à la plate-forme d'adapter l'application autant que nécessaire et sur des critères locaux puisqu'il n'est pas utile de conserver les informations.

Pour réduire les coûts de développement des applications grand-public, l'une des solutions consiste à proposer une plate-forme d'aide à la conception. Nous pensons qu'il sera nécessaire de proposer un tel outil même si celui-ci ne fait pas partie des objectifs de cette thèse. C'est pourquoi la plate-forme d'exécution et les modèles associés seront conçus dans l'idée d'être utilisables par un outil d'aide à la conception dont nous définirons la méthode de conception à partir des différents modèles.

Cette plate-forme sera dédiée à notre problématique de la QdS des applications multimédias et au modèle d'application centré sur l'utilisateur que nous présentons dans la suite.

2.2.2. Groupes et Sous-Groupes

La structure d'une entité reflète la principale préoccupation de celui qui l'a conçue. Ainsi notre structure est le reflet de notre définition de la QdS. Celle-ci souligne l'importance de la perception de l'utilisateur. En effet, contrairement à d'autres applications comme le calcul scientifique, un élément du service fourni, une fonctionnalité, consiste dans le cas des applications multimédias à présenter quelque chose à l'utilisateur. Nous proposons donc de structurer les applications en fonction de ce que perçoit l'utilisateur puisqu'une fonctionnalité ne sera prise en compte pour l'évaluation de la QdS qu'à condition que l'utilisateur puisse l'observer et l'évaluer. Nous aboutissons alors aux deux niveaux structurels intermédiaires entre les composants et l'application proposés dans nos travaux précédents [LAP 02] : les Groupes et les Sous-Groupes.

2.2.2.1. Définition

Le **Groupe** représente le service rendu à un utilisateur. En le définissant, le concepteur spécifie les différents types d'utilisateur de l'application. Chaque Groupe constitue une classe. Chaque utilisateur bénéficie d'une instance du Groupe qui l'intéresse. En fonction du nombre et du type d'utilisateurs présents dans l'application à un instant donné, celle-ci contient des instances de Groupe différentes (cf. Figure 26).

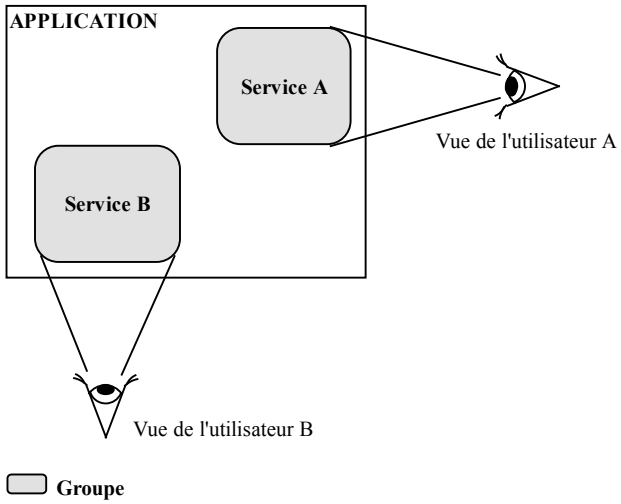


Figure 26 : Définition de la notion de Groupe

Ainsi pour la vidéoconférence présentée au paragraphe 2.1.2.1 p.84, deux Groupes différents décriront d'une part les intervenants de la conférence, les locuteurs, et d'autre part les téléspectateurs (cf. Figure 27).

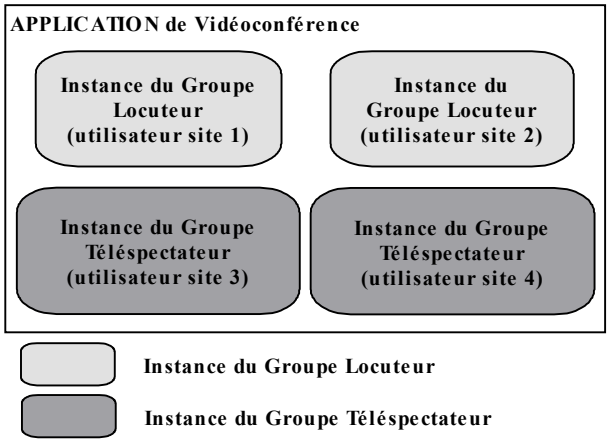


Figure 27 : Groupes dans une vidéoconférence

Notre modèle permet donc à un utilisateur de rejoindre ou de quitter l'application en cours d'exécution grâce à une instanciation ou une suppression de son instance de Groupe. Dans le cas particulier où un utilisateur dispose de plusieurs postes, un Groupe sera utilisé pour chaque poste. En effet, c'est vraisemblablement pour profiter de différents services que l'utilisateur utilise plusieurs postes. Ainsi, dans le cas d'une vidéosurveillance, une ou plusieurs personnes scrutent un ou plusieurs écrans selon le contexte.

Le **Sous-Groupe** représente une fonctionnalité, et une seule, du service rendu à un utilisateur. Un Groupe est donc composé de différents Sous-Groupes (cf. Figure 28). La composition d'une instance de Groupe en terme 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.

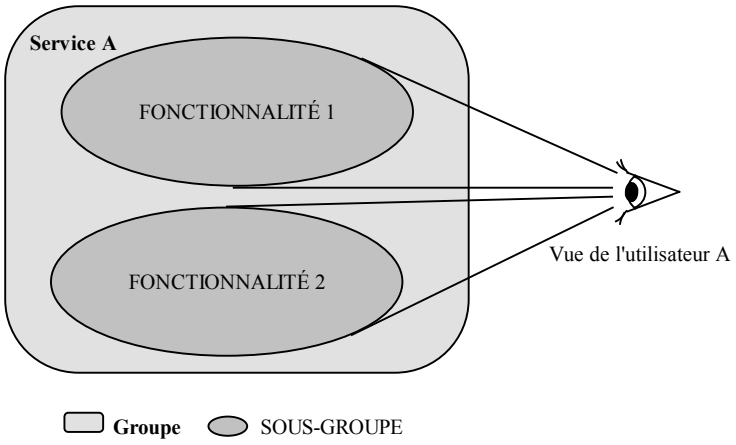


Figure 28 : Définition de la notion de Sous-Groupe

Ainsi pour la vidéoconférence, le Groupe téléspectateur contient deux Sous-Groupes, l'un fournissant les images et l'autre fournissant le son. Le Groupe Locuteur peut être composé uniquement des Sous-Groupes Son et Image ou fournir également une fonctionnalité de suivi automatique par la caméra permettant à l'orateur de se déplacer (cf. Figure 29).

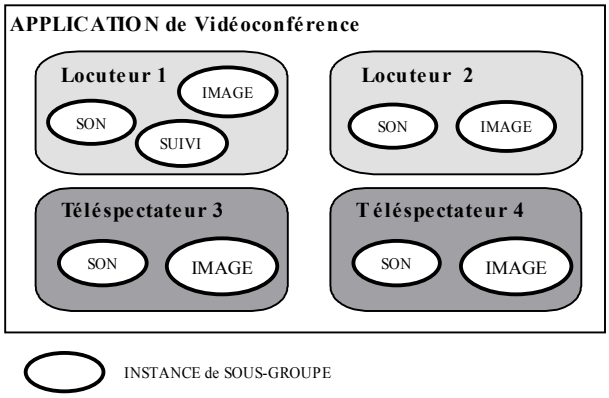


Figure 29 : Sous-Groupes dans une vidéoconférence

2.2.2.2. Propriétés

Du point de vue structurel, un Groupe est défini comme l'ensemble des composants rendant un service à un utilisateur. Il inclut donc toutes les fonctionnalités nécessaires au rendu de ce service. En particulier, il faut noter qu'il n'a besoin d'aucune autre information issue d'un autre service. Tels que nous les définissons, les instances de Groupes n'échangent pas d'information.

Cependant elles peuvent avoir des flux et des composants communs en particulier des composants tels que la caméra présente sur le site S_1 de la vidéoconférence. C'est le cas en particulier des composants matériels qui doivent être partagés entre différents utilisateurs et qui seront donc dupliqués dans la modélisation pour apparaître dans différents Groupes. Dans ce cas particulier, la caméra permet la transmission des images à tous les utilisateurs de l'application et appartiendra donc à toutes les instances de Groupes (cf. Figure 30 p.104). Si un Groupe doit utiliser un flux avec un autre Groupe, il incorpore tous les composants nécessaires à l'élaboration de ce flux. Notre modèle ne permet pas d'échange de flux d'information entre instances de Groupes mais autorise donc des composants et des flux communs.

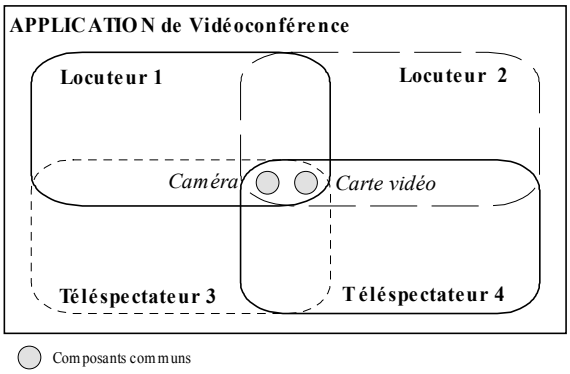


Figure 30 : Composants communs entre Groupes pour une vidéoconférence

De la même manière, un Sous-Groupe est défini comme l'ensemble des composants fournissant une fonctionnalité à un utilisateur. Il ne partage donc pas d'information avec les autres Sous-Groupes mais partage des composants et des flux communs du point de vue de la modélisation : dans le Groupe Locuteur, la caméra citée dans le paragraphe précédent appartient aux Sous-Groupes permettant la transmission de l'image pour que le locuteur surveille ce que les spectateurs voient. Elle appartient également au Sous-Groupe chargé du suivi automatique du locuteur. La plate-forme aura donc la charge de gérer la cohérence du modèle aussi bien au niveau Groupe qu'au niveau Sous-Groupe.

Cette propriété de non-transfert de flux entre Groupe et Sous-Groupe trouve son origine dans le fait que la structuration de l'application n'est pas basée sur les flux mais sur les services et fonctionnalités fournies à l'utilisateur. 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. Un Groupe ou un Sous-Groupe n'est pas une partie de la structure de l'application mais une partie du service. C'est pourquoi la fonctionnalité utilisée par un Groupe peut appartenir à d'autres. Ce modèle se situe donc au niveau fonctionnel et non pas au niveau structurel puisque la réalisation concrète est autre en ce qui concerne l'échange des flux.

2.2.3. Composants

Un Sous-Groupe, et par conséquent un Groupe, sont constitués de composants. Notre définition des composants de l'application ne se limite pas aux composants logiciels si bien que nous distinguons différents types de composants dont nous identifions la fonction sous le terme de rôle.

2.2.3.1. Définition des composants

Nous définissons un **composant de l'application,** noté $\mathbf{C_i}$**,** comme 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, la QdS est différente et l'utilisation du téléphone dans les deux cas se rapportera à deux configurations différentes de l'application.

2.2.3.2. Nature des composants

Un Sous-Groupe est ainsi obtenu non seulement par un assemblage, une composition fonctionnelle [COP 01] de composants logiciels existants ou développés pour l'application — tels qu'un élément de compression de données — mais aussi de composants matériels tels une caméra, un microphone ou un écran de visualisation. De plus, est considérée comme un composant de l'application une personne qui réalise un rôle utile à une fonctionnalité c'est-à-dire qui joue un rôle pour les autres utilisateurs : c'est le cas d'un traducteur qui permet de transmettre les paroles d'une vidéoconférence de manière intelligible pour certains téléspectateurs et propose donc une certaine QdS particulière.

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 [FRA 91] dans le cas de la co-conception matérielle/logicielle pour les composants logiciels et matériels :

« *a more fruitful approach to computer system design is to combine the hardware and software perspectives from the earliest stages of the design process and exploit the design flexibility and efficient allocation of function that such an approach offers*.»

soit

« 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. »

Or la flexibilité de conception et l'allocation efficace de fonctions sont des gages d'un développement et d'une maintenance optimaux des applications et donc d'un faible coût. De plus, la flexibilité de conception améliore la flexibilité de réalisation et l'adaptabilité en tant que « capacité [d'un système] à réagir selon le contexte, et selon les besoins et préférences des utilisateurs » [BAS 93].

Dans notre plate-forme, la QdS de la fonctionnalité fournie sera définie en cours d'exécution par le choix des composants du Sous-Groupe ou par la configuration de ces composants s'ils présentent plusieurs qualités de service. Ainsi on pourra, par exemple, agir sur la qualité des images pour s'adapter aux variations de débit des réseaux. Ces adaptations seront facilitées par la modélisation unique des composants quelle que soit leur nature. D'autre part, un composant peut réaliser plusieurs fonctionnalités comme transcoder à la fois du son et de l'image.

2.2.3.3. Différents types de composants

La nature des composants, logiciels, matériels, humains, n'est pas le seul élément permettant de les distinguer.

En effet, certains composants peuvent être duplicables comme les composants logiciels utilisant des ressources et des données renouvelables. Si besoin, l'application pourra donc utiliser différentes implantations du même composant au même moment avec différentes

qualités de service si celui-ci en propose plusieurs. Les composants de compression vidéo sont de ce type.

D'autres composants peuvent être déplaçables c'est-à-dire délocalisables. Ce sont exclusivement des composants logiciels. A priori, on ne peut en exclure aucun même si en pratique certains trop exigeants en ressources ou utilisant des ressources non renouvelables seront difficilement délocalisés. Ainsi sont délocalisables la plupart des composants logiciels comme, par exemple, ceux de traitement d'image alors que ne sont pas délocalisables les composants matériels et humains.

Enfin, la nature des applications multimédias impose à tout Sous-Groupe de rendre un service observable à l'utilisateur. Il existe donc dans chaque Sous-Groupe un composant que nous nommerons **composant observable** car c'est lui qui permet à l'utilisateur de percevoir la partie de service qui lui est rendu. Grâce à ce composant observable, il peut juger la QdS de la fonctionnalité : c'est donc un composant évaluable dont l'évaluation traduit la qualité du service fourni par le Sous-Groupe. Pour une fonctionnalité typiquement multimédia, comme la restitution des images, le composant observable est l'écran de visualisation. Pour une fonctionnalité plus classique comme le stockage d'informations, le composant observable est celui qui informe l'utilisateur de l'obtention du stockage. La qualité du service est alors essentiellement liée à la capacité de stockage et à la qualité de l'image stockée.

De plus, par définition, un Sous-Groupe fournit une seule fonctionnalité, il contient donc un seul élément observable. En effet, si le Sous-Groupe ne contient pas d'élément observable, il ne peut fournir de fonctionnalité et s'il en contient plusieurs, il fournit plusieurs fonctionnalités. Par définition du Sous-Groupe, le composant observable existe dans tout Sous-Groupe et est unique. Pour nous, un Sous-Groupe correspond à une seule fonctionnalité qui s'observe au travers d'un seul élément observable. De même, un élément observable traduit une seule fonctionnalité correspondant à un seul Sous-Groupe.

2.2.3.4. Rôle des composants

Chaque composant participe au rendu d'une fonctionnalité et d'un service à l'utilisateur. Nous définissons le **rôle** d'un ou de plusieurs composants, noté $\mathbf{r_i}$, comme sa fonction au sein de l'application. Le composant étant l'entité atomique, la brique de base de la construction de l'application, nous nommerons **rôle atomique**, noté $\mathbf{R_i}$, un rôle qui peut être réalisé par un seul composant de l'application. En effet, dans le cas général, une fonction de l'application pourra être réalisée par un composant ou par plusieurs en fonction de la QdS souhaitée. Ainsi, dans le cas de composants de traitement d'image, un traitement à plusieurs étapes peut être réalisé de manière peu performante en réutilisant un composant généraliste déjà existant ou de manière très performante par une suite de composants développés pour une application donnée — composants ad hoc. Le caractère atomique d'un rôle n'implique donc pas que ce rôle soit toujours réalisé par un seul composant.

Nous nommons **rôle observable** le rôle associé à un composant observable. Un rôle observable est donc un rôle qui permet à l'utilisateur de percevoir la partie de service qui lui est rendu. Il existe donc un et un seul rôle observable par Sous-Groupe.

2.2.4. Modèle des applications

Les applications sont construites à partir des différentes entités que nous avons définies, Groupe, Sous-Groupe, composant, Conduit et Processeur Élémentaire. Nous pouvons donc présenter le modèle structurel ainsi que la représentation utilisée lors de la conception.

2.2.4.1. Structure des applications

Nous proposons un modèle structurel hiérarchisé des applications. Celles-ci sont composées de Groupes eux-mêmes constitués de Sous-Groupes. Ces Sous-Groupes sont obtenus par l'assemblage de composants qui communiquent en échangeant des flux. La structure générale d'une application peut donc être représentée par le schéma en langage U.M.L. — *Unified Modeling Language* — de la Figure 31.

Nous choisissons une représentation à l'aide du langage U.M.L., puisqu'il est actuellement reconnu comme un standard [BOO 00] et qu'il ne contraint en rien les méthodes d'analyse utilisées par la suite [DUM 03].

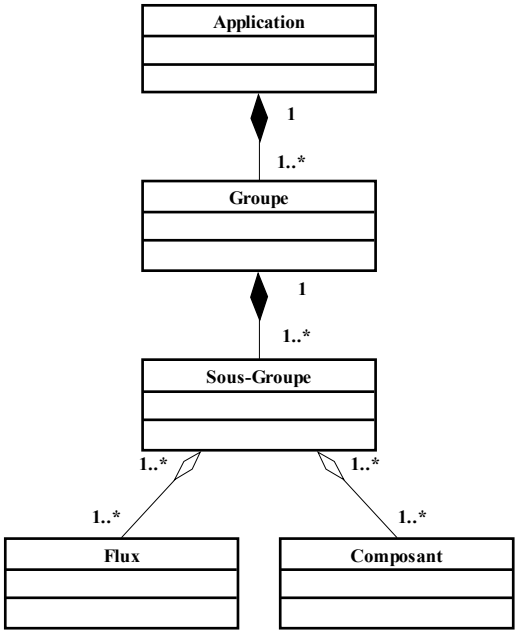


Figure 31 : Modèle structurel des applications multimédias réparties

2.2.4.2. Exemples

a. Une vidéoconférence

La structure de notre modèle d'application telle qu'elle est définie à ce niveau de l'étude peut donc être représentée dans le cas de notre exemple de vidéoconférence par la Figure 32 qui précise la constitution du Groupe Téléspectateur de l'utilisateur 3.

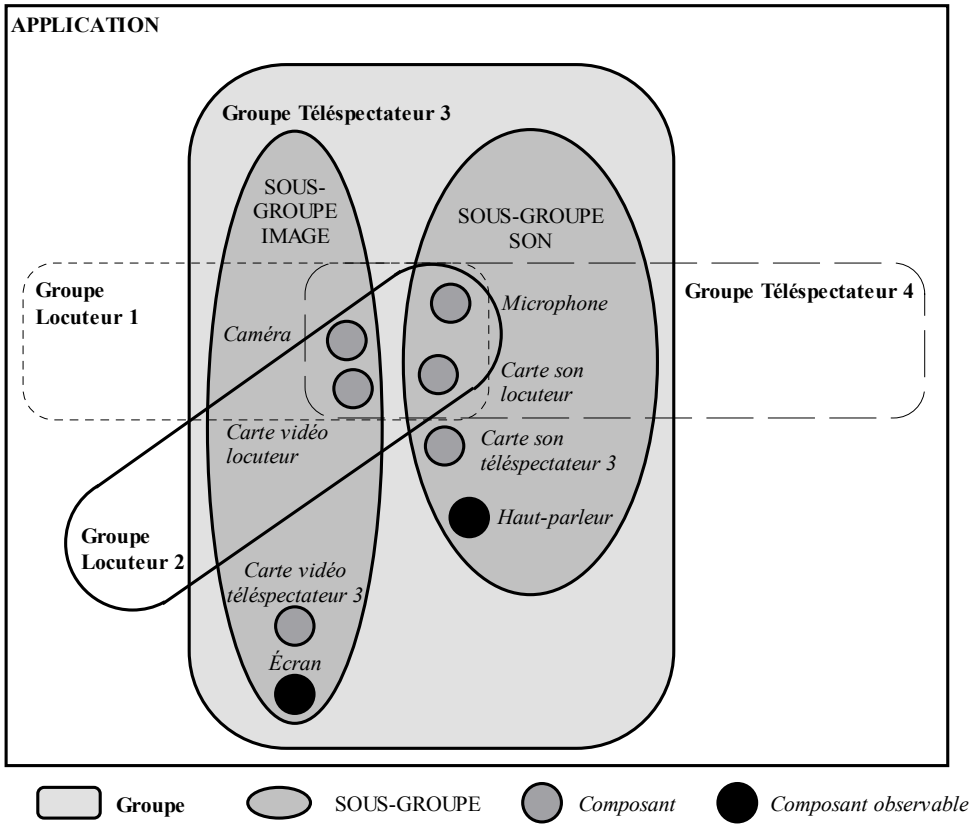


Figure 32 : Modèle d'une vidéoconférence

D'autres applications multimédias conversationnelles [HAF 98] (cf. §1.3.1.2 p.60) donnent lieu à ce même type de modélisation. C'est le cas du télé-enseignement où s'ajoutent éventuellement aux Sous-Groupes liés au son et aux images des fonctionnalités permettant, par exemple, la visualisation synchrone de diaporama ou l'échange de messages. Ce type de modèle pourrait également être utilisé pour le télédiagnostic médical. Dans ce cas, un Sous-Groupe peut permettre la commande à distance d'un appareil d'échographie introduisant ainsi une multimodalité qui va au-delà du multimédia classique comme dans l'exemple suivant.

b. Une vidéosurveillance

La vidéosurveillance, en particulier dans le domaine de la surveillance routière, nous permet de mettre en exergue notre vision des applications multimédias qui ne se restreignent pas à la transmission des média classiques que sont le son et l'image mais traitent également des flux discontinus d'événements tels que des alarmes. Nous pouvons citer ainsi le cas concret de la surveillance du Pont d'Aquitaine à Bordeaux mise en place au premier trimestre 2006 où un réseau de caméras permet de surveiller la fluidité du trafic et de détecter, entre autre, la survenue d'accidents.

Sur ce modèle et en l'absence de réseau privé, nous proposons l'exemple d'une vidéosurveillance où un seul Groupe Téléspectateur est défini : le service rendu est celui de

surveillance à distance par la transmission du son et des images captés par les caméras ainsi que la détection automatique de la survenue d'un bouchon routier. S'y ajoute la possibilité de modifier la configuration du trafic par la réduction du nombre de voies disponibles pour les usagers grâce à un affichage commandé à distance permettant de passer par exemple de deux fois trois voies à deux fois deux voies de manière à faciliter l'accès éventuel de secours. Ce Groupe peut donc être composé de quatre Sous-Groupes : un Sous-Groupe Son, un Sous-Groupe Image, un Sous-Groupe Alarme et un Sous-Groupe Commande. Plusieurs surveillants peuvent simultanément contrôler des parties différentes du trafic si bien que certaines caméras pourront appartenir à plusieurs instances de Groupe. Ces caméras appartiendront à général à deux Sous-Groupes au sein d'un même Groupe : le Sous-Groupe Image et le Sous-Groupe Alarme.

2.2.4.3. Configuration canonique

En fonction du contexte et de la qualité à obtenir, l'application et les Groupes pourront contenir une, plusieurs ou aucune entité appartenant à chaque type possible du niveau immédiatement inférieur. Ainsi l'application pourra contenir plusieurs instances d'un même Groupe et aucune d'un autre. Lors de la conception de l'application, il est nécessaire de raisonner sur des configurations qui englobent la totalité des problématiques à adresser. C'est pourquoi nous définissons la notion de **configuration canonique** d'une application ou d'un Groupe comme une configuration fictive contenant une instance et une seule de toutes les classes de modularité immédiatement inférieure disponibles pour construire cette entité. Nous mettons ainsi en exergue l'ensemble des relations envisageables entre ces éléments de modularité inférieure et nous disposerons des informations nécessaires pour construire l'ensemble des configurations de l'entité.

Ainsi la configuration canonique d'un Groupe contient un Sous-Groupe et un seul de chaque classe de Sous-Groupe. Elle permet en particulier la définition des flux synchrones au sein du Groupe en identifiant les flux d'informations d'un Sous-Groupe qui doivent demeurer synchronisés avec des flux d'un autre Sous-Groupe. La Figure 33 page 110 représente les configurations canoniques de notre vidéoconférence. L'application contient deux instances de Groupes car il y a deux Groupes possibles. Le Groupe Locuteur contient deux instances de Sous-Groupes permettant la transmission synchronisée du son et de l'image ainsi que le suivi des déplacements du locuteur par la caméra. Le Groupe Téléspectateur contient uniquement les Sous-Groupes Son et Image. Cette figure est à comparer avec la Figure 32 page 108 qui représente la même application en cours d'exécution et est donc composée d'autant d'instances de Groupes que d'utilisateurs.

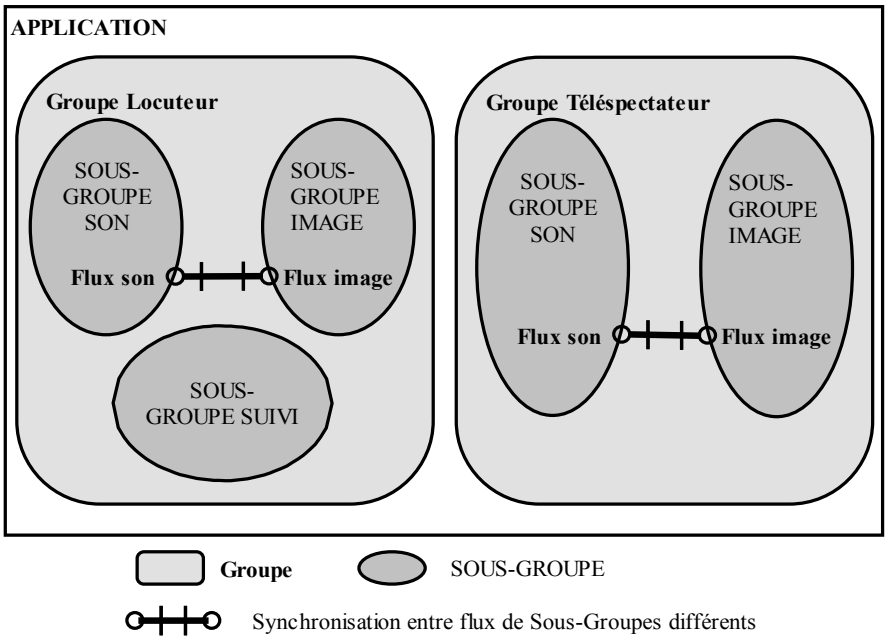


Figure 33 : Configurations canoniques de l'application et du Groupe Téléspectateur

Après avoir établi le modèle structurel de l'application, il nous reste à modéliser les liens entre les composants de l'application mutimédia sachant que la principale difficulté est d'assurer la synchronisation.

2.2.5. Entités pour la synchronisation

La synchronisation inter-flux (cf. §1.3.1.3.a p.61), que nous étudions ici, est l'une des contraintes temporelles les plus exigeantes parmi celles liées au multimédia. De même que la QdS, les contraintes de synchronisation font partie des préoccupations non fonctionnelles que nous choisissons de traiter à part de façon à faciliter la conception des applications dédiées au grand public et d'en minimiser le coût. Nous considérons donc la synchronisation comme un autre aspect de l'application — au sens de la programmation par aspects (cf. §1.2.1 p.45). Ce choix se justifie également par l'interaction sémantique existant éventuellement entre les flux d'informations dans les applications multimédias. En effet, différents flux, généralement le son et l'image, s'unissent pour créer la sémantique des informations à tel point que l'absence de l'un ou de l'autre rend l'information moins compréhensible voire incompréhensible.

La qualité de la synchronisation ne sera pas un critère de QdS puisque nous la posons comme condition *sine qua non* à la réalisation de notre application. Cette approche restrictive pourra être affinée dans des travaux futurs de notre équipe par la possibilité de nuancer la synchronisation pour régler la QdS mais, pour l'instant, les données sont transmises de façon synchrone quitte à être obligé d'en réduire la qualité. Ainsi, si elle n'est pas un critère de QdS, la synchronisation influera cependant sur la QdS.

Pour obtenir cette synchronisation, nous proposons d'utiliser les deux entités présentées par notre équipe de recherche dans des travaux connexes à cette thèse [DAL 05]. La première, le Conduit, permet de réaliser une transmission synchronisée d'informations. La seconde, le Processeur Élémentaire, est un conteneur de composant métier qui assure la synchronisation des flux. Ces deux entités permettent aux composants de coopérer et assurent la transmission synchronisée d'informations. A ce titre, elles appartiennent donc toutes deux à l'application.

2.2.5.1. Flux : Conduits

Les **Conduits,** notés $\mathbf{Cn_i}$, sont des connecteurs qui permettent d'assurer la transmission des flux d'informations, notés $\mathbf{Fl_i}$, entre les différents composants. Ils peuvent contenir un ou plusieurs flux (cf. Figure 34). Dans ce dernier cas, tous les flux transitant par un même Conduit sont synchronisés. Ainsi c'est un unique Conduit qui assurera la transmission des images et du son captés par le microphone et la caméra lors du discours d'un locuteur dans notre exemple de vidéoconférence. C'est également à l'aide d'un Conduit qu'une vidéo et des sous-titres pourront être transmis en conservant la synchronisation inter flux. En outre, un Conduit peut être réparti sur le réseau si les composants à l'origine et à la destination des flux sont répartis sur des sites différents. Le Conduit encapsule alors le protocole de communication.

Les Conduits sont capables de communiquer avec la plate-forme. En particulier, ils génèrent des événements de demande de reconfiguration s'ils détectent une saturation d'un flux ou une possibilité d'augmentation du débit sur ce flux. De plus, la plate-forme peut les connecter et les déconnecter si cela est nécessaire à une reconfiguration.

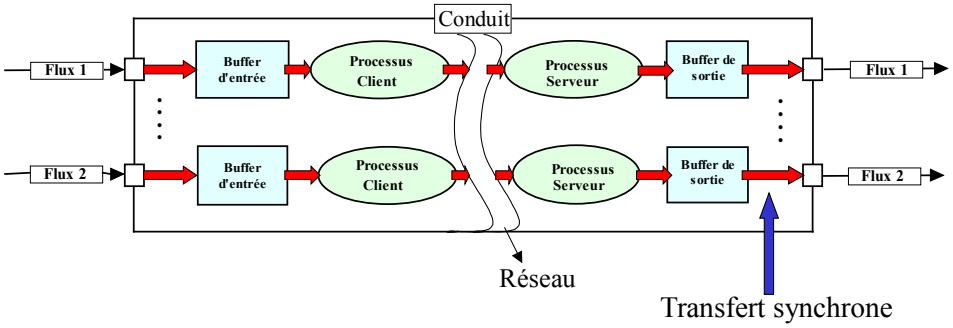


Figure 34 : Synchronisation des flux à l'aide des Conduits [BOX 05]

2.2.5.2. Processeurs Élémentaires

Les **Processeurs Élémentaires**, notés $\mathbf{PE_i}$, sont des conteneurs de composants logiciels. Ils encapsulent la logique métier et permettent d'implanter des préoccupations non fonctionnelles. Comme les Conduits, ils sont supervisables par la plate-forme de manière à assurer une adaptation correcte de l'application. En revanche, contrairement à eux, ils ne peuvent pas être répartis sur le réseau : un Processeur Élémentaire se situe sur un unique poste informatique.

Leur rôle principal est d'assurer le maintien de la synchronisation entre plusieurs flux lorsque seulement quelques-uns doivent être traités par le composant métier (cf. Figure 35

p.112). Le Processeur Élémentaire est alors connecté à tous les Conduits contenant les flux concernés. Certains flux sont traités par le composant métier alors que d'autres ne le sont pas. Grâce à des mécanismes de datation, le Processeur Élémentaire est capable de maintenir la synchronisation de la totalité des flux qui transitent [DAL 05]. Ainsi en encapsulant les composants métiers dans un Processeur Élémentaire, nous garantissons que les traitements auront lieu sur les flux concernant ce composant sans perdre la synchronisation inter et intra flux.

Par exemple, lors d'un traitement sur la taille des images dans la transmission d'une vidéo, ni le son ni les sous-titres ne sont modifiés mais l'image doit être réduite. Le Processeur Élémentaire recevra alors les trois flux d'informations mais seul le flux des images sera traité par le composant métier. En sortie, le flux d'images traité sera synchrone avec les flux de son et de sous-titres qui n'ont fait que transiter par le Processeur Élémentaire.

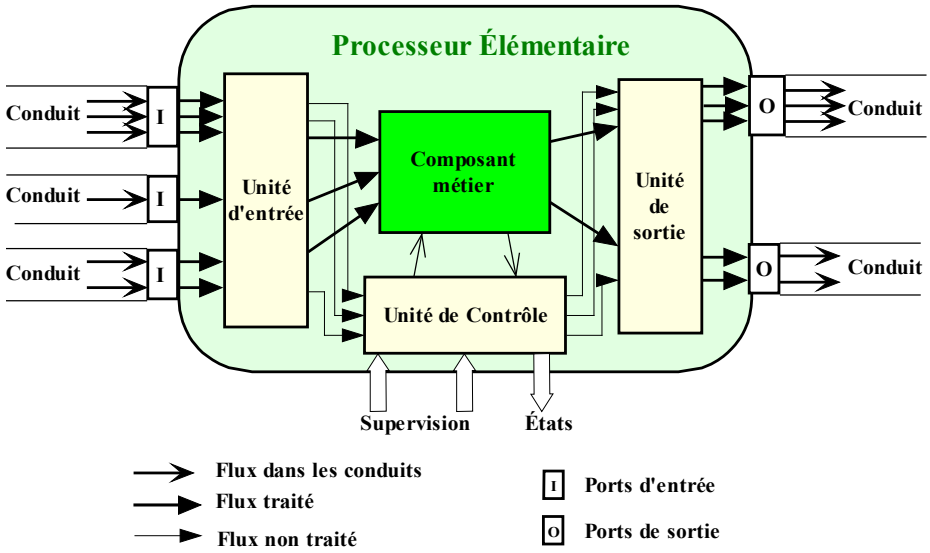


Figure 35 : Synchronisation des flux à l'aide des Processeurs Élémentaires [BOX 05]

2.2.5.3. Cas particulier des entités pour la synchronisation

En utilisant les Processeurs Élémentaires et les Conduits, nous avons choisi de traiter la synchronisation comme un autre aspect de l'application au même titre que la QdS. C'est pourquoi nous proposons un modèle structurel de l'application qui ne les représentent pas explicitement sachant cependant que tout composant et tout flux sera encapsulé dans de telles entités — sauf exception comme les composants matériels. Nous serons donc tenus de les faire apparaître à un autre stade de la représentation des applications (cf.2.3.1.2.d p.119).

Ce choix se justifie par l'impossibilité de définir la manière dont les Processeurs Élémentaires et les Conduits vont être utilisés pour assurer la synchronisation en l'absence de toute information sur l'implantation de l'application. En effet, le flux sortant d'un composant ne sera pas toujours synchronisé avec les flux sortant des mêmes composants en

fonction de leurs localisations respectives. Par exemple, dans le cas d'une synchronisation entre images et son, un Conduit réparti véhiculant le flux sonore contiendra un flux issu de la caméra ou un flux issu d'un composant de traitement d'image en fonction de la localisation de ce dernier. Le son partagera toujours un Conduit avec un flux contenant des images mais ce ne sera pas toujours le même en fonction de l'implantation de l'application.

C'est pourquoi dans le modèle structurel apparaissent uniquement des informations donnant les différents liens de synchronisation entre Sous-Groupes.

2.2.6. Synthèse

En raison des contraintes de développement imposées aux applications destinées au grand public, nous proposons de concevoir et d'exécuter les applications multimédias réparties sur une plate-forme. Nous proposons ici un modèle d'application qui nous permet de concevoir la plate-forme d'exécution. Du point de vue structurel, l'application est construite en fonction de la vision qu'a l'utilisateur du service qui lui est fourni — le Groupe (cf. Figure 36). Elle est composée de différentes fonctionnalités — les Sous-Groupes. Ces derniers sont constitués de composants, logiciels, matériels ou humains, reliés par des Conduits assurant la transmission des flux de données et encapsulés dans des Processeurs Élémentaires pour ce qui est des composants logiciels. Les Conduits et les Processeurs Élémentaires permettent de conserver la synchronisation des flux quel que soit le contexte. En effet, nous considérons cette synchronisation comme obligatoire à la délivrance d'un service. Elle ne sera donc pas négociable par la plate-forme.

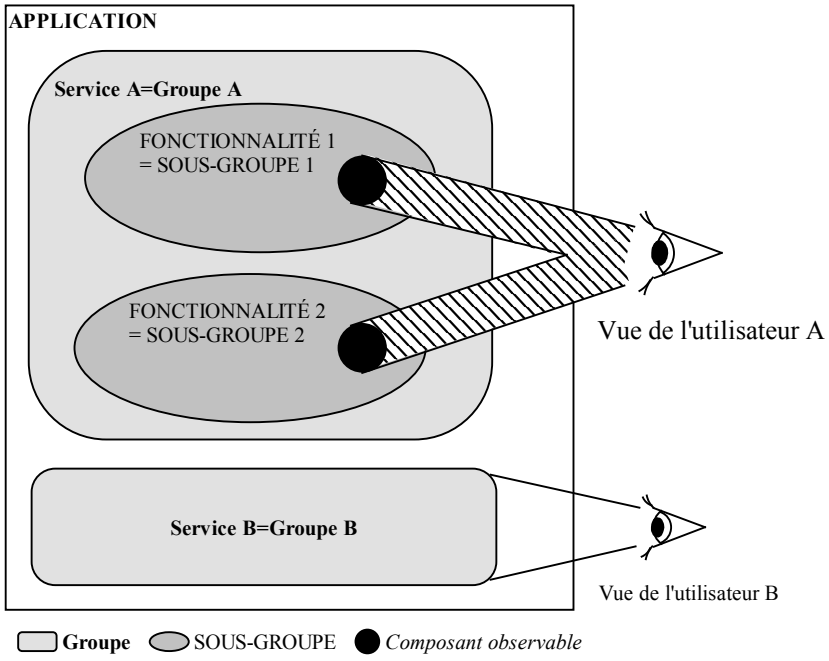


Figure 36 : Modélisation des applications à partir de la vision des utilisateurs

Grâce à ce modèle, l'analyse de l'application 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 de la QdS puis de s'intéresser aux fonctionnalités de bas niveau de l'application avant de procéder à l'implantation. Nous pouvons donc parler d'une conception centrée qualité de service.

2.3. Représentation des applications et de leurs qualités de service

Après avoir défini nos modèles de qualité de service et d'applications multimédias réparties, nous en proposons une représentation justifiée par la nécessité d'offrir un mode de conception adapté à la plate-forme d'exécution. Ainsi nous utilisons le même principe de représentation graphique pour les spécifications fonctionnelles de l'application, pour son implantation et pour l'évaluation de sa QdS. Enfin, disposant de tous les modèles et de toutes les représentations nécessaires, nous montrerons comment les utiliser pour construire une application supervisable par notre plate-forme.

2.3.1. Principes généraux

Nos choix de représentation sont motivés par notre objectif de disposer d'une représentation utilisable à la fois par le concepteur et par la plate-forme lors de l'exécution.

2.3.1.1. Justification de la représentation

Le modèle structurel que nous avons établi est centré sur l'utilisateur de manière à pouvoir modifier la structure de l'application pour optimiser la QdS. Cependant, l'étape préalable à l'optimisation du service est de s'assurer que ce service soit effectivement obtenu. C'est au concepteur de l'application qu'incombe cette vérification. Nous lui proposons une représentation des spécifications fonctionnelles de l'application qui soit manipulable par la plate-forme. Nous avons choisi les graphes car ce sont des modes de représentation adaptés à l'informatique mais surtout car ce sont des outils utilisés pour des problématiques proches de la nôtre comme nous l'avons vu précédemment (cf. §1.4.2 p.72).

Ainsi nous modélisons l'application par des graphes dont les différentes versions représentent les différents aspects 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.

Il existe une similitude de forme entre ces différentes représentations— et même un isomorphisme au sens mathématique du terme dans certains cas —, c'est pourquoi il est possible de considérer chaque étape comme une version différente d'un graphe plus complexe modélisant l'intégralité de l'application. Chaque partie de l'application, Groupe,

Sous-Groupe, pourra, si besoin, être modélisée séparément à l'aide de ces différents graphes.

Nous présentons maintenant le principe de cette représentation graphique puis les trois graphes modélisant l'application à proprement parler : le graphe des flots de contrôle, le graphe fonctionnel et le graphe de configuration. Le graphe d'évaluation fera l'objet d'une présentation séparée dans la partie dédiée à l'évaluation de la QdS.

2.3.1.2. Symbolisme graphique

Pour représenter l'application, nous proposons d'adapter le Graphe des Processus (cf. §1.4.2.1 p.73) et le Graphe Conditionnel des Processus (cf. §1.4.2.2 p.73) [ELE 00] à la problématique des applications multimédias réparties sur Internet. Nous utilisons un type de Graphe des Processus pour le graphe des flots de contrôle et le graphe fonctionnel alors que 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.

D'une manière générale, nous utiliserons des graphes orientés et polaires $G(V, E_s, E_c)$ dont la syntaxe est présentée ici dans ses grandes lignes.

a. Nœuds et arêtes

Les nœuds $P_i \in V$, avec V ensemble des nœuds, représentent des aspects différents des composants de l'application en fonction du graphe :

- le rôle d'un ou de plusieurs composants dans le graphe des flots de contrôle et le graphe fonctionnel ;
- un composant matériel ou humain ou un Processeur Élémentaire encapsulant un composant logiciel dans le graphe de configuration ;
- l'influence d'un composant ou d'un Processeur Élémentaire sur la QdS dans le graphe d'évaluation.

Dans le cas des graphes de configuration et d'évaluation, respectivement les Conduits et leurs influences sur la QdS seront également représentés par des nœuds. Ces nœuds auront des attributs comme la localisation des Processeurs Élémentaires dans le graphe de configuration.

Les arêtes $e_{ij} \in E$, avec E ensemble des arêtes et e_{ij} arête de P_i à P_j, identifient des informations liées aux flux de données entre composants. En fonction du graphe, elles signifient que :

- les données en sortie du composant P_i sont en entrée du composant P_j dans le graphe fonctionnel tandis que, dans le graphe de configuration, elles indiquent que ces données transitent par un Conduit ou par une connexion matérielle ;
- les valeurs de QdS en sortie du composant ou flux P_i se retrouvent en entrée du composant ou flux P_j dans le graphe d'évaluation ;

- le composant en amont doit être placé immédiatement avant le composant en aval dans le graphe des flots de contrôle.

Nous utiliserons également les deux types d'arêtes, simple ou conditionnelle (cf. §1.4.2.1 p.73), en leur octroyant une signification particulière. Les arêtes simples signifient que le nœud suivant appartient à l'application quelles que soient la configuration et la QdS choisies.

Une arête conditionnelle peut indiquer que :

- le nœud suivant appartient à l'application uniquement dans le cas de la configuration ou du choix décrit par la condition associée ;
- que le nœud origine de l'arête, un rôle ou un composant, impose l'utilisation d'un autre nœud, rôle ou composant, à un autre endroit du graphe.

Dans le dernier cas, elle permet d'exprimer la dépendance entre composants même lorsqu'ils ne sont pas reliés directement par un flux. Grâce aux arêtes conditionnelles nous visualisons ainsi les choix de rôles, de composants, de localisation ainsi que les contraintes fonctionnelles ou de compatibilité entre composants.

b. Orientation

L'orientation du graphe suivra la direction des flux d'informations si bien que le composant observable se trouvera en bout de chemin. Cependant nous conservons le caractère polaire du graphe car nous utilisons deux nœuds factices appelés de la même façon que [ELE 00] source et puits. Le puits aura ici une signification particulière puisqu'il sera le successeur immédiat du composant ou rôle observable. Le puits représente donc l'utilisateur puisque le composant observable traduit la vision que l'utilisateur a du service rendu par la partie de l'application décrite par le chemin situé entre la source et le puits. La source, quant à elle, réifie le point de départ en particulier pour le concepteur : celui-ci, en effet, suit rarement le sens des flux mais part plutôt du service qu'il veut rendre à l'utilisateur, et donc finalement du rôle observable, puis décrit tous les rôles nécessaires à l'obtention de ce service pour achever sa description au nœud source. Le concepteur aura donc une vision inversée du parcours du graphe par rapport à celle de l'utilisateur qui le suit de la source vers le puits.

c. Cycles

Nous utilisons les graphes proposés par [ELE 00] pour construire notre représentation de l'application. Cependant les graphes de flots de contrôle, les graphes fonctionnels et les graphes de configuration peuvent contenir des cycles à l'inverse des propositions de cet auteur. En effet, certains composants peuvent constituer une unité de contrôle et donc être inclus dans un cycle. Il y a alors deux cas de figure en fonction de la stabilité de l'assemblage de composant.

Dans le premier, cet assemblage est instable, son comportement oscille et il n'y aura donc aucune gestion possible de la qualité du service. Nous ferons donc l'hypothèse que l'application ne contient pas ce type d'assemblage ce qui correspond à faire l'hypothèse que le concepteur conçoit correctement l'application par exemple en utilisant des outils spécialisés de vérification [MIC 05].

Dans le second cas, la configuration est stable. Le cycle peut être assimilé à un système asservi où une information de contrôle — retour ou feedback — influe sur le signal ou l'information de sortie au même titre que le signal d'entrée comme au paragraphe 1.3.3.b. L'asservissement permet alors de résoudre un problème souvent lié à la précision en utilisant le parcours itératif du cycle durant un certain temps. Le retour permet de mémoriser et de comparer le résultat obtenu à un instant donné de manière à l'améliorer. L'utilisation d'un cycle introduit donc une dimension temporelle puisque le cycle est efficace uniquement lorsqu'il s'est stabilisé.

En automatique, la qualité d'un cycle dépend de critères statiques comme l'erreur statique ou de critères dynamiques comme l'erreur de traînage ou le temps de réponse. Elle est déterminée à partir des caractéristiques des éléments composants le cycle. C'est pourquoi nous considérons que la QdS d'un groupe de composants assemblés en cycle dépend aussi bien de la QdS de ces composants que d'informations liées à l'histoire de ce cycle de manière à prendre à compte le temps de réponse.

Pour la définir, nous pourrions utiliser des méthodes inspirées de l'automatique mais cela nous semble sortir du cadre de ces travaux. En effet, nous aurions alors besoin d'introduire une dimension temporelle dans l'évaluation de la QdS. Or pour l'utilisateur, centre de nos préoccupations, ce temps n'est pas significatif puisqu'il ne lui est pas possible de définir dans quelle phase — stabilisation ou non — se situe le cycle et donc si l'évaluation est significative c'est-à-dire si le temps de réponse est atteint ou non et par conséquent si la valeur du signal de sortie est significative de la QdS ou non.

C'est pourquoi nous proposons de considérer le cycle comme un seul bloc indivisible du point de vue de l'évaluation de la QdS car la QdS instantanée des signaux internes au cycle n'est vraisemblablement pas représentative de la QdS de la fonction ou rôle assurée par ce cycle. Nous assimilons alors le cycle de composants à un composant insécable. C'est pourquoi les cycles de flots de données présents dans l'application et modélisés dans les graphes de configuration n'aboutissent pas à des cycles dans le graphe d'évaluation de la QdS. En outre, dans la plupart des cas, les données transitant dans le sens inverse de l'écoulement des flux principaux ne sont pas multimédias mais sont des données de contrôle.

Dans la vidéoconférence, le Sous-Groupe assurant le suivi du locuteur est réalisé par des composants inclus dans un cycle (cf. Figure 37 p.119) : une caméra motorisée, une carte vidéo, un composant de traitement déterminant le mouvement de la caméra en fonction des déplacements du locuteur sur l'image [LUT 99] et un écran. Nous considérons que ce cycle n'influe sur la QdS que par son temps de traitement. Il équivaut alors à un composant insécable dont le temps de traitement est donné par la somme des temps des trois composants du cycle.

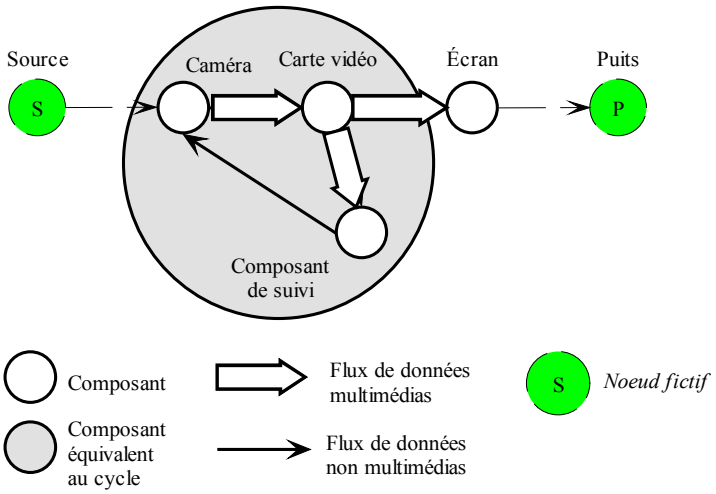


Figure 37 : Représentation informelle d'un cycle de composant

d. Exploitation des graphes

Les graphes que nous proposons ont deux fonctions principales :

- aider le concepteur dans sa tâche en lui permettant de représenter l'application aux différents stades de sa réalisation ;
- proposer un outil de représentation et d'évaluation des applications à la plate-forme lui permettant ainsi de les superviser.

Ainsi, le graphe des flots de contrôle définit l'ordre d'exécution des traitements appliqués aux flux d'informations. Les éléments le composant sont graduellement affinés de manière à aboutir au graphe fonctionnel en une analyse fonctionnelle descendante. Sont alors définis les rôles atomiques et les échanges de données de l'application.

A partir des informations fournies par le concepteur — composants disponibles, contraintes de localisation — le graphe fonctionnel est complété de manière à décrire, pour chaque rôle atomique, les différents choix de composants et de localisation possibles ainsi que la présence de Conduits et de Processeurs Élémentaires. C'est le graphe de configuration qui est alors obtenu.

Comme chaque nœud de ce graphe représente un élément qui influe sur la QdS, il est alors possible d'en dériver le graphe d'évaluation qui détermine l'algorithme de définition de la QdS de l'application en remplaçant chaque composant par son influence sur la QdS.

Le Tableau 8 regroupe les significations des nœuds et des arêtes dans les quatre graphes représentant les différents aspects du modèle d'application.

Graphe	Signification	
	Nœuds	Arêtes
Graphe des flots de contrôle	Rôle d'un ou de plusieurs composants	Le nœud amont "doit être immédiatement suivi par" le nœud aval.
Graphe fonctionnel	Rôle d'un composant (atomique)	Le nœud amont "envoie des données au" nœud aval.
Graphe de configuration	Processeur Élémentaire ou Composant	La sortie du nœud amont est l'entrée du nœud aval.
	Conduit ou connexion matérielle	
Graphe d'évaluation	Traitement (influence du nœud sur la qualité de service)	La qualité de service en sortie du nœud amont est en entrée du nœud aval.

Tableau 8 : Constitution des graphes modélisant l'application

2.3.2. Représentation des différents aspects de l'application

Nous avons présenté les représentations graphiques utilisées pour modéliser les différents aspects de l'application. Cependant, pour chaque partie du modèle, et en particulier pour la modélisation fonctionnelle et pour celle de l'implantation, les graphes présentent des spécificités que nous allons décrire. L'une des plus importante est la signification des arêtes conditionnelles.

2.3.2.1. Représentation fonctionnelle

La représentation fonctionnelle de l'application se fait grâce à deux graphes. Le premier, le graphe des flots de contrôle, permet de modéliser les spécifications fonctionnelles indiquées par le concepteur. Le second, le graphe fonctionnel, permet d'obtenir une décomposition fonctionnelle précise de l'application.

a. Graphe des flots de contrôle

Le graphe des flots de contrôle que certains auteurs nomment graphe de précédence [DEM 02] permet au concepteur de spécifier les fonctionnalités de l'application à différents niveaux hiérarchiques. Il est utilisé ensuite pour définir le graphe fonctionnel. Nous proposons que ce graphe soit le support de la décomposition fonctionnelle de l'application. C'est pourquoi il sera surtout utilisé pour modéliser les Sous-Groupes. C'est aussi pour cette raison, qu'en fonction de l'avancée de l'étude de l'application, les nœuds représenteront les rôles des assemblages de composants de cardinalité de plus en plus petite

pour aboutir à un graphe des flots de contrôle comportant uniquement des rôles atomiques (cf. §2.2.3.4 p.106).

Les arêtes indiquent 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 38) 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. 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.

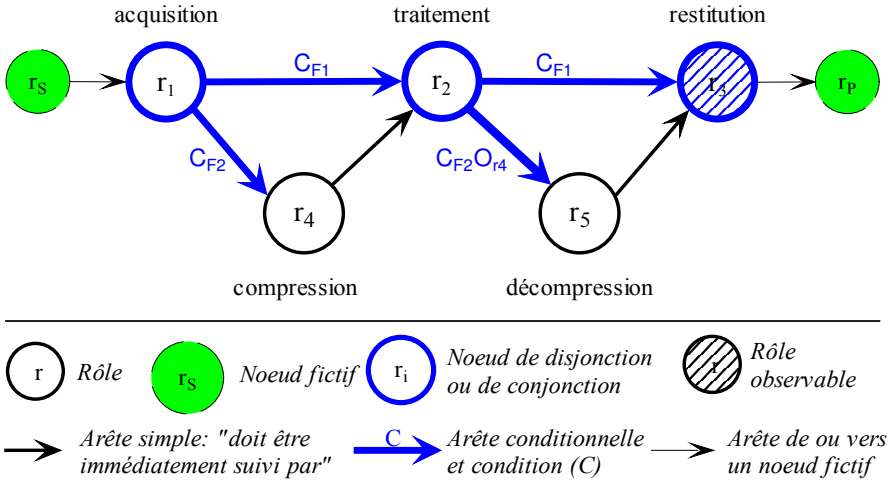


Figure 38 : Exemple de Graphe des flots de contrôle du Sous-Groupe Image

Cette représentation autorise la synchronisation entre rôles n'échangeant pas d'informations : un rôle nécessite qu'un autre ait fini sa tâche sans recevoir d'informations de lui. En d'autres termes, la relation entre " doit avoir pour prédécesseur immédiat " et "doit recevoir des données de" n'est pas une bijection. Une précédence implique k flux d'information avec $k \in N$. 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.

b. Graphe fonctionnel

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 que celui d'un assemblage de composants — 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 38 page 121 est donné par la Figure 39. 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} et 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 imposent de différencier les rôles atomiques et donc les composants d'acquisition, de compression et de traitement.

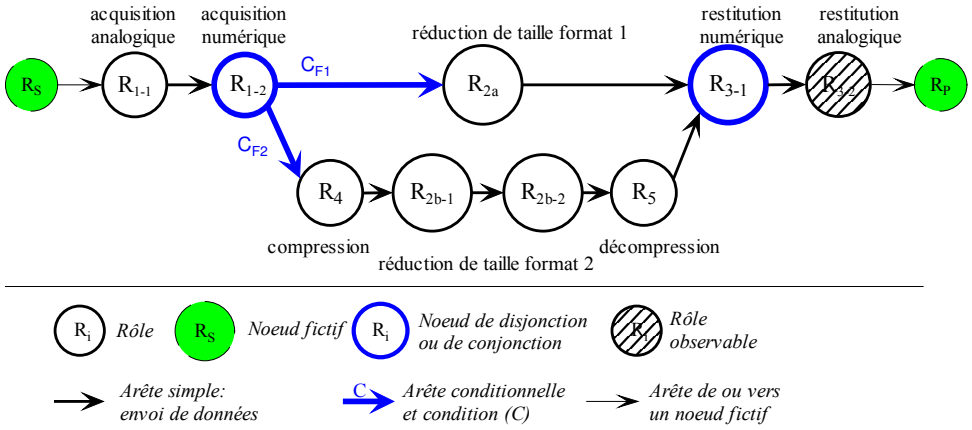


Figure 39 : Exemple de Graphe fonctionnel du Sous-Groupe Image

2.3.2.2. Représentation des configurations

a. Super-graphe des configurations

Le graphe de configuration décrit une configuration de l'application. Nous proposons de définir également un super-graphe des configurations [LIB 01] [CUI 01] (cf. §1.3.2.1 p.62) qui représente toutes les compositions possibles de l'application et leurs implantations. Il précise les composants, les Processeurs Élémentaires, leurs localisations, les Conduits utilisés pour les échanges de flux et les connexions matérielles. Il est construit à partir du

graphe fonctionnel et 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'arrête conditionnelle est utilisée si le second composant n'est pas relié au premier par un chemin simple c'est-à-dire soit directement soit par un chemin sans disjonction. Ainsi dans le graphe des flots de contrôle du Sous-Groupe Image (cf. Figure 38 p.121), 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 est un lieu de conjonction et de disjonction. En revanche, dans le graphe fonctionnel du Sous-Groupe Image (cf. Figure 39 p.122), un chemin direct relie les rôles atomiques de compression et décompression si bien que le choix de la compression — C_{F2} — impose naturellement l'usage d'une décompression.

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.

D'autre part, en cours d'exécution, la plate-forme utilisera les arêtes conditionnelles pour indiquer la localisation choisie pour un composant ce qui lui permettra de discerner les différentes implantations possibles d'une même configuration. Nous noterons C_{Pi} la condition liée à une arête qui ne sera parcourue qu'à condition que le poste P_i soit choisi pour implanter le composant C_i.

Enfin, dans les graphes utilisés en cours d'exécution par la plate-forme, les nœuds représentent d'une part, les Processeurs Élémentaires (cf. §2.2.5.2 p.111) et les Conduits (cf. §2.2.5.1 p.111) pour les composants et flux informatiques et d'autre part, les composants matériels ou humains et les connexions matérielles. Cependant 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 40). Les Processeurs Élémentaires n'assurant pas de synchronisation inter-flux ne seront généralement pas représentés dans les graphes de manière à en améliorer la lisibilité.

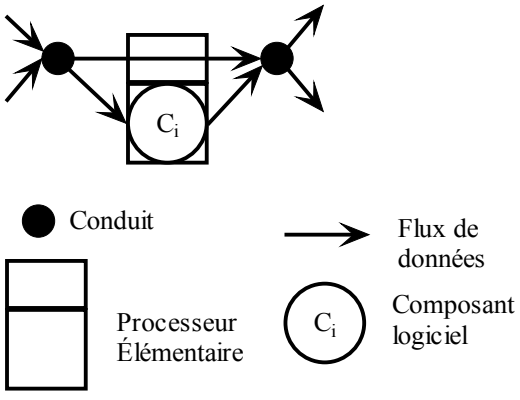


Figure 40 : Représentation de Processeur Élémentaire, de Composant et de Conduit

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

nous proposons que les arêtes conditionnelles soient 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 (cf. Figure 41).

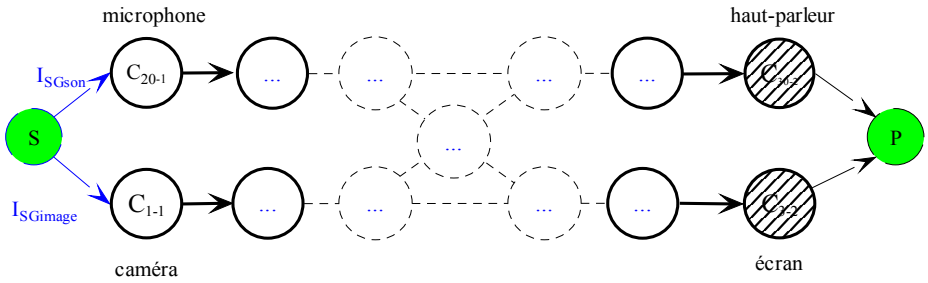


Figure 41 : Structure du super-graphe des configurations d'un Groupe

Dans un premier temps, nous définissons de façon théorique le super-graphe des configurations quel que soit le niveau de granularité — application, Groupe et Sous-Groupe — de manière à disposer d'un outil complet de représentation. Cependant, pour des applications à base de composants, il semble peu réaliste d'envisager que le concepteur ou la plate-forme manipulent un super-graphe au niveau de l'application alors que nous soupçonnons que notre problème sera NP-complet (cf. §1.4.3 p.75). L'étude que nous ferons de la complexité de notre problématique (cf. §3.1 p.151) nous permettra alors de fixer à quel niveau ces graphes seront utilisés. Ainsi nous pourrons proposer une façon réaliste de les construire soit par le concepteur soit de façon automatique en fonction des complexités étudiées sachant que la plate-forme pourra manipuler des graphes plus complexes que l'utilisateur dans la mesure où le problème demeure non polynomial.

b. Graphe d'une configuration

Le super-graphe des configurations G décrit les compositions possibles soit de l'application, soit d'un Groupe, soit d'un Sous-Groupe. Une configuration possible correspond à un sous-graphe $G_k \in G$, auquel n'est plus associée une expression logique comme dans [ELE 00] mais un ensemble d'expressions logiques qui constitue non plus un label (cf. §1.4.2.2.b p.74) mais un langage. 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 car alors elles ne fourniraient qu'une seule fonctionnalité à l'utilisateur, ce qui est le cas pour les Sous-Groupes. 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 la constituant. Une configuration peut donc être définie par le langage ou le sous-graphe $\mathbf{G_k}$ que nous nommons **graphe de configuration.**

c. Exemple du Sous-Groupe Image

La Figure 42 page 126 représente le super-graphe des configurations du Sous-Groupe Image évoqué aux paragraphes 2.3.2.1.a et 2.3.2.1.b. Nous y retrouvons les deux configurations possibles de la Figure 39 page 122, C_{F1} et C_{F2}, et les implantations possibles entre le poste du locuteur P_1 et celui du téléspectateur P_4 (cf. §2.1.2.1 p.84). Une majorité de composants est ici considérée comme indélocalisable. Cependant 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. Deux compressions sont disponibles. Enfin, pour signifier l'aspect réparti de certains Conduits (cf. §2.2.5.1 p.111), ces derniers ont été placés sur l'élément graphique représentant le réseau : ces Conduits sont répartis entre les deux sites.

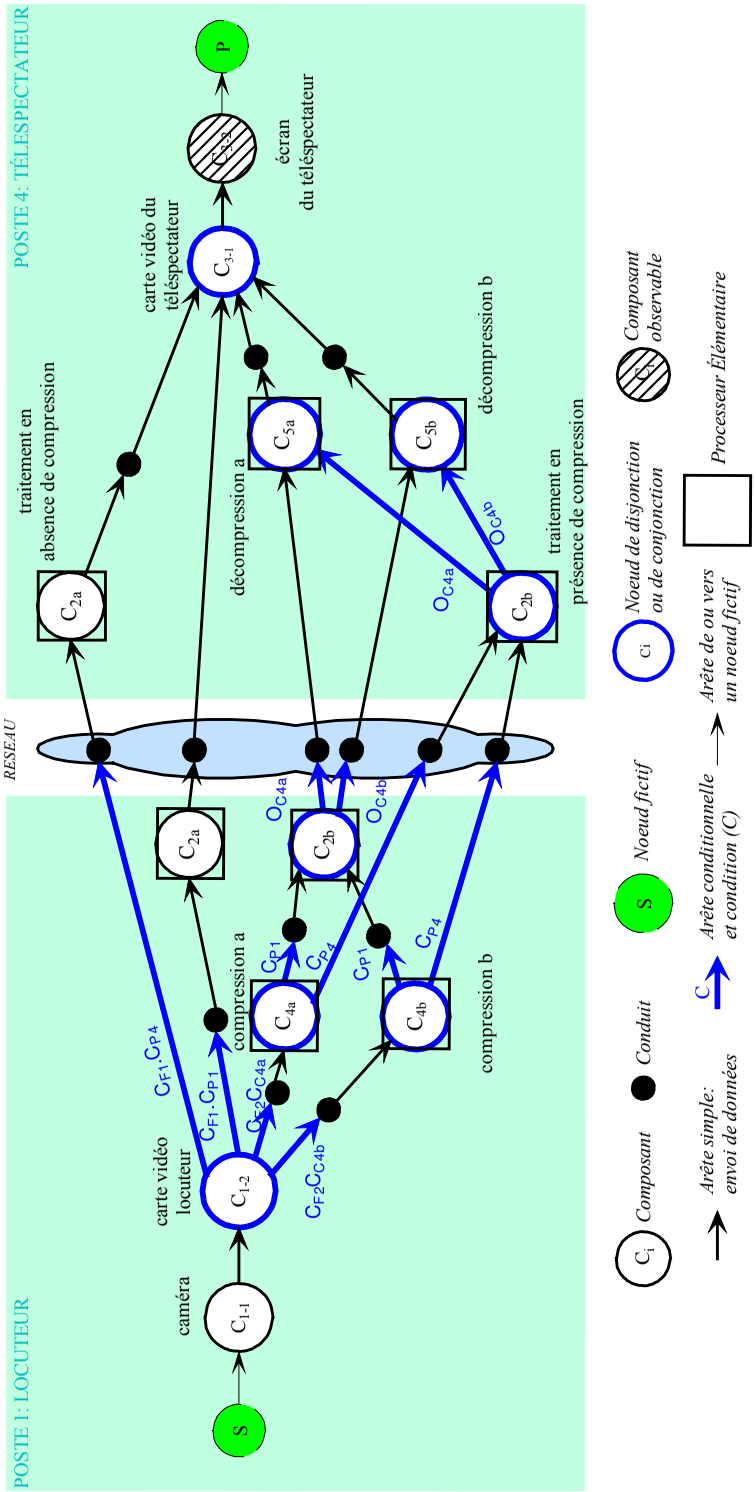


Figure 42 : Exemple de Graphe des configurations du Sous-Groupe Image

2.3.2.3. Signification des arêtes conditionnelles

Les trois graphes que nous venons de présenter utilisent différents types d'arêtes conditionnelles pour indiquer des informations de trois sortes :

- 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 liées à un choix précédent : 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.

Le Tableau 9 récapitule la signification des conditions liées à tous les types d'arêtes conditionnelles.

Information		Notation	Signification
Choix	de décomposition fonctionnelle	C_{Fi}	Si C_{Fi} est vraie, le chemin suivi correspond à la décomposition fonctionnelle i.
	de composant	C_{Ci}	Si C_{ci} est vraie, le chemin suivi correspond au choix du composant i.
	de localisation	C_{Pi}	Si C_{Pi} est vraie, le chemin suivi correspond au choix de la localisation i.
Obligation	de rôle	O_{Ri}	Si O_{Ri} est vraie, le chemin suivi précédemment (rôle i) impose le choix du rôle en aval de l'arête.
	de composant	O_{Ci}	Si O_{Ci} est vraie, le chemin suivi précédemment (composant i) impose le choix du composant en aval de l'arête.
Identification	de Groupe	I_{Gi}	Si I_{Gi} est vraie, le chemin suivi décrit le Groupe i.
	de Sous-Groupe	I_{SGi}	Si I_{SGi} est vraie, le chemin suivi décrit le Sous-Groupe i.

Tableau 9 : Signification des arêtes conditionnelles

2.3.3. Représentation de la qualité de service d'une application

Pour modéliser l'évaluation de la QdS, nous utilisons le même type de graphes que ceux que nous venons de présenter. Cette similitude de forme nous permet de définir la QdS d'une application modélisée selon nos propositions (cf. §2.2 p.99) à l'aide de notre modèle de QdS (cf. §2.1 p.83). Ce sont les méthodes d'évaluation du temps d'exécution de

systèmes embarqués distribués (cf. §1.4.2 p.72) qui nous ont inspirés. Le résultat est ensuite représenté par des notes dont nous donnons les dénominations et les règles d'obtention.

2.3.3.1. 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 multimédias et du modèle de QdS. Puis nous décrirons les graphes d'évaluation ainsi que leurs propriétés.

a. Graphe et vecteur de qualité de service

La QdS d'une application (cf. §2.1.2.2 p.85) se caractérise par deux critères : le critère intrinsèque regroupant les caractéristiques de qualité indépendantes du contexte et le critère contextuel regroupant celles dépendant du contexte. Evaluer la QdS d'une application consiste alors à évaluer ces deux critères. Il s'agit donc de définir les caractéristiques de chaque critère puis de les évaluer. Or les caractéristiques de l'application regroupent celles des services — les Groupes — et donc 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 de l'application elle-même. Nous proposons de structurer les caractéristiques de QdS d'un Sous-Groupe en un **vecteur de QdS** [LIB 01] (cf. §1.1.2.2.b p.39). Ce vecteur contiendra, d'une part les caractéristiques intrinsèques et d'autre part les caractéristiques contextuelles.

Rappelons que l'utilisateur perçoit le service au travers du composant observable (cf. §2.2.3.3 p.105) de sorte que les caractéristiques du Sous-Groupe sont celles du composant observable. Outre le contexte d'exécution, elles dépendent des flux reçus par ce composant et des spécificités de ce composant. Ces spécificités sont définies par son concepteur. Les flux reçus sont ceux émis par les composants en amont. Ils ont été transmis par un ou des Conduits qui en ont éventuellement modifié les caractéristiques. Les caractéristiques de QdS des composants en amont dépendent, elles aussi, des flux reçus et des spécificités de ces composants. 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.

De plus, les composants logiciels sont encapsulés dans des Processeurs Élémentaires tandis que les flux sont véhiculés par des Conduits. Ces deux entités en assurent en particulier la synchronisation. Si bien que la QdS d'un composant synchronisé avec un flux peut être différente de celle obtenue par ce même composant sans synchronisation : si la cadence d'un composant de traitement de son est plus élevée que celle des images synchronisées, le composant de traitement aura une qualité plus faible qu'en absence de synchronisation. C'est pourquoi les Processeurs Élémentaires et les Conduits influent sur la QdS. Chacun correspond donc, du point de vue de la QdS, à une modification apportée au vecteur QdS qui peut être modélisée par une fonction mathématique ou un algorithme de calcul.

L'application étant représentée par un graphe dont les nœuds sont les Processeurs Élémentaires et les Conduits, la détermination des caractéristiques de QdS sera représentée par le graphe isomorphe où les nœuds indiquent les modifications que les Processeurs Élémentaires et les Conduits imposent au vecteur de QdS représenté par les arêtes. Les

composants non informatiques et les connexions matérielles correspondront également à des nœuds. Nous obtenons alors un graphe d'évaluation de la QdS.

b. Graphes d'évaluation

Les graphes d'évaluation de la QdS sont des graphes orientés et polaires $G(V, E_s, E_c)$ respectant les principes exposés au paragraphe 2.3.1.2.

Les arêtes représentent 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 43). Nous utilisons ici une représentation très proche de celle proposée par [LIB 01] (cf. §1.1.2.2.b p.39). 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 43). 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 paramètre [WAN 96] (cf. §1.1.2.1.c p.37). Cependant le fait d'isoler l'influence de chaque nœud nous permet de ne pas nous limiter à des règles générales comme les règles additives, multiplicatives ou concaves. Ainsi, 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.

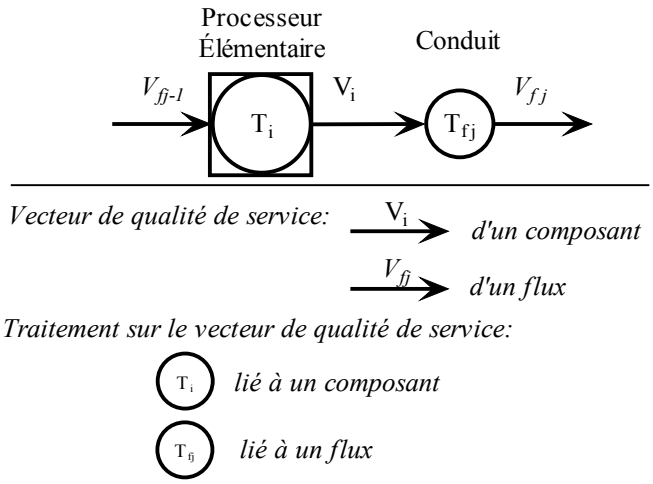


Figure 43 : Qualité de service d'un composant et d'un Conduit

Finalement la QdS du Sous-Groupe, qui est représentée par le vecteur de QdS en sortie du composant observable, peut être considérée comme le résultat de l'application des traitements représentés par les nœuds sur le ou les vecteurs de QdS en entrée du premier composant du Sous-Groupe c'est-à-dire en sortie du nœud fictif source. Déterminer la QdS nécessite tout d'abord de définir le premier vecteur de QdS et surtout sa composition. Dans

le cas des applications multimédias, il contiendra au moins le temps de traitement et le débit car ces caractéristiques sont critiques pour les applications multimédias. 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 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 42 page 126. La Figure 44 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.

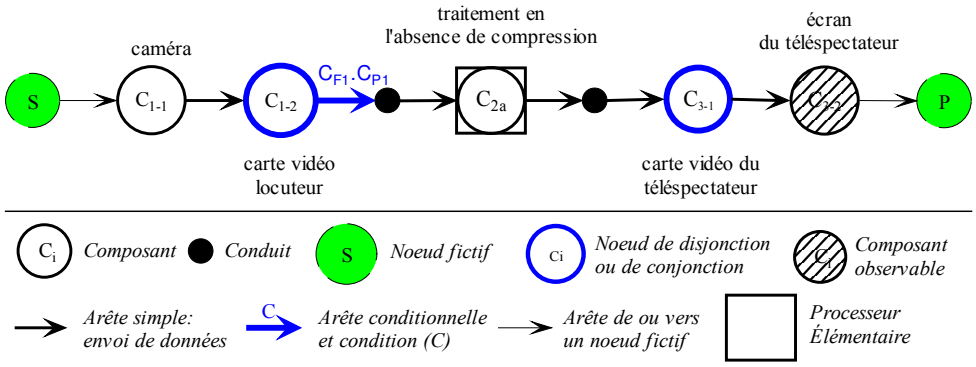


Figure 44 : Exemple de configuration du Sous-Groupe Image à évaluer

La Figure 45 décrit l'évaluation de la QdS de la configuration $L_1=C_{F1}S_1$ du Sous-Groupe Image soulignant l'isomorphisme entre les deux représentations.

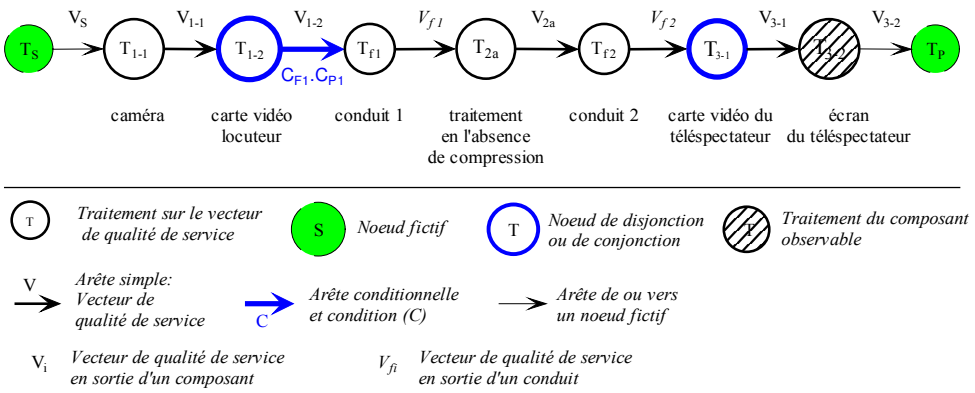


Figure 45 : Exemple de Graphe d'évaluation de la qualité de service du Sous-Groupe Image

Enfin, la Figure 46 page 131 illustre la définition des valeurs des différents vecteurs ainsi que les traitements qui permettent de définir la QdS du Sous-Groupe Image. En fonction de leurs caractéristiques, les composants et les Conduits influent sur une ou plusieurs caractéristiques de QdS. 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.

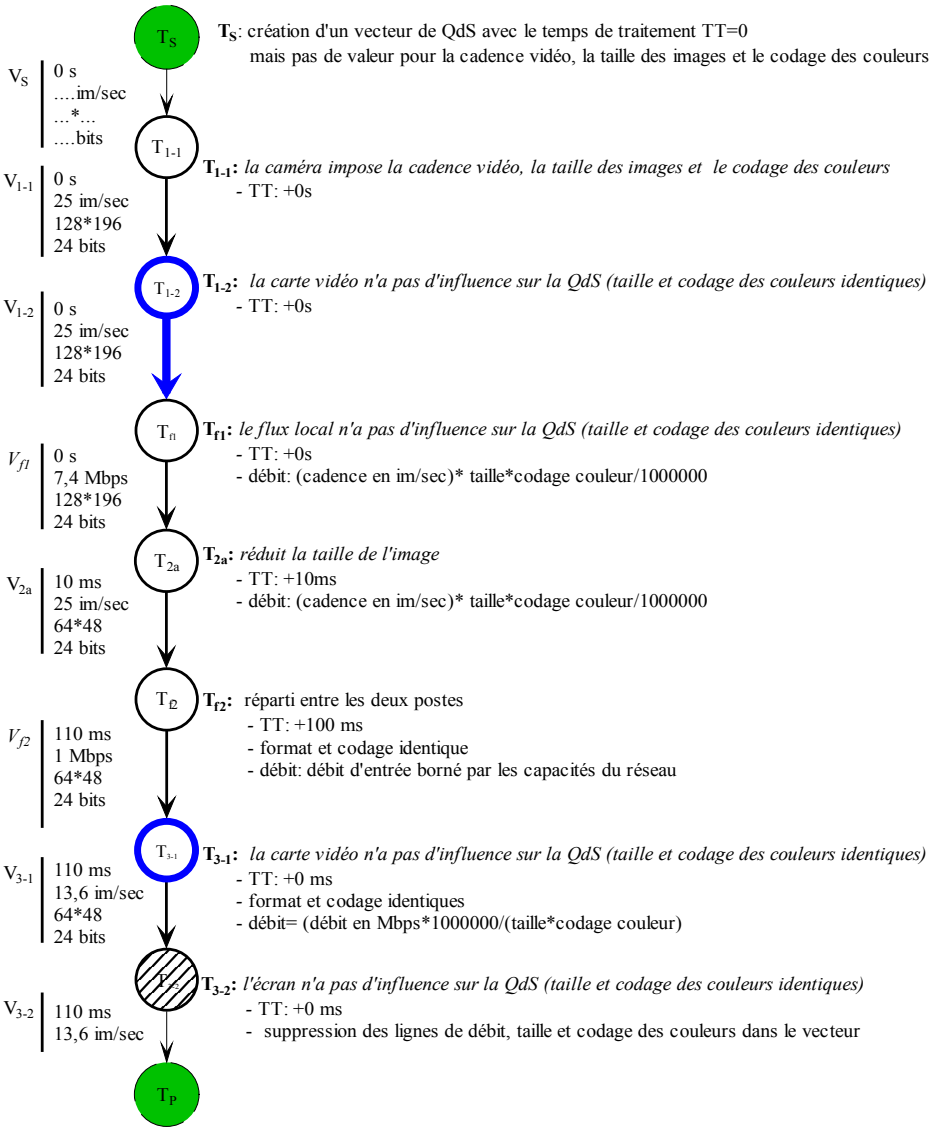


Figure 46 : Exemple d'évaluation du Sous-Groupe Image

Nous notons que ces figures identifient les nœuds de disjonction et de conjonction car ceux-ci permettent d'optimiser l'évaluation de la QdS comme nous allons le voir.

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

Le graphe d'évaluation représente, à l'aide d'un symbolisme particulier, les nœuds de conjonction et de disjonction car ils permettent d'identifier les parties communes à différentes configurations. Ce graphe dérive du graphe de configuration et ces nœuds 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 ou les traitements du graphe. Ainsi l'évaluation peut être optimisée et, avec elle, les actions de la plate-forme.

La partie du graphe comprise entre la source et le premier successeur de la source qui est un nœud de disjonction est commune à toutes les configurations (cf. Figure 47) : les traitements sont identiques et le vecteur QdS en sortie du nœud de disjonction est identique.

La partie du graphe comprise entre le nœud puits et le premier de ses prédécesseurs qui soit un nœud de conjonction est commune à toutes les configurations (cf. Figure 47) : les traitements sont identiques mais les vecteurs de QdS de chaque configuration ne sont, a priori, pas identiques car le vecteur de QdS en entrée de la partie du graphe considérée n'est pas le même pour toutes les configurations.

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 47 reprend l'exemple de la Figure 45 page 130 et identifie les parties d'évaluation communes à toutes les configurations soit uniquement par leurs traitements soit également par leurs vecteurs.

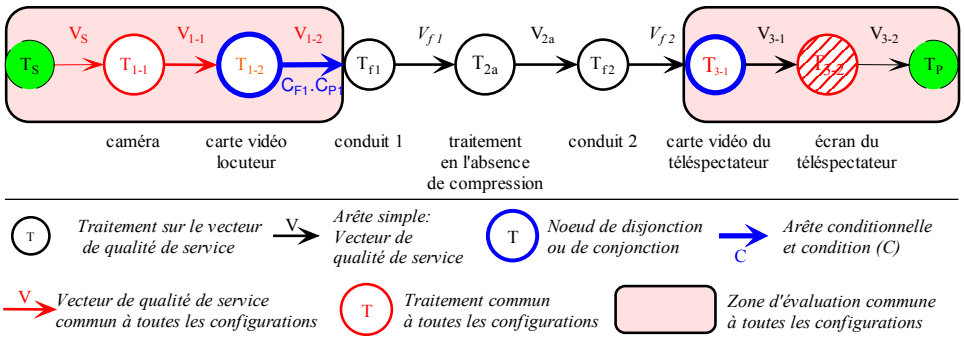


Figure 47 : Exemple d'évaluations communes à toutes les configurations

Etudions maintenant la configuration définie sur la Figure 42 page 126 par $L_2=C_{F2}C_{C4a}P_1$ c'est-à-dire la configuration du Sous-Groupe Image contenant des composants de compression et le composant de traitement sur le poste 1. L'évaluation de cette configuration est représentée par la Figure 48 page 133. Certaines parties de l'évaluation sont communes à toutes les configurations et d'autres à quelques-unes. Une telle étude permet donc d'optimiser l'évaluation en ne recalculant ni les traitements ni les vecteurs communs.

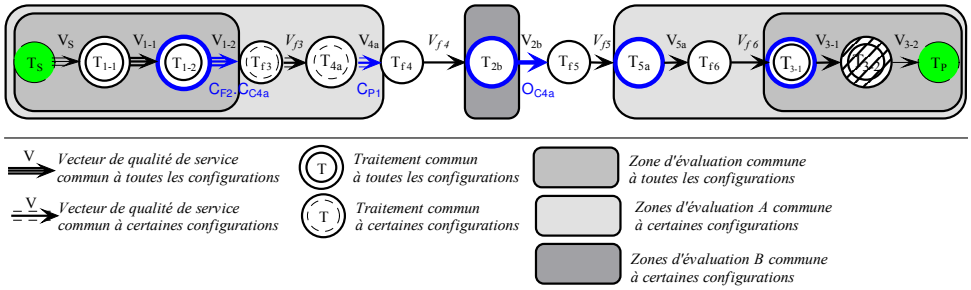


Figure 48 : Exemple d'évaluations communes à certaines ou à toutes les configurations

2.3.3.2. 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.

Soucieux de conserver toute sa généralité à notre démarche, nous représentons toutes les notes possibles alors qu'en pratique la granularité de leur définition dépendra des choix de qualité que propose l'application et du temps que l'utilisateur pourra accorder à la spécification de ses préférences. Ainsi, dans la plupart des applications multimédias réparties, la granularité des notes utilisées sera plus grande que le plus petit niveau utilisé ici, c'est-à-dire les caractéristiques de service d'un Sous-Groupe. Nous ne pouvons cependant pas nous dispenser de les représenter sous peine de restreindre arbitrairement notre domaine de recherche.

a. Nomenclature des notes de qualité de service

Nous avons défini précédemment (cf. §2.1.2.4 p.88) les notes In et Co comme les notes des critères intrinsèque et contextuel de l'application utilisées pour définir la QdS de l'application. Dans ce but, il est nécessaire de définir les notes des Groupes et Sous-Groupes. Le schéma présenté Figure 49 (p.134) indique la notation que nous utilisons par la suite pour représenter l'ensemble des notes. Pour chaque note, nous précisons entre parenthèse 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 parenthèse indique que cette note dépend de la configuration i étudiée.

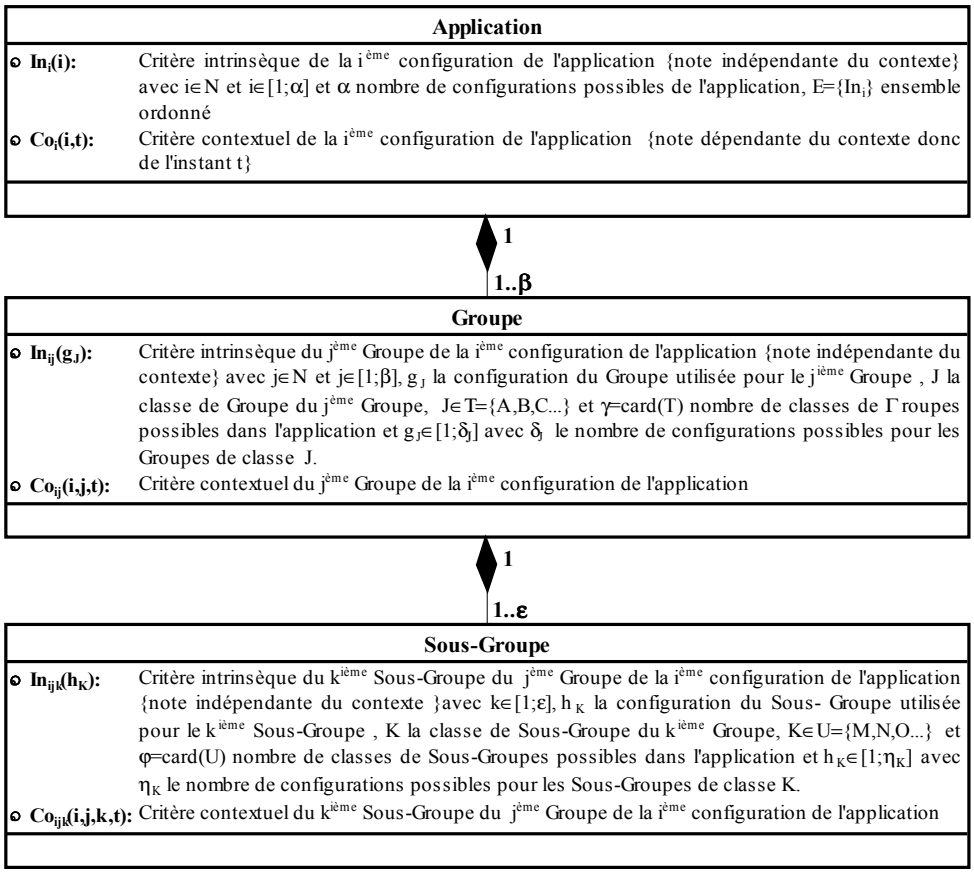


Figure 49 : Notes de qualité de service des constituants de l'application

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.

b. Qualité de service d'un Sous-Groupe

Les graphes d'évaluation (cf. Figure 46 p.131) permettent de définir les caractéristiques de la fonctionnalité fournie par un Sous-Groupe. En comparant ces caractéristiques avec celles désirées par l'utilisateur, il est possible d'attribuer une note à chacune puis de définir la note du Sous-Groupe pour chacun des critères grâce aux poids attribués par l'utilisateur aux différentes caractéristiques. Ces poids, p_l et q_l, lui permettent d'indiquer les caractéristiques de service qui ont le plus d'importance pour lui. Pour un Sous-Groupe donné, le nombre λ de caractéristiques intrinsèques peut être différent du nombre μ de caractéristiques contextuelles. De plus, tous les deux différent d'un Sous-Groupe à l'autre.

Ainsi la note du critère intrinsèque du k^{ième} Sous-Groupe du j^{ème} Groupe de la i^{ème} configuration représentée par In_{ijk} est une moyenne pondérée — par l'utilisateur — des notes m_l des caractéristiques intrinsèques de ce Sous-Groupe implantant la configuration h_K :

Formule 5 : $In_{ijk} = \dfrac{\sum_{l=1}^{\lambda_{h_K}} p_l . m_l}{\sum_{l=1}^{\lambda_{h_K}} p_l}$ **avec** $\begin{cases} \lambda_{h_K} & \text{le nombre de caractéristiques du critère intrinsèque du } k^{\text{ième}} \text{ Sous-Groupe} \\ m_l & \text{la note de la } l^{\text{ème}} \text{ caractéristique} \\ p_l & \text{le poids de la } l^{\text{ème}} \text{ caractéristique} \end{cases}$

La note du critère contextuel du $k^{ième}$ Sous-Groupe du $j^{ème}$ Groupe de la $i^{ème}$ configuration représentée par Co_{ijk} est une moyenne pondérée — par l'utilisateur — des notes n_l des caractéristiques contextuelles de ce Sous-Groupe :

Formule 6 : $Co_{ijk} = \dfrac{\sum_{l=1}^{\mu_{h_K}} q_l . n_l}{\sum_{l=1}^{\mu_{h_K}} q_l}$ **avec** $\begin{cases} \mu_{h_K} & \text{le nombre de caractéristiques du critère contextuel du } k^{\text{ième}} \text{ Sous-Groupe} \\ n_l & \text{la note de la } l^{\text{ème}} \text{ caractéristique} \\ q_l & \text{le poids de la } l^{\text{ème}} \text{ caractéristique} \end{cases}$

c. Qualité de service d'un Groupe

Les Sous-Groupes sont assemblés en Groupes. Les notes obtenues pour les Sous-Groupes permettent donc de calculer celles des Groupes.

Ainsi la note du critère intrinsèque du $j^{ème}$ Groupe de la $i^{ème}$ configuration représentée par In_{ij} est une moyenne pondérée des notes In_{ijk} des critères intrinsèques des Sous-Groupes le constituant. Ce Groupe met en œuvre la configuration g_J. Sa note intrinsèque est donnée par la formule suivante :

Formule 7 : $In_{ij} = \dfrac{\sum_{k=1}^{\varepsilon_{g_J}} r_k . In_{ijk}}{\sum_{k=1}^{\varepsilon_{g_J}} r_k}$ **avec** $\begin{cases} \varepsilon_{g_J} & \text{le nombre de Sous-Groupes du } j^{\text{ième}} \text{ Groupe} \\ In_{ijk} & \text{la note de In du } k^{\text{ième}} \text{ Sous-Groupe} \\ r_k & \text{le poids du } k^{\text{ième}} \text{ Sous-Groupe dans ce Groupe} \end{cases}$

La note du critère contextuel du $j^{ème}$ Groupe de la $i^{ème}$ configuration représentée par Co_{ij} est une moyenne pondérée des notes Co_{ijk} des critères intrinsèques des Sous-Groupes le constituant. C'est l'utilisateur qui indique les poids des notes de QdS des Sous-Groupes utilisés dans le calcul des notes d'un Groupe. Or il ignore si les caractéristiques de QdS sont contextuelles ou intrinsèques donc ces poids sont les mêmes pour le calcul de la note intrinsèque et celui de la note contextuelle du Groupe. La note contextuelle est donnée par la formule suivante :

Formule 8 : $Co_{ij} = \dfrac{\sum_{k=1}^{\varepsilon_{g_J}} r_k . Co_{ijk}}{\sum_{k=1}^{\varepsilon_{g_J}} r_k}$ **avec** $\begin{cases} \varepsilon_{g_J} & \text{le nombre de Sous-Groupes du } j^{\text{ième}} \text{ Groupe} \\ Co_{ijk} & \text{la note de Co du } k^{\text{ième}} \text{ Sous-Groupe} \\ r_k & \text{le poids du } k^{\text{ième}} \text{ Sous-Groupe dans ce Groupe} \end{cases}$

d. Qualité de service d'une application

La QdS d'une configuration i de l'application est définie (cf. §2.1.3.4.a p.95) comme la note minimale entre les notes des critères contextuel Co_i et intrinsèque In_i de l'application soit par :

$$QdS_i\left(Co_i, In_i\right) = \min(Co_i, In_i).$$

Or l'application est constituée par un assemblage de Groupes. La note du critère intrinsèque de la configuration i de l'application représentée par In_i est une moyenne non pondérée des notes In_{ij} des critères intrinsèques des Groupes la constituant. La moyenne est non pondérée car chaque utilisateur a la même importance vis-à-vis de l'application c'est-à-dire que nous faisons l'hypothèse qu'il n'y a pas d'utilisateur privilégié. La note intrinsèque de la configuration i de l'application est alors donnée par la formule suivante :

Formule 9 : $In_i = \dfrac{\sum_{j=1}^{\beta_i} In_{ij}}{\beta_i}$ **avec** $\begin{cases} \beta_i & \text{le nombre de Groupes} \\ In_{ij} & \text{la note de In du } j^{ème} \text{ Groupe} \end{cases}$

De même, la note du critère contextuel de la configuration i de l'application représentée par Co_i est une moyenne non pondérée des notes Co_{ij} des critères contextuels des Groupes la constituant :

Formule 10 : $Co_i = \dfrac{\sum_{j=1}^{\beta_i} Co_{ij}}{\beta_i}$ **avec** $\begin{cases} \beta_i & \text{le nombre de Groupes} \\ Co_{ij} & \text{la note de Co du } j^{ème} \text{ Groupe} \end{cases}$

Avec la présentation du système d'évaluation de la QdS, nous achevons de décrire notre modélisation des applications et de leur QdS. La plate-forme d'exécution peut être utilisée uniquement avec des applications respectant ces modèles. C'est pourquoi nous proposons une méthode de conception d'applications multimédias réparties qui assure que celles-ci puissent être supervisées par notre plate-forme.

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

Notre objectif est de proposer une architecture logicielle pour intégrer la qualité de service dans les applications multimédias réparties sur Internet. Nous avons choisi d'utiliser une plate-forme d'exécution qui utilise notre modèle de qualité de service et notre modèle d'application pour superviser l'application et en optimiser la qualité de service. Pour concevoir ce système, nous avons dû faire des hypothèses sur la phase de conception de l'application et en particulier sur les informations que la conception permettra de fournir à la plate-forme d'exécution. Ces hypothèses sont les prémices de futurs travaux sur un outil

d'aide à la conception qui sera intégré dans notre plate-forme. C'est à ce titre que nous présentons notre méthode de conception inhérente à la plate-forme.

Elle peut être représentée par une suite d'étapes permettant tout d'abord la définition de la structure de l'application en passant par la spécification des contraintes fonctionnelles, des contraintes de synchronisation et des composants disponibles, puis permettant 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.

2.3.4.1. Phase 1 : Définition de la structure

Nous précisons tout d'abord les différentes étapes de la définition de la structure avant d'étudier plus précisément la méthode de construction des différents graphes.

a. Différentes étapes de la conception

Tout d'abord, le concepteur définit les types de services que l'application devra fournir. Il identifie les classes de Groupes et les classes de Sous-Groupes représentant les fonctionnalités des services (cf. Tableau 10 p.138). Ces informations permettent de spécifier la composition des configurations canoniques de l'application et des Groupes ainsi que les différentes configurations possibles de Groupes en terme de Sous-Groupes. En outre, au fur et à mesure de la définition des Groupes et des Sous-Groupes, il indique quels sont les flux qui devront être synchronisés. Le cas échéant, le concepteur de l'application pourra spécifier pour chaque classe de Groupe une configuration matérielle minimale comme, par exemple, l'utilisation d'un composant spécifique à ce Groupe tel un microphone.

Poursuivant la décomposition fonctionnelle à l'aide de notre modèle, le concepteur spécifie le graphe des flots de contrôle. Il peut alors identifier les rôles des composants nécessaires à la réalisation des Sous-Groupes. Il formalise cela par un graphe fonctionnel des Sous-Groupes non achevé puisqu'il ne peut pas encore définir les rôles qui seront atomiques. Ce graphe lui permet de spécifier les flux de données échangés par les composants ou par les assemblages de composants

Puis, le concepteur réalise l'inventaire des composants dont il dispose a priori et ceux qu'il va devoir développer ou acquérir. Au fur et à mesure de l'établissement de la liste des composants disponibles, le concepteur compare les rôles dont il a besoin et ceux réalisés par les composants. Il précise également la nature matérielle ou logicielle, les caractéristiques, les performances et les contraintes d'utilisation, telles les compatibilités et incompatibilités, de chaque composant.

Il peut alors finaliser le graphe fonctionnel des Sous-Groupes en utilisant exclusivement des rôles atomiques.

La structure de l'application est alors entièrement définie et nous en résumons les étapes dans le Tableau 10.

ÉTAPES		
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 10 : Définition de la structure de l'application

Pour illustrer notre méthode, nous décrivons une des façons dont un concepteur peut construire l'exemple de vidéoconférence présenté en 2.1.2.1. En effet, en fonction des composants dont il dispose et des différentes qualités de service qu'il souhaite proposer aux utilisateurs, celui-ci pourra élaborer différents logiciels.

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

- le Groupe Locuteur utilisé pour les deux intervenants de la conférence qui devront disposer d'une caméra motorisée et d'un microphone sur les deux postes concernés — poste 1 et poste 2 ;
- le Groupe Téléspectateur utilisé sur le poste 3 — muni d'un téléphone portable— et le poste 4 — équipé d'un ordinateur personnel — par des utilisateurs souhaitant assister aux débats.

Le concepteur peut donc définir la configuration canonique de l'application en terme de Groupe (cf. Figure 50).

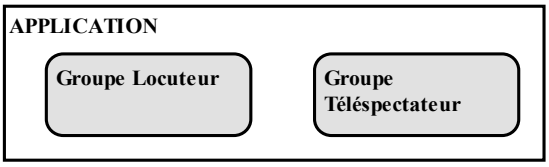


Figure 50 : Configuration canonique de l'application de vidéoconférence

Puis la deuxième étape lui 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, pour le premier, d'un Sous-Groupe Son et d'un Sous-Groupe Image et pour le second 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 51 et Figure 52) ainsi que les différents graphes des configurations en terme de Sous-Groupes.

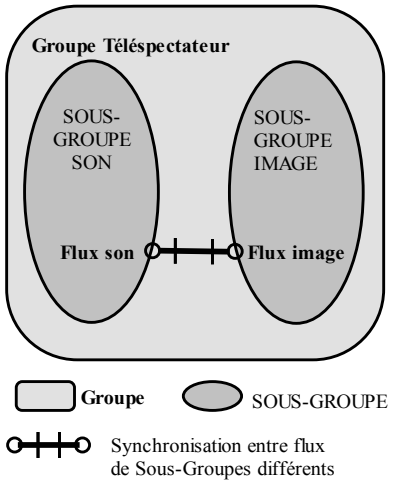


Figure 51 : Configuration canonique du Groupe Téléspectateur

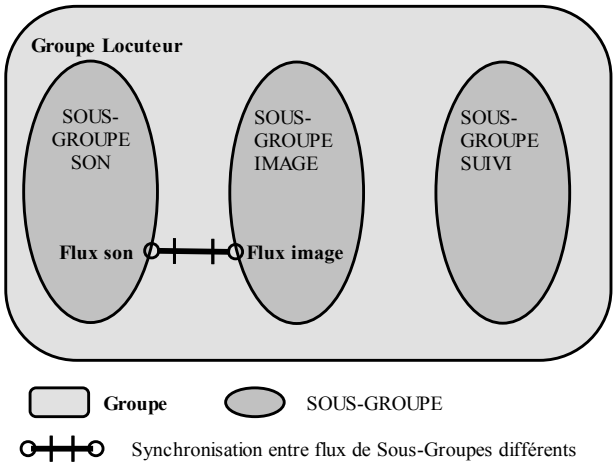


Figure 52 : Configuration canonique du Groupe Locuteur

b. 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 53) (cf. Figure 54) (cf. Figure 55). 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.

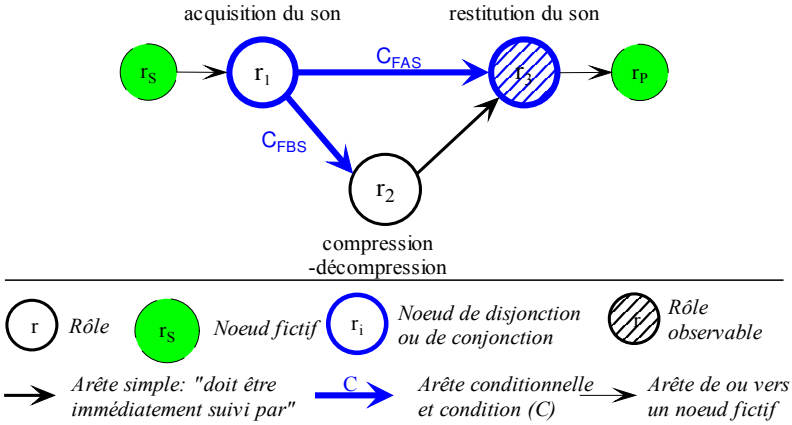


Figure 53 : Graphe des flots de contrôle du Sous-Groupe Son

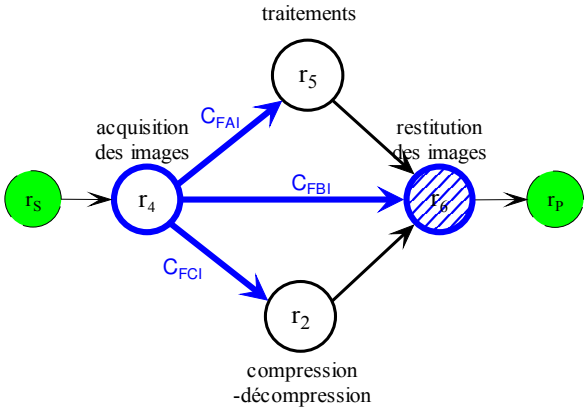


Figure 54 : Graphe des flots de contrôle du Sous-Groupe Image

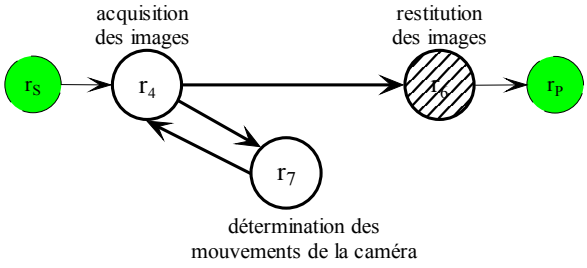


Figure 55 : Graphe des flots de contrôle du Sous-Groupe Suivi

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, un ou plusieurs flux de données ou par une ou des contraintes de synchronisation. 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 les réaliser par un assemblage de composants. Lorsque tous les rôles du graphe sont atomiques, le graphe est appelé graphe fonctionnel (cf. Figure 56) (cf. Figure 57) (cf. Figure 58) 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.

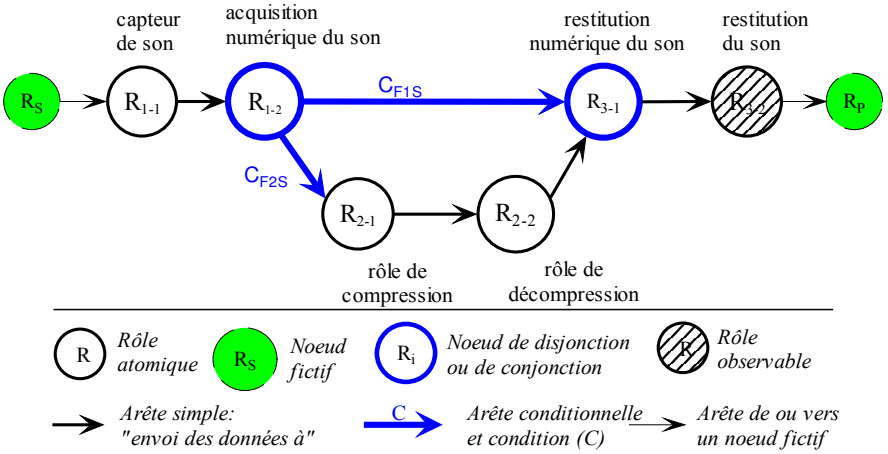


Figure 56 : Graphe fonctionnel du Sous-Groupe Son

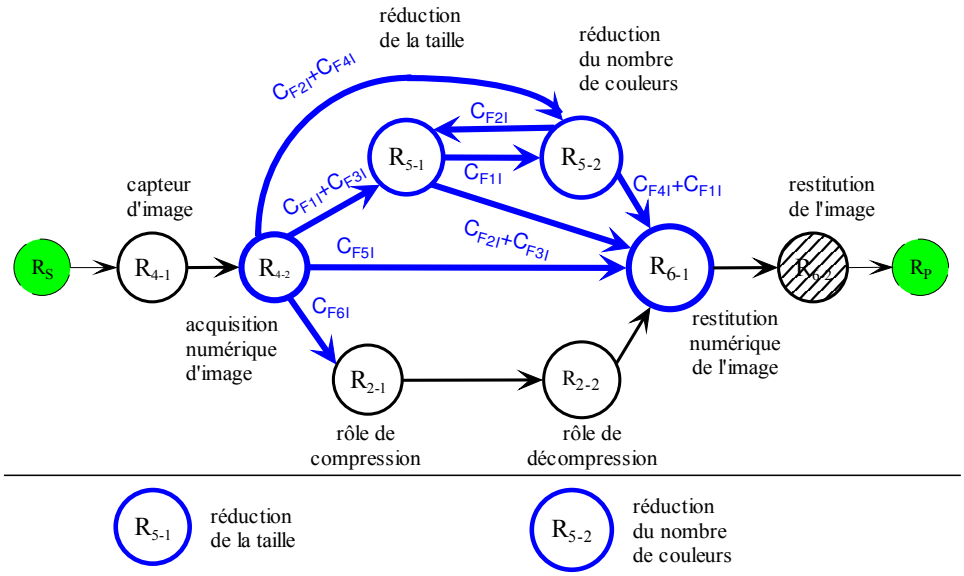


Figure 57 : Graphe fonctionnel du Sous-Groupe Image

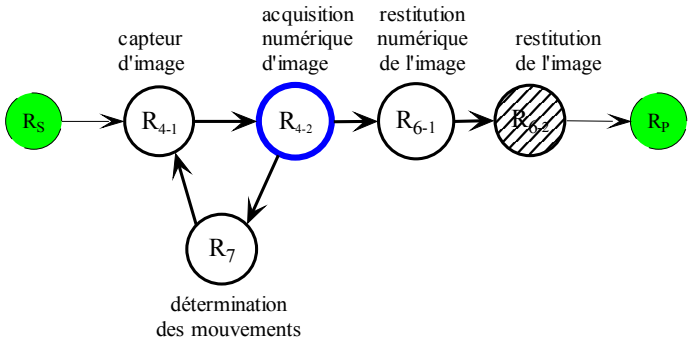


Figure 58 : Graphe fonctionnel du Sous-Groupe Suivi

2.3.4.2. Phase 2 : Définition des différentes configurations de l'application

Elle consiste à définir le graphe canonique des configurations de l'application à partir de la configuration canonique de l'application en terme de Groupes, de graphes des configurations des Groupes en terme de Sous-Groupes et de graphes fonctionnels des Sous-Groupes.

a. Transmission et synchronisation

Il reste maintenant à définir quels flux doivent transiter par des Conduits ou des Processeurs Élémentaires.

Pour des raisons de synchronisation, tout flux d'information est transmis grâce à un Conduit (cf. §2.2.5.1 p.111). Il est donc possible de définir des règles de réécriture du graphe fonctionnel qui permettent de savoir comment regrouper les flux d'informations dans des Conduits. En effet, deux flux synchronisés doivent transiter par un même Conduit. 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 59 page 143. Lorsque deux flux du graphe fonctionnel sont synchronisés, les flux correspondant du graphe de configuration appartiennent à un même Conduit. Si un de ces flux doit être traité par un rôle du graphe fonctionnel, les flux correspondant du graphe de configuration transitent par le Processeur Élémentaire encapsulant le composant implantant ce rôle. Remarquons que les Conduits et les Processeurs Élémentaires assurent la synchronisation d'un nombre quelconque de flux.

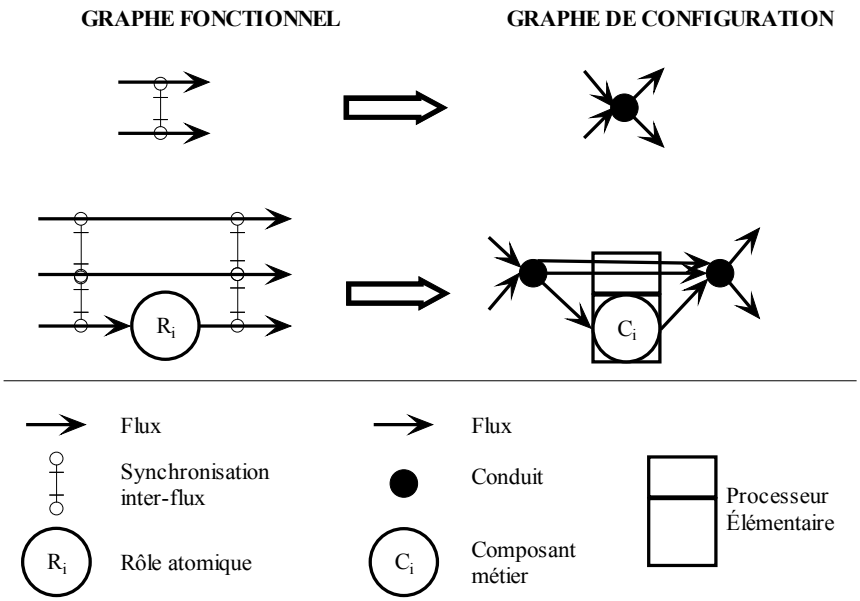


Figure 59 : Règles d'insertion des Conduits et des Processeurs Élémentaires pour les flux informatiques

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 60 p.144) qui réunit la décomposition C_{F1S} du Sous-Groupe Son (cf. Figure 56 p.141) et la décomposition C_{F3I} du Sous-Groupe Image (cf. Figure 57 p.141) pour construire le graphe de l'une des configurations correspondantes (cf. Figure 61 p.144).

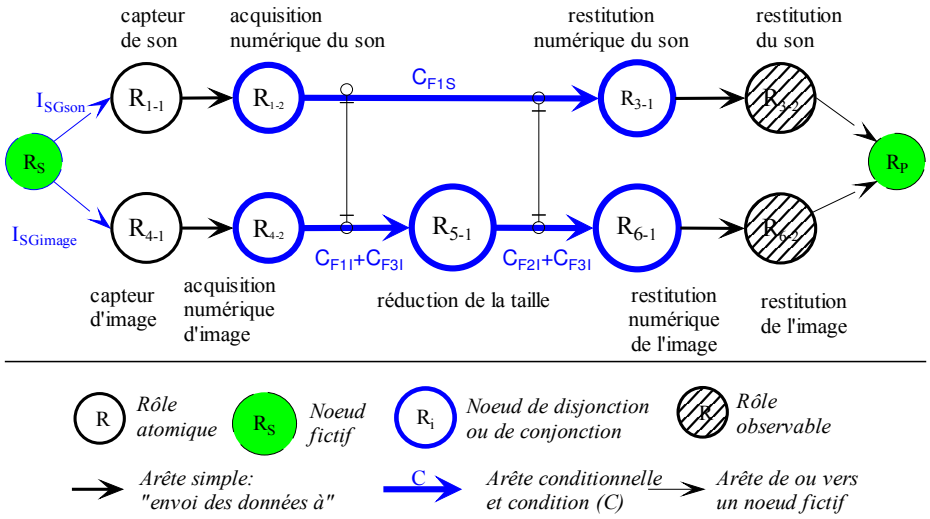


Figure 60 : Graphe fonctionnel d'une décomposition du Groupe Téléspectateur

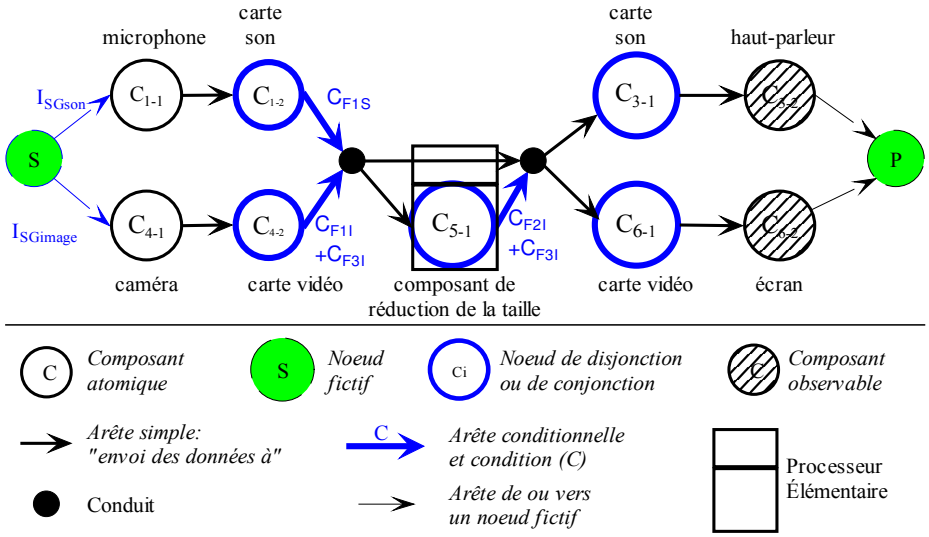


Figure 61 : Graphe d'une configuration du Groupe Téléspectateur

b. 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 une arête conditionnelle si plusieurs composants peuvent le réaliser. Pour parcourir le graphe, il est alors essentiel de respecter l'orientation indiquée par les arêtes car l'utilisation d'un composant pour un rôle peut imposer l'utilisation d'un

autre en aval. Cette information sera précisée par une arête conditionnelle (cf. §2.3.2.3 p.127). 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 62) décrit ainsi plusieurs configurations pour la décomposition fonctionnelle C_{F1I} (cf. Figure 57 p.36) 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 — un pour chaque rôle.

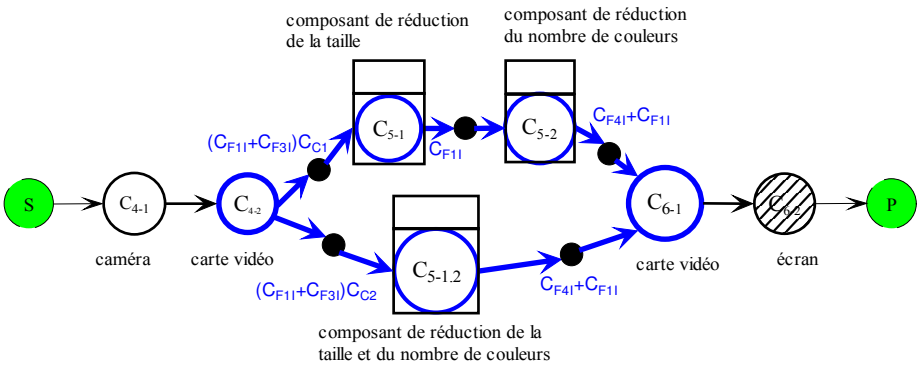


Figure 62 : Super-graphe partiel des configurations du Sous-Groupe Image

c. Récapitulatif de la phase 2

Le Tableau 11 résume les différentes étapes de la phase de définition des configurations de l'application. 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 terme de composants à l'aide des arêtes conditionnelles. A ce stade de la conception, l'implantation n'est que partiellement définie puisque les différents postes disponibles ne sont pas connus.

	ÉTAPES	
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 11 : Définition des différentes configurations de l'application

La conception de l'application doit être réalisée en totale cohérence avec les modèles et les représentations utilisés par notre plate-forme. Et inversement, la conception de la plate-forme nécessite d'avoir défini au préalable les informations dont elle disposera. C'est

pourquoi nous en avons esquissé ici les grandes lignes. Un outil d'aide à la conception pourrait être envisagé mais ce n'est pas l'objet de ce travail.

La plate-forme disposera donc de la définition de toutes les configurations possibles de l'application sous la forme d'un graphe canonique des configurations. Elle connaît donc les services disponibles pour les utilisateurs – les Groupes. Pour chaque service, elle connaît les différentes associations de fonctionnalités pouvant le réaliser — les Sous-Groupes. Pour chaque configuration d'un service, elle a les moyens de définir les différents assemblages de composants utilisables. Ce sont toutes ces informations que notre modèle de plate-forme pourra utiliser pour gérer la qualité de service des applications multimédias réparties.

2.3.5. Synthèse

Nous choisissons de représenter l'évaluation de la QdS de la même manière que les différents aspects de l'application c'est-à-dire sous la forme de graphes orientés polaires. 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.

2.4. Conclusion

Guidés par notre volonté de proposer une gestion de la qualité de service adaptée aux applications multimédias sur Internet, nous présentons un modèle de qualité de service centré sur l'utilisateur. La QdS reflète donc la satisfaction de l'utilisateur. Elle se décompose en deux niveaux hiérarchiques : les caractéristiques et les critères. Ces derniers distinguent les caractéristiques qui dépendent du contexte de celles qui en sont indépendantes.

D'autre part, l'importance primordiale de la perception par l'utilisateur est l'une des caractéristiques des applications multimédias réparties. C'est pourquoi nous modélisons l'application en fonction du service fourni à l'utilisateur et de deux niveaux, le Groupe et le Sous-Groupe, représentant un service et ses fonctionnalités. Ces dernières sont obtenues par un assemblage de composants, logiciels, matériels ou humains, reliés par des flux qui sont véhiculés par des Conduits alors que les composants logiciels sont encapsulés dans des Processeurs Élémentaires.

En effet, nous faisons le choix de poser la synchronisation des flux comme une condition *sine qua non* à leur transmission. Les Conduits et les Processeurs Élémentaires permettent alors de réaliser cette synchronisation. Cette optique se justifie d'une part, par les caractéristiques des applications multimédias où les différents flux, en particulier le son et l'image, s'unissent pour créer la sémantique des informations et d'autre part, par les contraintes de développement pour le grand public où la conception doit être simplifiée pour réduire les coûts. Ceci passe souvent par une gestion séparée des contraintes non fonctionnelles.

Enfin, nous proposons une représentation unifiée des applications et de leur qualité de service sous forme de graphes orientés polaires. Cette représentation permet de passer de manière isomorphe des spécifications fonctionnelles à l'implantation de l'application puis à l'évaluation de sa qualité de service. Elle permet donc d'automatiser certaines tâches du développement. Par son choix, nous souhaitons ainsi préserver la possibilité de proposer une plate-forme aidant le concepteur dans sa tâche même si ce n'est pas l'objet des travaux présents. En effet, ceux-ci portent sur la plate-forme d'exécution permettant de gérer la qualité de service pendant le fonctionnement de l'application.

Partie 3: Modélisation de la plate-forme

Nous avons défini un modèle de qualité de service, un modèle d'application multimédia répartie et une méthode de conception. Nous connaissons donc les informations dont disposera la plate-forme pour superviser l'application et en optimiser la QdS. Dans cette troisième partie, nous présentons un modèle de plate-forme d'exécution qui prend en charge à la fois le déploiement de l'application, en particulier l'arrivée ou le départ d'un utilisateur, et sa supervision. Nous proposons que la plate-forme reconfigure dynamiquement l'application en adaptant sa composition et son implantation aux variations du contexte d'exécution. Il s'agit donc de définir un assemblage optimum de composants. Or ce problème est connu pour être NP-complet dans le cas général. Nous débutons donc notre présentation par une étude de la complexité algorithmique de notre problème afin d'identifier ce que la plate-forme pourra exécuter et ce qui n'est pas réalisable.

Connaissant ces limites, nous présentons ensuite les principes généraux de notre modèle de plate-forme avant de définir précisément ce modèle et son implantation. Nous proposons alors un prototype qui nous permet de valider l'ensemble de nos modèles et l'intégralité de nos travaux.

3.1. Complexité algorithmique et solutions

Configurer un assemblage optimum de composants est réputé comme un problème NP-complet dans le cas général (cf. §1.4.3.2 p.76). Il est donc nécessaire de connaître la complexité exacte des différents aspects de notre problématique pour proposer une plate-forme réaliste. C'est pourquoi nous étudierons tout d'abord la complexité de l'optimisation de la QdS dans le cas général. Puis nous vérifierons que nos hypothèses de modélisation simplifient cette complexité. Enfin nous évaluerons de façon précise la complexité de la définition des différentes configurations de l'application représentée par notre modèle. L'objectif est que la plate-forme puisse définir l'ensemble des configurations et trouver celle proposant la meilleure QdS pour un contexte donné, tout ceci dans un délai acceptable.

3.1.1. Complexité du problème posé

La fonction de complexité (cf. §1.4.3.1 p.76) mesure « les ressources utilisées par un algorithme » qui sont « caractérisées par la taille de l'instance du problème à traiter » [WOL 91]. Nous faisons l'hypothèse communément admise que « la frontière entre (fonction de) complexité acceptable et inacceptable se situe à la limite entre fonction polynomiale et non polynomiale. » [WOL 91]. Nous considérerons la complexité en temps de calcul notée O et nous ferons une analyse du pire cas c'est-à-dire que nous considérerons que le temps de calcul pour des données de taille n est le temps de calcul maximum pour des données de cette taille. La fonction complexité est alors définie de la manière suivante :

« Une fonction g(n) est dite O(f(n)) s'il existe des constantes c et n_0 telles que
pour tout $n>n_0$ $g(n) \leq c.f(n)$. » [WOL 91].

Nous considérerons comme acceptable toute fonction de complexité polynomiale, c'est-à-dire de la forme $O(n^k)$ avec k fixé.

3.1.1.1. Complexité de l'optimisation de la qualité de service

Cette étude détermine si un programme informatique peut trouver la configuration d'une application qui fournit la meilleure QdS dans un contexte donné lorsque l'application est construite à base de composants logiciels. Dans cet objectif, nous établissons la complexité du problème général avant de définir si l'utilisation de nos modèles et de leurs hypothèses permet de simplifier cette complexité.

a. Problème général

Nous motivons notre étude des pires cas avant de donner la démonstration de la complexité de l'optimisation de la QdS dans des applications construites à base de composants logiciels.

Justification de la démarche

Pour optimiser le service fourni par une application à base de composants, il est nécessaire de réaliser deux choix principaux en complément du choix d'ordonnancement qui est rendu plus marginal par les contraintes fonctionnelles liées au transfert des flux : le choix des composants et le choix de leur localisation. Pour chacun, les différentes alternatives doivent être évaluées en sachant que :

- les composants communiquent ;
- ils doivent respecter des contraintes de précédence laissant une liberté variant avec l'application pour fixer l'ordonnancement ;
- ils peuvent être placés sur des machines hétérogènes sans obligation d'utiliser toutes celles qui sont disponibles ;
- la durée que met un composant pour traiter ses données est variable et dépend des données, du composant et du contexte.

Cependant, pour une application, ces choix sont fortement restreints par le nombre de composants disponibles et leurs incompatibilités, par les composants non délocalisables et enfin par les contraintes fonctionnelles qui limitent les ordonnancements utilisables.

Malheureusement, les contraintes restreignant les choix sont spécifiques à chaque application. Nous ne pouvons donc pas faire l'hypothèse que, pour chaque choix, le nombre de possibilités est restreint sous peine de ne pouvoir proposer une solution valable pour toute application multimédia répartie sur Internet. Nous allons donc étudier, pour chacun, le pire cas du point de vue du nombre de possibilités. Nous restons cependant conscients que les cas rencontrés en pratique seront plus favorables.

Etude des pires cas

Choisir une configuration optimale du point de vue de la QdS peut être vue comme une combinaison de problèmes d'ordonnancement, de choix et de placement de composants.

En effet, l'ordonnancement consiste à définir un ordre relatif d'exécution des traitements et des transmissions. Dans notre cas, lorsque les rôles atomiques à utiliser sont choisis et que les contraintes fonctionnelles sont respectées, choisir une décomposition fonctionnelle revient à ordonner les composants pour lesquels l'ordre n'est pas fixé par les fonctionnalités. Or ce problème est NP-complet (cf. §1.4.3.2.b p.77). La démonstration en a été donnée pour des versions très simples sous le terme d'Ordonnancement Minimum avec Contraintes de Précédence [GAR 79]. Dans le pire cas, choisir une décomposition fonctionnelle optimale est donc un problème non polynomial que la plate-forme ne pourra mener à bien.

D'autre part, choisir un sous-ensemble de k composants parmi un ensemble donné de cardinalité n peut se faire de $C_k^n = \frac{n!}{k!(n-k)!}$ façons (cf. §1.4.3.2.a p.76). Déterminer tous ces sous-ensembles et choisir le meilleur du point de vue de la QdS est donc un problème non polynomial.

Enfin, quand les composants à utiliser pour construire l'application sont connus, choisir le déploiement est un problème de placement dont une version simplifiée a été démontrée comme NP-complète sous l'appellation d'Ordonnancement Multiprocesseur Minimum avec Facteur de Vitesse (cf. §1.4.3.2.b p.77). Le choix d'un déploiement optimal est donc non polynomial. Il est important de le noter car lorsque l'on voudra jouer sur ce critère pour améliorer la QdS, on ne pourra étudier toutes les possibilités. Il sera donc nécessaire de définir un ensemble d'étude à évaluer.

Complexité des pires cas

Choisir une décomposition fonctionnelle, choisir un ensemble de composants et choisir leur placement, sont des problèmes non polynomiaux dans le pire cas. Bien que nous sachions que les contraintes fonctionnelles, les contraintes de précédence, et les contraintes matérielles — compatibilité, localisation — réduiront cette complexité dans les cas concrets, nous serons obligés de proposer une heuristique de recherche de la meilleure configuration qui ne soit pas triviale *i.e.* de type " définir toutes les possibilités, les ordonner et choisir la meilleure ". En particulier, il sera nécessaire de réduire l'ensemble des configurations évaluées lorsque nous rechercherons la meilleure.

D'autre part, puisque notre problème est la combinaison de problèmes qui ont été démontrés comme NP-complets dans l'hypothèse communément admise que P≠NP (cf. §1.4.3.1 p.76) nous pouvons penser qu'il est au moins NP-complet voire plus difficile encore. Nous allons le démontrer.

Démonstration

Nous allons démontrer qu'un cas particulier de notre problème est NP-complet. Notre problème sera donc au moins NP-complet.

Supposons que l'application n'ait qu'un seul utilisateur et que celui-ci dispose de m processeurs identiques. Supposons, encore, que l'application soit constituée de T composants de durée unitaire devant respecter un graphe de précédence. Supposons enfin que la QdS soit uniquement liée au temps de traitement.

Optimiser la QdS revient donc à déterminer l'ordonnancement le plus rapide. Ce problème appelé UET-Scheduling [RAY 87] est NP-complet [CHR 89] [FER 89] [LAG 90] (cf. §1.4.3.2.b p.77) si on peut faire varier m comme dans notre cas ou si on peut faire varier le graphe de précédence et que celui-ci est quelconque — en particulier si ce n'est pas un arbre *i.e.* un graphe connexe acyclique. Or dans le cas général, ce graphe ne sera pas un arbre à cause des composants communs à plusieurs Sous-Groupes.

Notre problème général d'optimisation de la QdS dans les applications multimédias réparties est donc NP complet et non polynomial dans le pire cas.

b. Complexité de l'optimisation dans notre modèle

Nous venons de montrer que déterminer une configuration proposant la meilleure QdS en fonction du contexte est un problème non polynomial dans le cas général. Or notre modélisation des applications et de la QdS met en exergue des contraintes, en particulier de synchronisation, qui pourraient simplifier la complexité. C'est pourquoi nous étudions maintenant la complexité de l'optimisation de la QdS dans notre modèle.

Le problème étudié est celui de trouver la configuration de l'application qui fournira la meilleure QdS dans un contexte donné. Par configuration, nous entendons un ensemble de composants qui réalisent l'application et qui sont répartis de manière spécifique. Trouver la configuration, c'est donc trouver les composants et trouver leur déploiement de manière à ce qu'ils fournissent la meilleure QdS. Or cette QdS est représentée par deux critères intrinsèque et contextuel calculés à partir de moyennes de notes de caractéristiques de QdS définies en parcourant des graphes d'évaluation. Notre problème se ramène donc à trouver un ensemble de chemins élémentaires du graphe d'évaluation qui optimisent la QdS. Ces chemins définissent alors la configuration optimale de l'application.

Tel qu'il est maintenant énoncé, notre problème est identique à celui du routage basé sur des contraintes de qualité de service dans un réseau constitué par le graphe d'évaluation. Ces problèmes sont NP-complets que ce soit dans le cas multicast pour plusieurs métriques additives [KUI 02] ou dans le cas unicast pour un nombre de métriques supérieur à 2 [WAN 96] (cf. §1.4.3.2.a p.76). Ces deux cas particuliers étant NP-complets, notre problème est au moins NP-complet.

Les modèles que nous utilisons pour la QdS et pour les applications ne permettent donc pas de simplifier la complexité de notre problème qui demeure non polynomiale. La plate-forme ne pourra donc pas trouver la meilleure configuration du point de vue de la QdS. Nous allons définir précisément la complexité de l'un des aspects de notre problème : la définition de toutes les configurations utilisables pour l'application qui revient à définir toutes les qualités de service disponibles. Cette étude permet de spécifier le degré de complexité minimal de notre problématique du point de vue du problème de décision. Or l'optimisation est, en général, plus complexe que la décision. Nous aurons ainsi un aperçu plus précis des tâches que la plate-forme ne pourra réaliser ainsi que de l'origine de cette complexité.

3.1.1.2. Évaluation de la complexité de la définition des configurations

Nous allons définir la notation utilisée puis nous majorerons le nombre de configurations possibles pour une application de manière à donner un ordre de grandeur de la complexité de notre problème.

a. Notation

Soit U le nombre maximal d'utilisateurs simultanés de l'application *i.e.* le nombre maximal d'instances de Groupes à un instant donné.

Soit P le nombre maximal de postes *i.e.* le nombre maximal de terminaux informatiques utilisables par l'application. Ce nombre peut être supérieur à U puisqu'il est possible d'avoir des postes sans utilisateur final comme dans le cas d'un traducteur — qui est

considéré comme un composant et non pas comme un utilisateur — ou dans le cas d'un serveur de données. Si plusieurs utilisateurs se partagent le même poste, le même terminal informatique, ils sont modélisés par le même Groupe et la plate-forme les considère comme un seul utilisateur.

Soit *H* le nombre maximal d'assemblages différents de Sous-Groupes qui peuvent entrer dans la composition d'un Groupe donné.

Soit *G* le nombre maximal d'instances de Sous-Groupes que peut contenir une instance de Groupe.

Soit *F* le nombre maximal de décompositions fonctionnelles possibles pour un Sous-Groupe.

Soit *R* le nombre maximal de rôles atomiques pouvant intervenir dans la décomposition fonctionnelle d'un Sous-Groupe.

Soit *C* le nombre maximal de composants réalisant un rôle atomique donné. A priori, si le concepteur propose d'utiliser différents composants pour un même rôle, c'est qu'ils proposent différentes caractéristiques de service.

b. Majoration du nombre de configurations

Le nombre de possibilités de déploiement d'un rôle qui peut être réalisé par *C* composants et implanté sur *P* postes est donné par : $C.P$.

Le nombre de possibilités de déploiement d'une décomposition fonctionnelle qui est constituée d'un assemblage de *R* rôles atomiques est donné par : $(C.P)^R$.

Le nombre de possibilités de déploiement d'un Sous-Groupe qui peut être réalisé par *F* décompositions fonctionnelles différentes est donné par : $(C.P)^R F$.

Le nombre de possibilités de déploiement d'un assemblage de *G* Sous-Groupes réalisant un Groupe est donné par : $\left[(C.P)^R F\right]^G$

Le nombre de possibilités de déploiement d'un Groupe dans ses *H* assemblages possibles de Sous-Groupes est donné par : $\left[(C.P)^R F\right]^G H$.

Enfin, le nombre de possibilités de déploiement d'une l'application qui peut être utilisée par *U* utilisateurs simultanés est donné par : $\left\{\left[(C.P)^R F\right]^G H\right\}^U$.

Formule 11 : $\left\{\left[(C.P)^R F\right]^G H\right\}^U$

Le nombre maximal de configurations possibles de l'application est donc donné par ce nombre de possibilités de déploiement de l'application. Afin de mesurer l'ampleur de la complexité, le Tableau 12 présente les valeurs obtenues dans quatre cas simples qui diffèrent par les valeurs des différentes variables. Le quatrième cas correspond à la vidéoconférence telle qu'elle a été définie aux paragraphes 2.1.2.1 et 2.3.4. Nous supposons également que le concepteur dispose d'un seul composant pour chaque rôle atomique

excepté pour la réduction de la taille des images pour laquelle il peut utiliser 2 composants différents : C=2 et R=6.

Notation	Nombre de	1er cas	2ème cas	3ème cas	Vidéo-conférence
U	utilisateurs	10	7	5	4
P	postes	10	5	5	4
G	Sous-Groupes	5	3	3	3
F	décompositions fonctionnelles	3	2	2	6
H	assemblages de Sous-Groupes	2	2	2	2
R	rôles pour une décomposition fonctionnelle	20	10	10	6
C	composants pour un rôle	5	3	2	2
CP	déploiements d'un rôle	50	15	10	8
$(CP)^R$	déploiements d'une décomposition fonctionnelle	1E+34	6E+11	1E+10	262144
$(CP)^R F$	déploiements d'un Sous-Groupe	3E+34	1E+12	2E+10	1572864
$((CP)^R F)^G$	déploiements d'un assemblage de Sous-Groupes	2E+172	2E+36	8E+30	4E+18
$((CP)^R F)^G H$	déploiements d'un Groupe	4E+172	3E+36	2E+31	8E+18
$(((CP)^R F)^G H)^U$	déploiements de l'application	$> 10^{307}$	3E+255	1E+156	4E+75

Tableau 12 : Exemple de nombre maximal de configurations

Nous voyons donc que, même dans l'exemple très simple de vidéoconférence que nous avons choisi, la complexité dans le pire cas est très importante. Nous observons que le premier terme exponentiel apparaît pour le nombre de possibilités de déploiement d'une décomposition fonctionnelle. L'exposant est donné par le nombre de rôles atomiques nécessaires pour réaliser une fonctionnalité. Théoriquement, dès ce stade, la plate-forme ne pourra pas définir tous les déploiements possibles d'une configuration.

3.1.1.3. Complexité réelle

Nous venons de montrer que déterminer la configuration d'une application optimale du point de vue de la QdS est un problème NP-complet dans le pire cas. Nous avons vu que les contraintes induites par notre modélisation n'influent pas sur cette complexité. De façon plus précise, la détermination des possibilités de déploiement d'une décomposition fonctionnelle est déjà un problème non polynomial que la plate-forme ne pourra pas réaliser en théorie.

Cependant, la complexité réelle de notre algorithme ne sera évidemment pas celle calculée précédemment car les nécessités d'une modélisation mathématique générique induisent une complexité largement supérieure à la complexité réelle. Nous ne saurions le dire mieux que dans [MOR 02] :

« *Formal approaches to compositional reasoning treat software components as mathematical objects that are described by mathematical models, for example labeled*

transition networks. While the mathematical treatment of software components is the great strength of any formal approach, it is also its Achilles heel. Automated theorem proving is known to be NP-complete, and thus introduce new problems of scale and complexity in place of those eliminated by divide and conquer. We do not argue against formal reasoning. Instead, we observe that where formal approaches are infeasible, empirical approaches must be used. In this setting, software components and component assemblies are treated as physical rather than mathematical objects. »

pouvant se traduire par :

« Les approches formelles pour la composition traitent les composants logiciels en tant qu'objets mathématiques décrits par des modèles mathématiques par exemple des réseaux avec des transitions étiquetées. Or si le traitement mathématique des composants logiciels est la grande force de toute approche formelle, c'est aussi son talon d'Achille. Il est connu que la démonstration automatisée de théorème est NP-complète, ainsi elle introduit de nouveaux problèmes d'échelle et de complexité remplaçant ceux éliminés par le " diviser pour régner ". Nous ne sommes pas contre le raisonnement formel. Au contraire, nous observons que là où les approches formelles sont infaisables, les approches empiriques doivent être utilisées. Dans ce cadre, les composants logiciels et les assemblages de composants sont traités en objets physiques plutôt que mathématiques. »

Forts de cette analyse, nous pourrons utiliser les caractéristiques réelles de l'application pour réduire la complexité. En particulier, nous n'avons pas pu prendre en compte le fait que :

- tous les composants ne peuvent pas être déplacés et principalement les composants observables ;
- tous les rôles ne disposeront pas de plusieurs composants pour être réalisés ;
- l'utilisation de certains rôles ou composants impose celle d'autres rôles ou composants comme lors d'une compression décompression ;
- les synchronisations réalisées par les Conduits et les Processeurs Élémentaires réduisent les possibilités de localisation.

D'autre part, les décompositions fonctionnelles découlent des graphes des flots de contrôle définis par le concepteur et des possibilités de remplacer un rôle non atomique par un arrangement de rôles atomiques que l'on peut éventuellement ordonnancer selon différentes possibilités. Or le concepteur conçoit l'application en sachant que proposer différentes qualités de service n'a de sens que si l'utilisateur peut donner son avis et percevoir la différence entre les configurations. Nous pouvons donc faire l'hypothèse que le nombre de décompositions fonctionnelles et, a fortiori, le nombre d'ordonnancements possibles, n'est pas trop important.

Pour illustrer ce propos, nous calculons le nombre de configurations différentes que peut proposer notre exemple de vidéoconférence.

Le Sous-Groupe Suivi est décrit par une seule décomposition fonctionnelle dans laquelle chaque rôle peut être réalisé par un seul composant parmi lesquels un seul est délocalisable — le composant réalisant le calcul des mouvements de la caméra. Ce Sous-Groupe propose donc P configurations en fonction des postes utilisés par l'application.

Le Sous-Groupe Son est décrit par deux décompositions fonctionnelles qui se distinguent par la présence ou l'absence de compression. Chaque rôle peut y être réalisé par un seul composant. Pour l'une des décompositions fonctionnelles, deux composants sont

délocalisables sur P postes alors qu'aucun ne l'est dans l'autre. Ce Sous-Groupe propose donc $1+P^2$ configurations.

Le même type de raisonnement permet de déterminer que le Sous-Groupe Image propose $\Sigma=5.P^2+3.P+1$ configurations grâce à ses 6 décompositions fonctionnelles.

Le Groupe Téléspectateur peut être réalisé par un seul assemblage de Sous-Groupes : un Sous-Groupe Son et un Sous-Groupe Image. Il dispose donc de Σ. $(1+P^2)$ configurations pour être implanté.

Le Groupe Locuteur peut être réalisé par deux assemblages de Sous-Groupes différents correspondant à la présence ou l'absence du Sous-Groupe Suivi en plus des Sous-Groupes Son et Image. Ce Groupe propose donc $\Sigma\left(1+P^2\right)+\Sigma\left(1+P^2\right)P=\Sigma\left(1+P^2\right)(1+P)$.

De plus, il existe 13 assemblages de Groupes possibles :

- 2 possibilités lorsque seul un téléspectateur est présent soit $2.\Sigma\left(1+P^2\right)(1+P)\Big|_{P=1}=72$ configurations possibles pour cet assemblage ;
- 2 possibilités lorsque seul un locuteur est présent soit $\Sigma\left(1+P^2\right)\Big|_{P=1}=18$ configurations possibles pour cet assemblage ;
- 2×2 possibilités lorsqu'un locuteur et un téléspectateur sont présents soit $2.\Sigma\left(1+P^2\right)(1+P).2.\Sigma\left(1+P^2\right)\Big|_{P=2}=218700$ configurations possibles pour cet assemblage ;
- 2 possibilités lorsqu'un locuteur et deux téléspectateurs sont présents soit $\Sigma\left(1+P^2\right)(1+P).2.\Sigma\left(1+P^2\right)\Big|_{P=3}=2420000$ configurations possibles pour cet assemblage ;
- 2 possibilités lorsque deux locuteurs et un téléspectateur sont présents soit $2.\Sigma\left(1+P^2\right)(1+P).\Sigma\left(1+P^2\right)\Big|_{P=3}=2420000$ configurations possibles pour cet assemblage ;
- 1 possibilité lorsque les quatre utilisateurs sont présents soit $2.\Sigma\left(1+P^2\right)(1+P).2.\Sigma\left(1+P^2\right)\Big|_{P=4}=24995610$ configurations possibles pour cet assemblage.

Le nombre de configurations possibles pour notre exemple de vidéoconférence dépasse donc 30 millions. Bien qu'incomparablement plus faible que celui donné au Tableau 12 page 156, ce nombre demeure tout de même excessivement élevé. La prise en compte des contraintes réelles a donc permis de réduire significativement la complexité mais insuffisamment pour s'assurer que la plate-forme pourra accomplir sa tâche dans un délai acceptable.

Ainsi la complexité de notre application sera plus faible que celle établie au paragraphe 3.1.1.2.b tout en restant très élevé. Nous ne pourrons donc pas faire l'hypothèse que la plate-forme pourra définir toutes les configurations possibles et les évaluer pour déterminer celle de meilleure QdS. Nous allons donc proposer un modèle fonctionnel moins simpliste

que celui-ci. D'autant plus que si nous pouvons nous contenter d'un algorithme non polynomial pour les tâches à réaliser avant l'exécution de l'application puisque le concepteur peut patienter quelques heures lors du développement, il nous faudra veiller à ce que, par leur efficacité, les tâches réalisées en cours d'exécution conservent aux reconfigurations tout leur intérêt. Ainsi la plate-forme devra réagir suffisamment rapidement par rapport aux variations du contexte et par rapport à l'utilisateur en particulier lorsqu'un blocage du service est imminent.

3.1.2. Solutions à la complexité

Le modèle fonctionnel de la plate-forme montre comment celle-ci détermine la configuration à implanter pour un contexte donné alors que nous venons de démontrer qu'il ne lui est pas possible de trouver à chaque fois et directement la configuration optimale. Nous présentons dans la suite les principes de la méthode heuristique utilisée par la plate-forme pour améliorer la QdS malgré la complexité du problème. Puis, comme cette méthode repose principalement sur la proximité de service entre deux configurations, nous définissons également cette notion.

3.1.2.1. Méthode d'optimisation de la qualité de service utilisée par la plate-forme

La plate-forme ne peut réaliser aucune des tâches identifiées comme non polynomiales. Nous proposons donc un modèle adapté aux applications multimédias réparties basé sur une série hiérarchisée d'itérations.

a. Approche itérative

Nous venons de démontrer que trouver la configuration ayant la meilleure QdS dans un contexte donné est un problème non polynomial aux différentes étapes que sont le choix des assemblages de composants, celui de leur ordonnancement et celui de leur déploiement. Une solution serait d'utiliser l'un des algorithmes habituellement proposés pour résoudre les problèmes NP-complets tels l'utilisation des listes, des algorithmes de *backtracking* ou des algorithmes génétiques.

Or ces méthodes de résolution mathématique ne tiennent pas compte de la proximité des solutions qu'elles proposent de sorte qu'elles présentent le risque de provoquer des changements brutaux de service susceptibles de perturber l'utilisateur. Elles vont donc à l'encontre de la plasticité nécessaire aux reconfigurations c'est-à-dire la capacité du système à s'adapter aux variations des ressources interactionnelles, computationnelles, communicationnelles et environnementales tout en conservant la continuité ergonomique [COU 01].

Nous allons donc proposer une heuristique dédiée aux applications multimédias réparties qui optimise la QdS en fonction du contexte sans détériorer celle-ci par des discontinuités. Dans ce cadre, nous choisissons une heuristique itérative qui réalise une évolution graduelle de la QdS. Chaque itération recherche une configuration meilleure que

celle en cours d'exécution — et non plus la meilleure — qu'elle implante. Ainsi la QdS est améliorée à chaque itération et le système va converger vers la meilleure configuration.

b. Principe de choix de la future configuration

Nous avons vu qu'il n'est pas possible de définir toutes les configurations ni de les évaluer pour trouver une configuration meilleure que l'actuelle. Nous proposons qu'à chaque itération, la plate-forme définisse un ensemble initial fini de configurations dont elle détermine la QdS. Si au moins l'une de ces configurations est meilleure que l'actuelle, la plate-forme la choisit et l'implante. Si aucune configuration n'est meilleure, la plate-forme élargit l'ensemble d'étude et reprend la recherche. A chaque élargissement du champ de recherche correspond un éloignement plus important par rapport à la situation actuelle. La plasticité sera donc assurée de façon optimale.

Si, après la reconfiguration, la solution implantée n'est pas la meilleure, la plate-forme le détecte et recommence une recherche avec de nouveaux ensembles d'étude de manière à élargir l'espace de recherche. Le résultat de la recherche et sa durée vont dépendre de l'ordre dans lequel sont définis les différents ensembles car il y a aura probablement plusieurs configurations meilleures que l'actuelle. Le principe du choix de la future configuration est résumé dans la Figure 63.

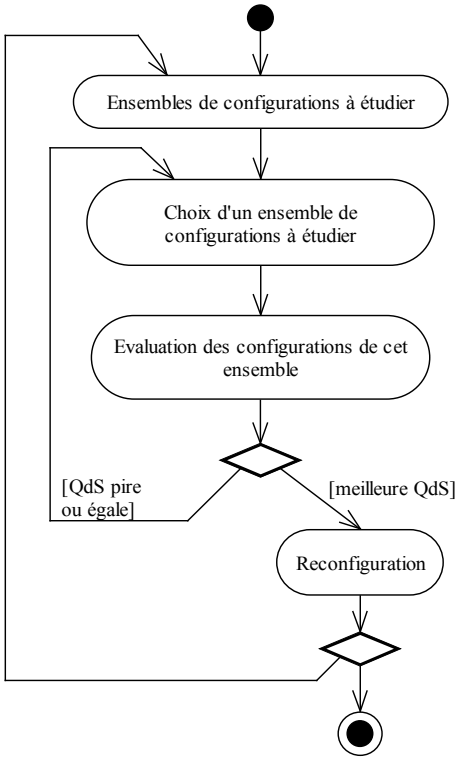


Figure 63 : Principe de choix de la future configuration

Notons que la comparaison de la QdS des configurations sera simplifiée par les propriétés de la fonction QdS (cf. §2.1.3.4.b p.96).

c. Configurations étudiées et ordre d'évaluation

L'efficacité de l'heuristique dépend du choix des configurations à étudier et de leur ordre d'étude de sorte que le choix des critères permettant de définir les différents ensembles d'étude doit être guidé par les caractéristiques propres aux applications multimédias et à leur QdS.

Tout d'abord, la reconfiguration doit respecter les contraintes temporelles des applications multimédias. Elle doit donc être la plus rapide possible de manière à ce que la plate-forme soit la plus efficace possible. Nous proposons donc de choisir, à chaque étape, l'ensemble d'étude qui a le plus de chance de proposer une amélioration et, pour cela, nous allons utiliser les informations décrivant le contexte dont dispose la plate-forme. Elles vont nous indiquer quelle entité modifier en priorité dans la configuration actuelle : quel Groupe, quel composant, quel flux.

D'autre part, nous avons vu qu'il était essentiel d'assurer la continuité ergonomique de l'application. Nous allons donc rechercher une configuration meilleure en partant de celles qui modifient peu le service de l'utilisateur pour aller vers celles qui le modifient significativement. Ainsi la détermination des différents ensembles d'étude des configurations, et l'ordre selon lequel la plate-forme va les évaluer, vont être guidés par la proximité de service. Celle-ci va donc nous indiquer comment modifier la configuration actuelle et en particulier à quel niveau structurel, Groupe, Sous-Groupe, rôle, ou composant.

Pour utiliser les deux critères que nous venons de définir — informations sur le contexte et proximité de service — il est nécessaire de connaître les informations dont dispose la plate-forme pour définir le contexte et pour définir la proximité de service. La perception du contexte est une problématique connue pour laquelle nous présenterons brièvement nos propositions alors que l'évaluation de la proximité du service est plus originale, c'est pourquoi nous nous attarderons plus longuement sur son étude.

d. Perception du contexte

La sensibilité au contexte ou *context-awareness* est l'une des nécessités de l'adaptation des applications. Elle consiste, pour la plate-forme, à être capable de récolter et de traiter les informations pertinentes pour la supervision de l'application. Nous proposons qu'elle utilise pour cela trois types d'informations.

Le premier type regroupe les mesures passives effectuées sur les composants et les flux par les Conduits et les Processeurs Élémentaires. Elles permettent, d'une part, d'évaluer la QdS de la configuration actuelle et, d'autre part, de détecter lorsqu'une reconfiguration est nécessaire parce que la QdS se dégrade ou, au contraire, parce qu'elle pourrait être améliorée. Ainsi les Conduits et les Processeurs Élémentaires possèdent en entrée et en sortie des buffers dont l'état, saturé ou quasi-vide, reflète la fluidité d'écoulement des flux d'informations et donc les problèmes de saturation et de sous-emploi des ressources. Dans une telle situation, le Conduit ou le Processeur Élémentaire génère un **événement dit de reconfiguration** que la plate-forme traite.

L'application est constituée de composants reliés par des flux. Notre implantation encapsule les flux dans des Conduits et les composants logiciels dans des Processeurs Élémentaires. Leur rôle est, d'une part, d'assurer la synchronisation des flux d'informations tant lors de leur transport — Conduits — que de leur traitement — PE — et d'autre part, de jouer un rôle de surveillance du contexte d'exécution de l'application. En particulier, le

Conduit peut détecter un problème de transport d'information et identifier le flux concerné à partir des informations le décrivant. De plus, il est capable d'identifier si l'origine du problème sur un flux est le site récepteur ou l'état de la liaison. Dans le premier cas, c'est le composant récepteur qui sera considéré comme problématique tandis que, dans le second cas, c'est le composant émetteur. De la même façon, le Processeur Élémentaire génère un événement lorsque le composant qu'il encapsule pose problème, en particulier, lorsqu'il ne parvient pas à traiter la totalité des informations qui lui parviennent ou à disposer des ressources qu'il revendique. La plate-forme doit alors adapter sa réponse à chaque cas. Nous qualifierons dorénavant de **problématique** le composant que le Conduit ou le Processeur Élémentaire identifie comme à l'origine du besoin de reconfiguration.

Remarquons que la surveillance de l'application ne permet pas seulement de détecter les baisses de QdS mais elle détermine également les contextes dans lesquels la QdS peut être augmentée. En effet, un Conduit ou un Processeur Élémentaire peut détecter une sous utilisation des buffers et mettre en place des mesures actives. Ces dernières constituent le deuxième type d'informations relatives au contexte et permettent de tester la marge de manœuvre dont dispose l'application en terme de ressources systèmes telles la bande passante ou l'occupation d'un processeur. Elles servent, d'une part, à évaluer la QdS potentielle d'une configuration différente de celle en cours d'exécution et d'autre part, à déterminer si une reconfiguration est nécessaire. Un événement de reconfiguration est alors généré par le Conduit ou le Processeur Élémentaire concerné.

Enfin, les dernières informations sur le contexte sont recueillies par des **composants dits espions** qui lèvent des événements de reconfiguration liés à des informations non mesurables comme la langue utilisée par un locuteur dans une vidéoconférence. Ces événements indiquent que le service n'est plus adapté. Nous proposons que le concepteur associe un composant espion à chaque caractéristique intrinsèque de service qui lui semblera opportune. Ce composant compare l'état de la caractéristique avec les critères optimaux que peut fournir l'application en fonction des vœux de l'utilisateur et génère un événement de reconfiguration s'il y a une différence. L'événement produit induit des contraintes sur les solutions à trouver comme l'utilisation obligatoire de tel composant ou de telle fonctionnalité ou la suppression d'une entité. Il est à remarquer que des composants espions sont également utilisés pour détecter l'arrivée ou le départ d'un utilisateur et participent de ce fait au déploiement de l'application en provoquant la création ou la suppression d'une instance de Groupe.

La plate-forme répartie doit donc récolter ces trois types d'informations. Nous supposons qu'elle dispose à tout instant des informations de contexte du poste concerné à la manière des superviseurs utilisés dans les systèmes industriels pour deux raisons. La première est l'utilisation d'événements qui doit permettre d'alléger la charge d'occupation de la plate-forme pour cette récolte. La seconde est le constat établi dans les S.B.S. — Services Basés Capteurs — où récolter une information de contexte a le même coût en temps qu'en récolter plusieurs. Nous proposons que l'évaluation de la QdS par la plate-forme soit répartie de manière à ce que les entités qui s'exécuteraient sur un poste soient évaluées sur ce poste. Ainsi, lorsque la plate-forme évalue une configuration, elle dispose, en temps réel, des dernières mesures valides du contexte.

3.1.2.2. Proximité du service

Le second critère permettant de choisir les configurations à évaluer est la proximité de leur service avec celui fourni par la configuration en cours d'exécution. L'efficacité de la plate-forme dépend principalement de la définition et de la mise en œuvre de cette notion.

C'est pourquoi nous allons la définir précisément avant d'expliciter son lien avec l'architecture de l'application, puis expliquer pourquoi et comment le concepteur et les utilisateurs interviennent dans sa détermination. Nous pourrons alors proposer un classement des configurations en fonction de la proximité de leur service avec celui fourni par la configuration en cours d'exécution. C'est en suivant cet ordre que la plate-forme étudiera leur QdS pour trouver une qualité meilleure et l'implanter.

a. Définition de la proximité du service

Nous dirons que deux configurations ont des **services proches** [éloignés] si l'utilisateur perçoit peu ou pas [énormément] le passage de l'une à l'autre. Ce critère doit guider l'ordre d'évaluation des configurations potentielles de l'application dans un contexte donné. Nous cherchons donc une relation de type $\mathbf{C_A > C_B}$ signifiant que si deux configurations, C_A et C_B, diffèrent de la configuration actuelle dans leur structure ou leur composition par A, elles ont un service plus éloigné de celui de la configuration actuelle que si elles en diffèrent par B. En d'autres termes, un changement sur A est plus perceptible qu'un changement sur B. La configuration C_B sera alors évaluée par la plate-forme avant la configuration C_A.

Pour que le service rendu soit proche, la première condition est que ce service soit de même nature c'est-à-dire que les caractéristiques du service ne dépendant pas du contexte soient identiques. Notre modèle traduit alors cette propriété par des notes identiques pour le critère intrinsèque de QdS. Une fois cette première condition remplie, si le changement de service est imperceptible, c'est qu'en plus d'avoir un critère intrinsèque identique, les critères contextuels sont eux aussi identiques, condition nécessaire mais non suffisante. Or si les critères intrinsèque et contextuel ont les mêmes notes, les deux configurations ont la même QdS. Inversement, par définition de la QdS, si les critères intrinsèque et contextuel sont identiques, les services sont identiques pour l'utilisateur et donc les deux configurations ont des services identiques. Nous pouvons donc dire que deux configurations fournissent un service proche si et seulement si leurs notes des critères intrinsèque et contextuel sont proches ce qui implique que les notes de QdS soient proches. Nous remarquerons que ceci n'implique en rien que la configuration la plus proche au sens du service soit celle dont la note de QdS est la plus proche.

Finalement, la plate-forme commencera donc sa recherche par l'évaluation des configurations ayant la même note de critère intrinsèque que la configuration actuelle. Elle se contentera donc dans un premier temps de modifier la note du critère contextuel. Cependant, les premières configurations étudiées parmi celles de critère intrinsèque identique seront celles proposant le service le plus proche et c'est l'architecture de l'application qui va nous aider à les déterminer.

b. Lien entre l'architecture et la proximité de service

Notre modèle architectural de l'application est basé sur la perception que l'utilisateur a du service et reflète donc la proximité de service. En effet, si une configuration diffère de l'actuelle par un Groupe — $\mathbf{C_G}$ —, elle rend un service totalement différent à l'utilisateur concerné. Si la différence est au niveau Sous-Groupe — $\mathbf{C_{SG}}$ —, elle fournit le même service mais à l'aide de fonctionnalités différentes. Elles ont donc une plus grande proximité de service que dans le cas précédent. Enfin, si seuls des composants les différencient, elles fournissent les mêmes fonctionnalités mais de manière différente.

En revanche, pour les niveaux suivants, rôle — $\mathbf{C_R}$ — et composant — $\mathbf{C_C}$ —, il n'existe pas de lien direct entre le niveau où intervient la différence entre deux

configurations et la perception qu'a l'utilisateur du passage d'une configuration à l'autre. En effet, le choix de composants pour certains rôles comme le traitement d'image peut avoir de grandes répercussions sur le service fourni alors qu'un changement de rôles peut ne pas être directement perceptible comme dans le cas d'une transmission d'image avec ou sans compression dans un contexte favorable.

Nous avons donc deux relations d'ordre partiel : $C_G > C_{SG} > C_R$ et $C_G > C_{SG} > C_C$. Comme l'architecture ne nous permet pas d'ordonner les différences de niveaux bas — C_R et C_C —, il faut trouver d'autres critères issus de l'aspect subjectif de la proximité de service. Ils nous seront donnés par le concepteur puis l'utilisateur.

c. Définition de la proximité du service par le concepteur

Le concepteur connaît la sémantique des rôles de l'application. De plus, contrairement à la plate-forme, il peut identifier les rôles et composants qui ont des répercussions importantes sur le service rendu par rapport à d'autres plus neutres. Nous allons donc utiliser la notion d'entité critique pour déterminer la proximité de service lorsque les configurations diffèrent par des rôles ou des composants.

Définition des entités critiques

Nous définissons une **entité — rôle ou composant — critique** pour l'utilisateur comme un rôle ou un composant dont la modification, la suppression ou l'ajout a des répercussions notables et directes sur le service fourni par opposition à celles dont les répercussions sont moindres ou indirectes — comme la compression qui agit sur les performances par le biais de l'encombrement réseau. Sont critiques uniquement les entités pour lesquelles la plate-forme a un choix à effectuer : le choix de la décomposition fonctionnelle, le choix du composant réalisant le rôle ou le choix du déploiement. Un rôle ou un composant qui ne peut être implanté que d'une seule façon ne peut pas être critique.

C'est donc le concepteur de l'application qui marquera comme critique un rôle ou un composant. Pour cela, il dispose uniquement des caractéristiques de service non contextuelles. Un composant réalisant un rôle critique hérite, a priori, du caractère critique mais peut le perdre s'il s'avère qu'il n'y a pas de choix à effectuer pour l'implanter. De la même façon, un rôle critique perdra son caractère critique s'il est présent dans toutes les décompositions fonctionnelles alors que le composant le réalisant sera critique si la plate-forme doit effectuer un choix d'implantation. Nous remarquons qu'une fonctionnalité — un Sous-Groupe — contient forcément un rôle et un composant observable qui sont a priori critiques au départ de l'étude.

Reprenons l'exemple de la transmission d'image décrit au paragraphe 2.3.4.1.b à la Figure 57 p.141. Ce Sous-Groupe peut être réalisé (cf. Figure 64 p.165) soit :

- avec une réduction de la taille des images et/ou du codage des couleurs ;
- avec une compression des images ;
- avec une transmission directe sans traitement.

Rappelons que de manière à pouvoir fournir simultanément des images de qualité différentes aux différents utilisateurs, ces traitements ne sont pas réalisés à l'acquisition. Certains rôles sont présents dans toutes les configurations possibles et ne sont donc pas critiques. Il s'agit de $R_{4\text{-}1}$, $R_{4\text{-}2}$, $R_{6\text{-}1}$ et $R_{6\text{-}2}$. La présence d'une compression et d'une

décompression est imperceptible pour l'utilisateur. En effet, comme nous l'avons déjà vu, celui-ci n'est sensible qu'à l'amélioration du débit résultant : les rôles $R_{2\text{-}1}$ et $R_{2\text{-}2}$ ne sont donc pas critiques. En revanche, l'utilisateur est sensible à la réduction de la taille des images et du nombre de couleurs : les rôles $R_{5\text{-}1}$ et $R_{5\text{-}2}$ sont donc critiques.

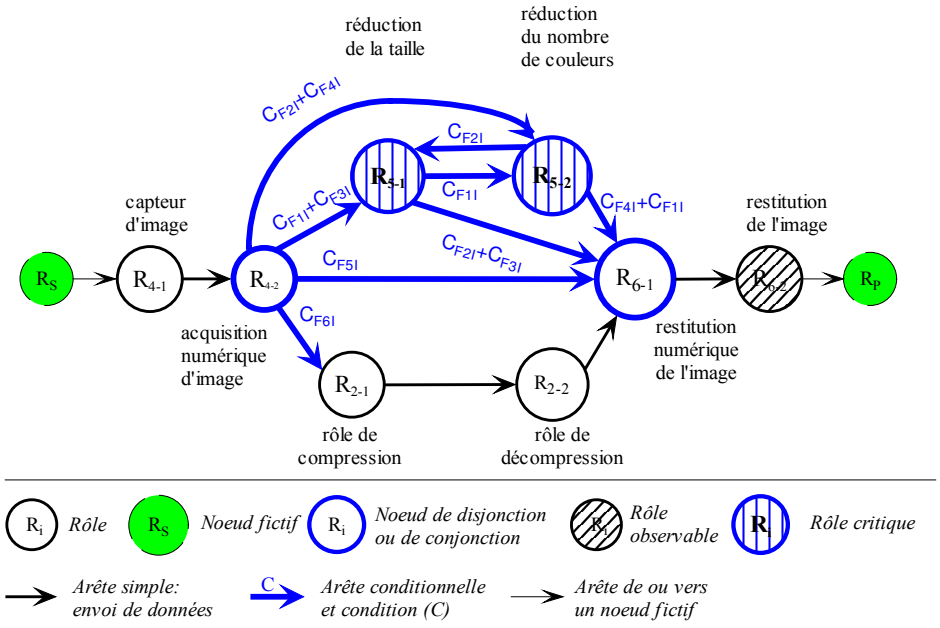


Figure 64 : Rôles critiques dans le Sous-Groupe Image

Proximité de service à partir des entités critiques

Les entités critiques vont nous permettre de classer les configurations en fonction de la proximité de leur service avec la configuration en cours d'exécution lorsqu'elles sont réalisées à l'aide des mêmes Sous-Groupes que cette dernière. Par définition des entités critiques, une configuration qui diffère de celle en cours d'exécution par un composant [rôle] critique, différence que l'on notera $\mathbf{C_{CC}}$ [$\mathbf{C_{RC}}$], sera plus éloignée du service de cette dernière qu'une configuration différant par un composant non critique, différence notée $\mathbf{C_{CnC}}$ [$\mathbf{C_{RnC}}$]. Nous aurons donc les relations $C_{CC}>C_{CnC}$ et $C_{RC}>C_{RnC}$. De plus, toujours par définition, une modification concernant une entité critique est plus perceptible qu'une autre touchant une entité non critique donc $C_{CC}>C_{RnC}$. D'autre part, modifier un rôle permet de modifier la manière de faire et non pas seulement la QdS. Les configurations qui diffèrent de la configuration en cours d'exécution par un rôle ont donc un service plus éloigné que celles qui en diffèrent par un composant ce que nous transcrivons par $C_{RC}>C_{CC}$ et $C_{RnC}>C_{CnC}$. Finalement les entités critiques permettent d'établir la relation :

$$C_{RC}>C_{CC}>C_{RnC}>C_{CnC.}$$

Nous remarquons que modifier la configuration en changeant un composant ou un rôle non critique provoquera une faible variation de la note du critère intrinsèque et ceci par définition de la notion de criticité. En revanche, le critère contextuel pourra être fortement changé comme dans le cas de l'utilisation d'une compression lorsque le réseau de transmission sature. La relation que nous venons d'établir respecte bien l'ordre énoncé au paragraphe 3.1.2.2.a : la plate-forme évalue d'abord les configurations ayant des critères

intrinsèques identiques à ceux de la configuration actuelle avant d'étudier celles qui ont des notes intrinsèques différentes.

Cette étude nous permet d'établir quelles sont les configurations qui offrent le service le plus proche de celle en cours d'exécution parmi celles qui en diffèrent par un Groupe, un Sous-Groupe, un rôle critique, un composant critique, un rôle non critique ou un composant non critique. Cet ordre est représenté par la relation :

$$C_G > C_{SG} > C_{RC} > C_{CC} > C_{RnC} > C_{CnC}.$$

C'est donc dans cet ordre que la plate-forme va évaluer la QdS que ces configurations pourraient fournir si elles étaient implantées. Elle pourra ainsi déterminer celle qui permettra d'améliorer la QdS. Cependant nous n'avons pas établi d'ordre entre deux configurations qui diffèrent de la configuration en cours d'exécution par le même niveau structurel — C_G ou C_{SG} — ou la même entité critique — C_{RC}, C_{CC}, C_{RnC} ou C_{CnC}. C'est l'utilisateur qui va nous fournir le critère discriminant.

d. Définition de la proximité du service par l'utilisateur

L'architecture et le concepteur de l'application nous permettent de juger globalement de la proximité de service entre deux configurations mais seul l'utilisateur peut la définir précisément puisqu'il connaît seul les caractéristiques importantes pour lui. Or il n'est pas possible de lui demander la totalité des informations qui permettraient de classer toutes les configurations car cela représenterait une quantité d'informations trop importante. En particulier, plus les configurations à ordonner diffèrent par une entité de bas niveau architectural plus le nombre de configurations à étudier est grand. Nous proposons donc que le concepteur définisse, en fonction de l'application, le niveau le plus bas sur lequel la plate-forme proposera à l'utilisateur d'exprimer ses préférences.

Ainsi, nous proposons une définition du nombre de paramètres de QdS qui se situe entre les deux attitudes généralement observées dans le domaine des réseaux [AUR 98]. La première limite les paramètres décrivant les flux à un ou deux de manière à ce que le système reste facilement gérable mais restreint les possibilités de QdS. La seconde propose des spécifications multivaluées plus complexes et donc plus coûteuses. Nous laissons ce choix à la discrétion du concepteur qui peut donc adapter l'étendue de l'offre de qualité.

Principe

L'utilisateur précisera ses vœux de service de différentes façons en fonction de la complexité du niveau concerné. Ainsi il pourra ordonner toutes les possibilités pour des complexités faibles comme les Groupes. Il paramètrera le système d'évaluation pour des complexités un peu plus élevées et la plate-forme se chargera alors de générer les notes et d'ordonner les possibilités étudiées. En revanche, pour les complexités trop élevées, de type exponentiel, il ne lui sera pas possible d'exposer ses préférences.

Nous proposons que l'utilisateur choisisse son Groupe — son type de service — parmi ceux disponibles. En effet, leur nombre U n'est pas élevé puisque les Groupes ont été définis par le concepteur. Puis il classe les assemblages de Sous-Groupes — fonctionnalités — possibles pour réaliser ce service. Ils sont au nombre de H et sont également établis par le concepteur. Pour chaque assemblage, il donne un poids aux Sous-Groupes sous forme de pourcentage dont la somme doit faire 100%. Ce système de pourcentage permet d'éviter que l'estimation de la qualité d'un Groupe ne dépende du nombre de ses fonctionnalités.

Dans notre exemple de vidéoconférence, les utilisateurs sur les postes 1 et 2 ont choisi le Groupe Locuteur et ceux sur les postes 3 et 4, le Groupe Téléspectateur (cf. Tableau 13). Les premiers classent ensuite les deux configurations possibles pour leur Groupe en fonction de la présence du Sous-Groupe assurant le suivi de leurs mouvements. Nous supposons que le premier utilisateur préfère bénéficier du suivi de ses mouvements alors que le second n'est pas intéressé par cette fonctionnalité. Puis, chaque utilisateur indique ses préférences de service pour chaque configuration de son Groupe. Ainsi le téléspectateur du poste 3 précise qu'il accorde plus d'importance à la réception du son que de l'image et donne donc les pourcentages 75 au son et 25 à l'image. Les vœux des utilisateurs pour notre exemple de vidéoconférence sont alors présentés dans le Tableau 13.

Poste	1		2		3	4
Groupe choisi	Locuteur		Locuteur		Téléspect.	Téléspect.
Classement des assemblages de Sous-Groupes	1 Son Image Suivi	2 Son Image	1 Son Image Suivi	1 Son Image		
Poids des Sous-Groupes	40% Son	50% Son	50% Son	50% Son	75% Son	50% Son
	40% Image	50% Image	50% Image	50% Image	25% Image	50% Image
	20% Suivi		0% Suivi			

Tableau 13 : Vœux des utilisateurs dans l'exemple de vidéoconférence

Passé ce niveau, le nombre d'assemblages de composants utilisables pour un Sous-Groupe peut être très élevé, c'est pourquoi nous proposons que le concepteur définisse à quelle granularité du Sous-Groupe l'utilisateur indique ses souhaits sous la forme du système de notation des caractéristiques de service pertinentes. En dessous de cette granularité, il n'est plus possible de récolter les préférences de l'utilisateur. Nous appelons cette granularité " familles de configurations".

Familles de configurations

Le concepteur utilise l'ordre établi par l'étude de la proximité de service pour choisir le niveau de différenciation entre les familles : $C_{RC}>C_{CC}>C_{RnC}>C_{CnC}$. Une famille peut être définie de quatre façons comme l'ensemble des configurations ayant :

- les mêmes rôles critiques, famille notée **RC** ;
- les mêmes rôles critiques et les mêmes composants critiques, famille notée **CC ;**
- les mêmes rôles critiques et non critiques et les mêmes composants critiques, famille notée **RnC** ;
- les mêmes rôles critiques et non critiques et les mêmes composants critiques et non critiques. Elles auront alors la même composition mais elles n'implantent pas systématiquement la même décomposition fonctionnelle puisque ces rôles peuvent être ordonnés différemment. Cette famille est notée **CnC**.

En fonction de l'application, le concepteur choisit quel niveau de définition utiliser pour les familles puis, pour chaque famille, il identifie les caractéristiques intrinsèques et contextuelles qui seront prises en compte. Pour cela, il utilise les spécifications des composants qui la constituent et l'architecture qui la décrit. Les notes des caractéristiques contextuelles dépendent du contexte. Le concepteur pourra cependant préciser leurs valeurs limites que l'utilisateur notera ce qui permettra de définir par extrapolation le système de notation contextuelle. Les caractéristiques intrinsèques ne dépendent que des familles.

Celles-ci seront donc caractérisées par une note de critère intrinsèque unique qui est identique pour toutes les configurations de la famille et qui est attribuée par l'utilisateur. C'est donc le nombre d'informations que l'utilisateur peut donner sur ses préférences qui permet de limiter la complexité du problème grâce à la définition des familles.

Reprenons l'exemple de la vidéoconférence et du Sous-Groupe chargé de la transmission de l'image (cf. §3.1.2.2.c p.164). Le concepteur a défini (cf. Figure 64 p.165) les rôles critiques à partir de la fonctionnalité du Sous-Groupe et des différentes décompositions fonctionnelles. Nous avons supposé (cf. §3.1.1.2.b p.155) qu'il dispose d'un seul composant pour chaque rôle atomique excepté pour la réduction de la taille des images pour laquelle il peut utiliser deux composants différents : l'un réduisant la taille de moitié et l'autre du quart. Ainsi il est possible d'utiliser simultanément ces deux réductions pour des utilisateurs différents. Seul le composant de réduction de la taille des images est donc critique dans le Sous-Groupe puisque c'est le seul pour lequel il y a un choix d'implantation. La Figure 65 identifie les composants critiques dans les différentes configurations du Sous-Groupe Image qui utilise le composant $C_{5\text{-}1a}$ pour réaliser la réduction de la taille des images.

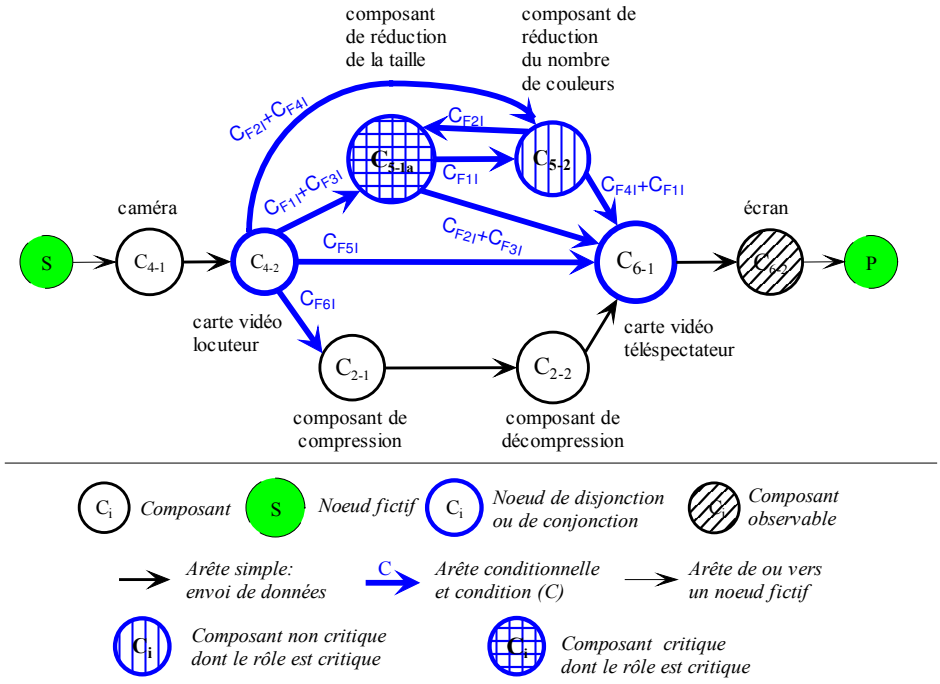


Figure 65 : Composants critiques dans le Sous-Groupe Image

Notons qu'il est également possible qu'un composant soit critique alors que son rôle ne l'est pas. En effet, ce serait le cas de la carte d'acquisition vidéo si le locuteur disposait de deux éléments de qualité différente. Comme ce rôle est présent dans toutes les décompositions, il n'est pas critique. En revanche, si l'utilisateur disposait de deux cartes le composant carte vidéo serait critique car la qualité de l'acquisition est très perceptible pour l'utilisateur.

Le concepteur peut alors définir les familles en fonction des rôles critiques et des composants critiques. Il va s'appuyer pour cela sur le Tableau 14 qui décrit les rôles

critiques ou non critiques et les composants critiques ou non critiques pour chaque décomposition fonctionnelle du Sous-Groupe.

Configuration	Rôles		Composants	
	critiques	non critiques	critiques	non critiques
C_{F1I}	R_{5-1} R_{5-2}	R_{4-1} R_{4-2} R_{6-1} R_{6-2}	C_{5-1a} (ou C_{5-1b})	C_{4-1} C_{4-2} C_{6-1} C_{6-2} C_{5-2}
C_{F2I}	R_{5-1} R_{5-2}	R_{4-1} R_{4-2} R_{6-1} R_{6-2}	C_{5-1a} (ou C_{5-1b})	C_{4-1} C_{4-2} C_{6-1} C_{6-2} C_{5-2}
C_{F3I}	R_{5-1}	R_{4-1} R_{4-2} R_{6-1} R_{6-2}	C_{5-1a} (ou C_{5-1b})	C_{4-1} C_{4-2} C_{6-1} C_{6-2}
C_{F4I}	R_{5-2}	R_{4-1} R_{4-2} R_{6-1} R_{6-2}	–	C_{4-1} C_{4-2} C_{6-1} C_{6-2} C_{5-2}
C_{F5I}	–	R_{4-1} R_{4-2} R_{6-1} R_{6-2}	–	C_{4-1} C_{4-2} C_{6-1} C_{6-2}
C_{F6I}	–	R_{4-1} R_{4-2} R_{6-1} R_{6-2} R_{2-1} R_{2-2}	–	C_{4-1} C_{4-2} C_{6-1} C_{6-2} C_{2-1} C_{2-2}

Tableau 14 : Caractère critique des rôles et composants des décompositions fonctionnelles du Sous-Groupe Image

Il y a alors quatre définitions possibles pour les familles. La première correspond à regrouper dans une même famille les configurations ayant les mêmes rôles critiques. Le Tableau 14 nous permet d'identifier les familles suivantes :

- la famille des configurations ayant R_{5-1} et R_{5-2} comme rôles critiques : ce sont les configurations qui sont décrites par les décompositions fonctionnelles C_{F1I} et C_{F2I} ;
- la famille des configurations ayant R_{5-1} comme unique rôle critique : ce sont les configurations qui sont décrites par la décomposition fonctionnelle C_{F3I} ;
- la famille des configurations ayant R_{5-2} comme unique rôle critique : ce sont les configurations qui sont décrites par la décomposition fonctionnelle C_{F4I} ;
- la famille des configurations n'ayant pas de rôle critique : ce sont les configurations qui sont décrites par les décompositions fonctionnelles C_{F5I} et C_{F6I}.

En utilisant le même raisonnement pour les autres définitions possibles des familles, le concepteur établit alors toutes les possibilités de définitions qui sont regroupées dans le Tableau 15.

Niveau de définition des familles		Décompositions fonctionnelles des configurations des familles						
Nom	Entités communes	Famille 1	Famille 2	Famille 3	Famille 4	Famille 5	Famille 6	Famille 7
RC	**Rôles critiques**	**C_{F1I} et C_{F2I}**	**C_{F3I}**	**C_{F4I}**	**C_{F5I} et C_{F6I}**			
CC	Rôles critiques et composants critiques	C_{F1I} et C_{F2I} avec le composant $C_{5\text{-}1a}$	C_{F1I} et C_{F2I} avec le composant $C_{5\text{-}1b}$	C_{F3I} avec le composant $C_{5\text{-}1a}$	C_{F3I} avec le composant $C_{5\text{-}1b}$	C_{F4I}	C_{F5I} et C_{F6I}	
RnC	Rôles critiques et non critiques et composants critiques	C_{F1I} et C_{F2I} avec le composant $C_{5\text{-}1a}$	C_{F1I} et C_{F2I} avec le composant $C_{5\text{-}1b}$	C_{F3I} avec le composant $C_{5\text{-}1a}$	C_{F3I} avec le composant $C_{5\text{-}1b}$	C_{F4I}	C_{F5I}	C_{F6I}
CnC	Rôles critiques et non critiques et composants critiques et non critiques	C_{F1I} et C_{F2I} avec le composant $C_{5\text{-}1a}$	C_{F1I} et C_{F2I} avec le composant $C_{5\text{-}1b}$	C_{F3I} avec le composant $C_{5\text{-}1a}$	C_{F3I} avec le composant $C_{5\text{-}1b}$	C_{F4I}	C_{F5I}	C_{F6I}

Tableau 15 : Différentes définitions des familles pour le Sous-Groupe Image

En fonction de l'étendue de l'offre de QdS qu'il souhaite proposer aux utilisateurs, le concepteur choisit le niveau de définition des familles qu'il va utiliser. Nous remarquons que plus la granularité des comparaisons entre configurations s'affine, plus le nombre de familles augmente : la définition RC fournit quatre familles alors que la définition C_nC aboutit à sept familles.

Supposons maintenant que le concepteur choisisse de définir les familles au niveau RC. Il dispose donc de quatre familles pour lesquelles il doit spécifier les caractéristiques intrinsèques et contextuelles qui seront prises en compte. Il étudie les caractéristiques des composants :

- la caméra a une cadence vidéo de 30 images par seconde, images de taille 128 sur 196 pixels avec un codage des couleurs sur 24 bits — *couleurs vraies* ;
- les deux composants de réduction de la taille des images réduisent celle-ci à 64 sur 98 pour $C_{5\text{-}1a}$ et à 64 sur 48 pour $C_{5\text{-}1b}$ permettant ainsi de modifier également l'orientation des images ;
- le composant de réduction du codage des couleurs limite ce dernier à 8 bits — 256 couleurs.

Il décide alors d'utiliser comme caractéristiques intrinsèques la taille des images et leur codage et comme caractéristiques contextuelles le temps de traitement ou de transmission et le débit. Il propose alors aux utilisateurs de remplir un formulaire que nous présentons dans la Figure 66 avec, pour exemple, les réponses du téléspectateur du poste 4.

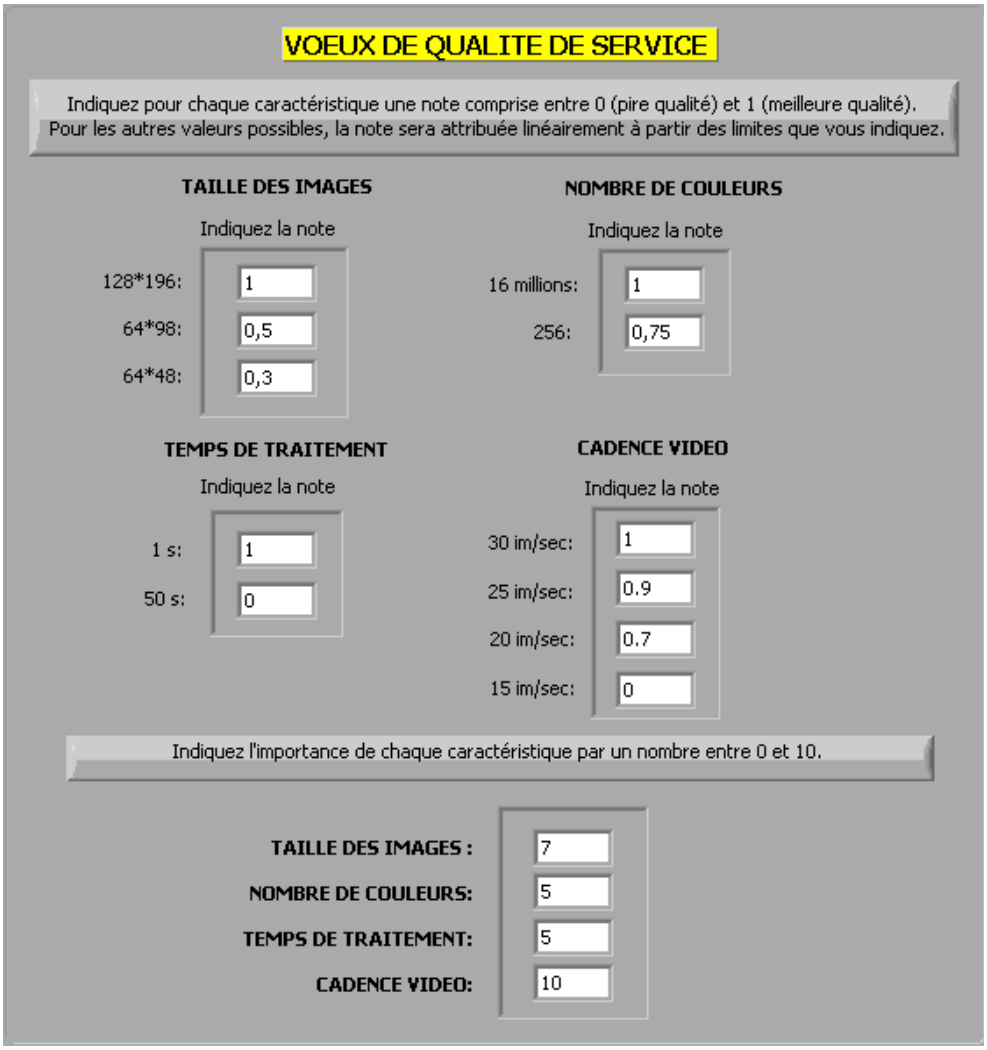


Figure 66 : Vœux d'un utilisateur de la vidéoconférence

Grâce à ces indications, le concepteur peut déterminer les notes intrinsèques des configurations des différentes familles (cf. Tableau 16 p.172) et ordonner celles-ci de la meilleure à la pire :

1. Famille 4 ;
2. Famille 3 ;
3. Famille 2 ;
4. Famille 1.

Famille		Note intrinsèque		
N°	Composition	Taille des images: poids = 7	Nombre de couleurs: poids = 5	Critère intrinsèque
1	C_{F1I} et C_{F2I}	0,5 ou 0,25	0,5	0,5 ou 0,35
2	C_{F3I}	0,5 ou 0,25	1	0,7 ou 0,56
3	C_{F4I}	1	0,5	0,79
4	C_{F5I} et C_{F6I}	1	1	1

Tableau 16 : Notes intrinsèques des familles du Sous-Groupe Image dans la vidéoconférence

Enfin, remarquons que définir les familles à l'aide des critères RnC et CnC équivaut à utiliser des rôles et composants non critiques pour différencier des configurations. Contrairement aux apparences, ceci n'est pas contradictoire avec la définition des entités critiques. En effet, le concepteur définit de façon subjective les rôles et composants critiques comme ceux dont la modification, la suppression ou l'ajout a des répercussions notables et directes sur le service fourni (cf. §3.1.2.2.c p.164). En lui proposant ces définitions, nous lui assurons de pouvoir affiner les souhaits des utilisateurs si les entités critiques proposent un nombre faible de familles et donc de choix de services. Il pourra ainsi distinguer des composants dont l'influence n'était pas, a priori, primordiale.

Utilisation des familles

La définition des familles et leur notation par l'utilisateur va permettre tout d'abord d'ordonner les familles de Sous-Groupes en fonction des vœux de celui-ci. Puis il sera possible de déterminer quelle est la famille de la configuration en cours d'exécution. Enfin, la plate-forme pourra choisir celle ayant le service le plus proche tout en augmentant la QdS. De plus, les notes intrinsèques obtenues permettent de noter et de classer les assemblages de Sous-Groupes.

Le nombre de caractéristiques à noter sera défini à partir du nombre G de fonctionnalités d'un Groupe et du nombre maximal Φ de familles pour un Sous-Groupe comme étant au maximum Φ^G. Ce nombre n'est pas rédhibitoire, d'une part, car Φ est faible puisque c'est le concepteur qui le choisit. D'autre part, G reflète la variété, la complexité, de l'application. En effet, si l'utilisateur souhaite un service complexe, il doit accepter de passer plus de temps à le paramétrer contrairement à un service simple. Toutefois le concepteur pourra proposer des paramètres par défaut qui faciliteront cette tâche.

Ainsi les familles permettent non seulement de constituer les ensembles d'étude des configurations mais encore de les ordonner grâce à l'aide de l'utilisateur. Cependant les vœux de celui-ci ne permettent pas d'ordonner les configurations au sein d'une famille. C'est en présentant notre modèle de plate-forme que nous préciserons comment nous utilisons, à nouveau, la structure de l'application pour résoudre ce problème.

3.1.3. Synthèse

La plate-forme d'exécution supervise l'application et la reconfigure pour optimiser la QdS. Pour un contexte donné nécessitant une reconfiguration, elle doit donc choisir celle qui sera implantée. Or elle ne peut définir la meilleure configuration du point de vue de la QdS (cf. §3.1.1 p.151) car ce problème est non polynomial. Nous avons donc proposé qu'elle implante une configuration meilleure dont le service soit le plus proche possible de la configuration en cours d'exécution (cf. §3.1.2.2 p.162). Si celle-ci n'est pas la meilleure, la génération d'événements de reconfiguration permettra d'améliorer la QdS par une nouvelle recherche et d'atteindre ainsi, de proche en proche, la meilleure configuration par itérations successives (cf. §3.1.2.1.a p.159). La Figure 67 page 174 illustre ce principe mis en œuvre par la plate-forme pour la recherche itérative d'optimum et qui est apparenté à la méthode du recuit simulé utilisé en particulier dans des problèmes de placement et de routage en électronique.

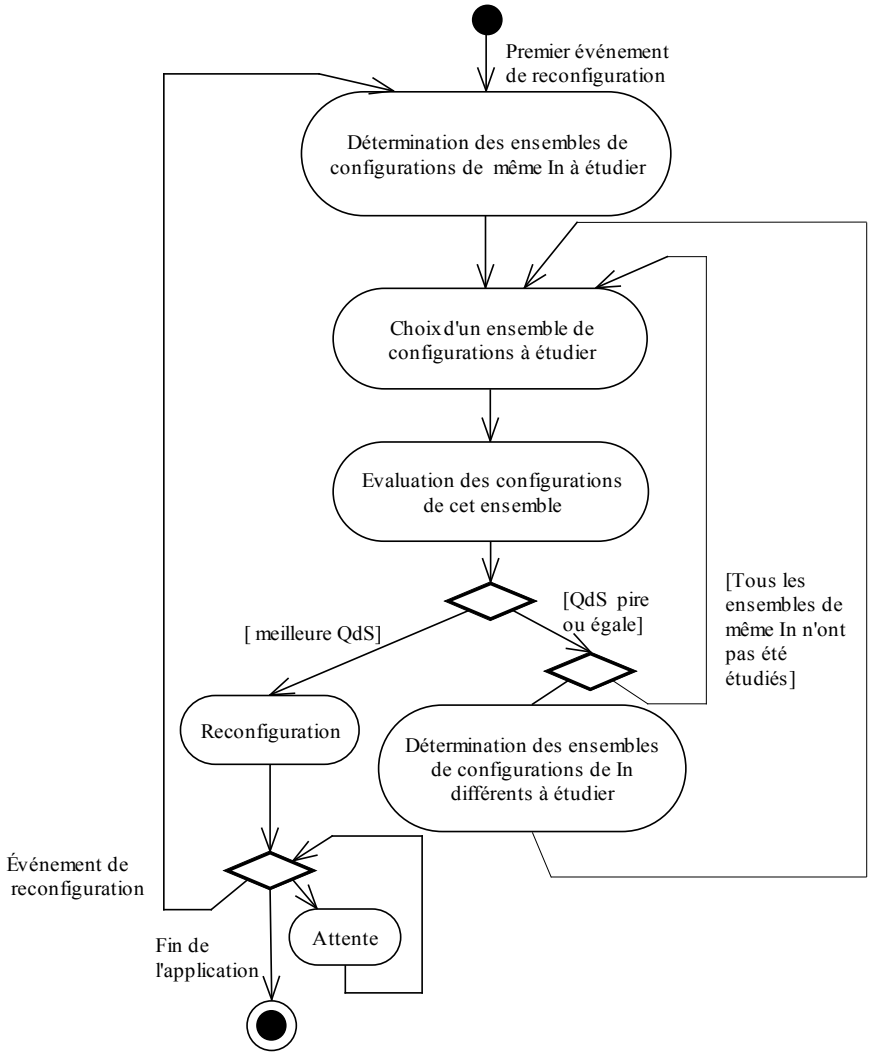


Figure 67 : Heuristique de recherche d'une meilleure configuration

La recherche d'une configuration meilleure se fait en étudiant successivement des ensembles finis de configurations ayant des services proches, les familles (cf. §3.1.2.2.d p.166). Chaque famille possède la même note de critère intrinsèque. La plate-forme recherche tout d'abord des solutions parmi les configurations de même critère intrinsèque avant d'évaluer la QdS des configurations ayant un critère intrinsèque différent (cf. §3.1.2.2.a p.163) comme dans l'exemple de la Figure 68.

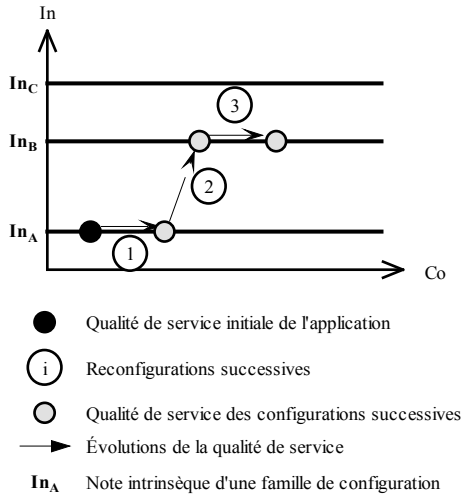


Figure 68 : Exemple d'évolution de la qualité de service

Pour être efficace, la plate-forme doit cibler les modifications à réaliser sur l'application. Or les événements de reconfiguration (cf. §3.1.2.1.d p.161) lui indiquent soit quels composants de l'application sont problématiques soit quelles entités doivent être utilisées ou supprimées. Dans le premier cas, la plate-forme va restreindre l'ensemble d'étude aux configurations qui ne diffèrent de celle en cours d'exécution que par le composant à l'origine de l'événement de reconfiguration. Dans le second cas, se pose le problème du déploiement *ex-nihilo* ou de la suppression de certaines entités.

3.2. Modèle de la plate-forme d'exécution

Pour reconfigurer l'application, la plate-forme doit trouver, à chaque itération, une configuration de meilleure QdS que celle en cours d'exécution. L'algorithme utilisé dépendra de l'entité à l'origine de la demande de reconfiguration. Ce déclencheur sera soit un composant de l'application si la donnée critique est mesurable soit un composant espion dans le cas contraire. Nous exposons ici notre modèle de plate-forme et l'heuristique qu'elle utilise avant de démontrer la validité de ce modèle par l'étude de sa complexité.

3.2.1. Reconfiguration provoquée par un composant de l'application

Nous présentons la façon dont la plate-forme trouve une configuration meilleure lorsqu'un composant de l'application pose problème, qu'il ait été identifié par un Conduit ou par un Processeur Élémentaire. L'objectif est de proposer une amélioration tout en conservant la continuité ergonomique, guidés pour cela par l'ordre établi précédemment : $C_G > C_{SG} > C_{RC} > C_{CC} > C_{RnC} > C_{CnC}$ (cf. §3.1.2.2.c p.164). Parallèlement à cet ordre, la modification du composant peut intervenir à l'un des trois niveaux suivants présentés du moins perceptible au plus perceptible :

- changement de décomposition fonctionnelle et/ou de déploiement ;
- changement de composant implantant le même rôle ;
- changement de rôle.

Puis la plate-forme peut modifier l'assemblage de Sous-Groupes. Nous proposons de définir quatre ensembles de configurations que la plate-forme évalue en partant de celui ayant le service le plus proche de celui fourni. Puis elle passe à l'ensemble suivant si elle ne trouve pas de configuration meilleure. Dans le cas contraire, elle reconfigure l'application.

3.2.1.1. Premier ensemble : changement de déploiement et d'ordonnancement

La plate-forme évalue tout d'abord les configurations de l'application ayant les mêmes composants que la configuration en cours d'exécution mais où le déploiement et l'ordonnancement du composant problématique (cf. §3.1.2.1.d p.161) sont différents. Elles n'ont donc pas obligatoirement la même décomposition fonctionnelle. Ces modifications ne sont pas directement perceptibles par l'utilisateur et nous n'avons donc aucun critère pour en ordonner l'étude. C'est pour cette raison que nous proposons de toutes les évaluer (cf. Figure 69).

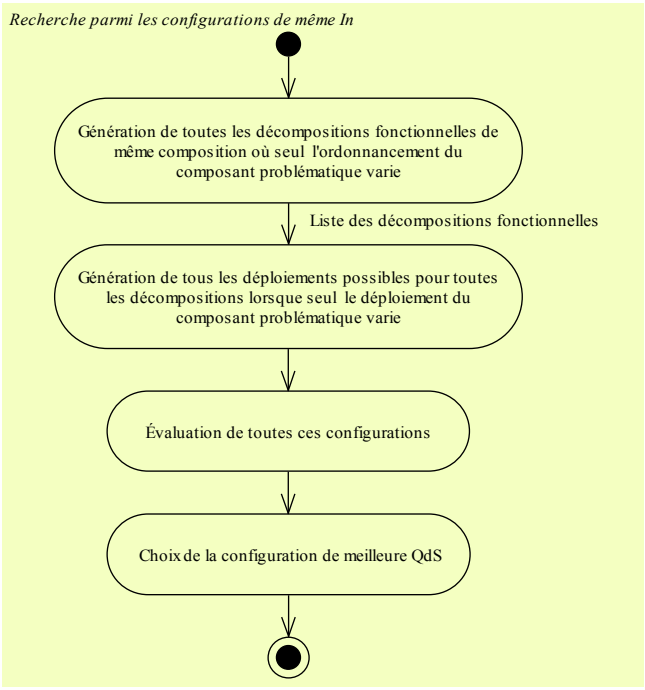


Figure 69 : Modification de l'ordonnancement et du déploiement

Ordonner différemment le composant problématique revient à le déplacer sur tous les noeuds du graphe si ce déplacement correspond à une décomposition fonctionnelle valide. Le nombre des différents ordonnancements possibles est donc borné par F nombre de décompositions fonctionnelles que nous avons établi comme étant relativement faible (cf. §3.1.1.3 p.156). Nous pouvons donc exprimer le nombre de configurations à évaluer à cette étape en fonction de F. Pour un composant, il existe $P.F$ possibilités de délocalisation avec P nombre de postes disponibles et $P^2.F$ possibilités de duplication. La complexité de cette étude C_{omE1} est donc

Formule 12 : $C_{omE1} = \mathrm{O}\left(P^2 F\right)$.

Cette complexité est acceptable puisque polynomiale. De plus, en pratique, la complexité du redéploiement sera réduite par les composants indélocalisables comme les composants matériels et par ceux dont la localisation est liée à celle d'un autre dans le cas de flux synchronisés qui doivent alors transiter par le même Conduit.

Ainsi dans l'exemple de vidéoconférence, si le composant problématique appartient au Sous-Groupe Image, le nombre de configurations à évaluer sera 96 puisque P= 4 et F=6. Il est donc tout à fait réaliste de proposer que la plate-forme évalue la totalité des configurations de cet ensemble.

3.2.1.2. Deuxième ensemble : changement de composant

Si aucune configuration du premier ensemble ne propose une QdS meilleure que celle obtenue par la configuration exécutée, la plate-forme étudie les configurations ayant les mêmes composants que l'actuelle excepté pour le composant problématique qui sera remplacé par un autre composant réalisant le même rôle. Il lui faut alors choisir parmi les *C* composants réalisant le rôle concerné.

Deux cas de figure se présentent alors. Si changer de composant implique de changer de famille, les notes intrinsèques de ces familles permettent d'ordonner l'étude en partant de la famille la plus proche de celle exécutée. La plate-forme pourra donc arrêter sa recherche dès qu'elle aura trouvé une configuration meilleure que celle en cours d'exécution. Si changer de composant se fait au sein de la même famille, les notes des familles ne peuvent donner l'ordre d'étude, c'est pourquoi nous proposons d'étudier les *C* possibilités. Une fois le composant choisi, la plate-forme doit traiter un problème identique au précédent et applique donc le même algorithme que pour le premier ensemble. La Figure 70 page 180 présente l'algorithme de recherche d'une configuration meilleure par modification de composant problématique en soulignant la recherche parmi les configurations de même note intrinsèque.

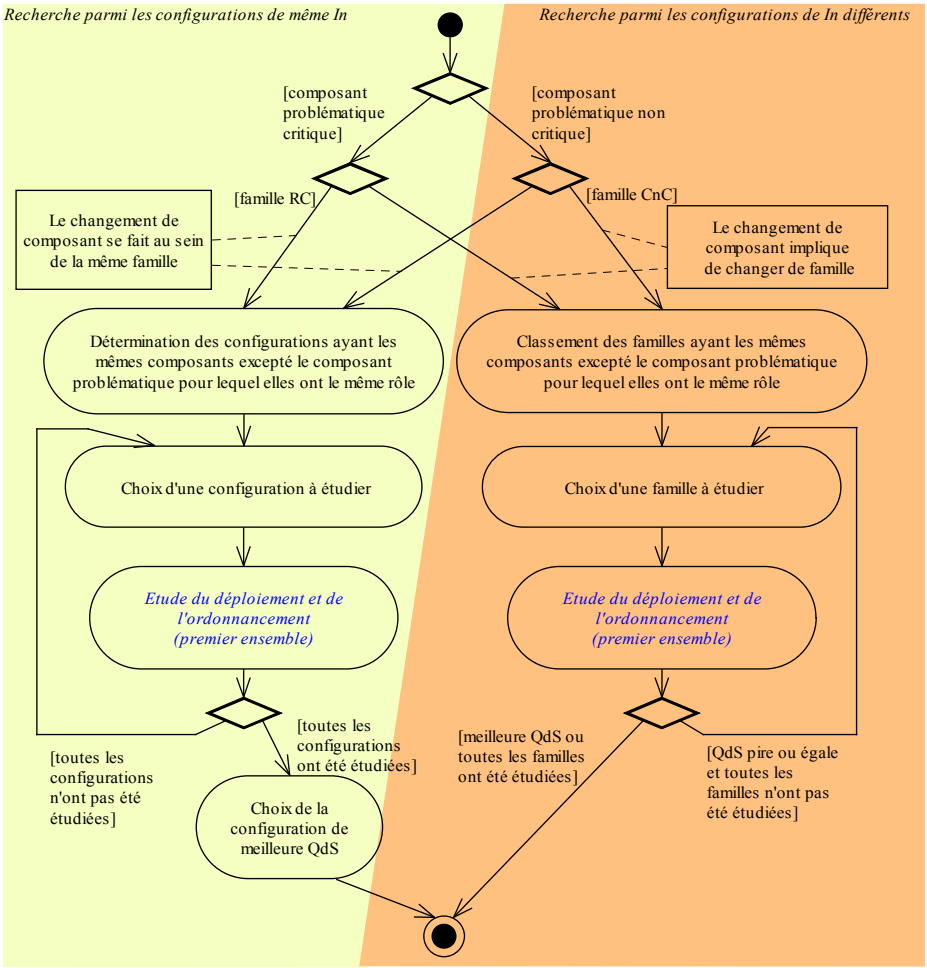


Figure 70 : Modification du composant problématique

La complexité C_{omE2} de cette étude est donnée par

Formule 13 : $C_{omE2} = \mathrm{O}\left(CP^2F\right)$.

Dans le cas de notre exemple de vidéoconférence, si le composant problématique appartient au Sous-Groupe Image, le nombre de configurations à évaluer est de 192 puisque P=4, F=6 et C=2. En pratique, cet algorithme est donc réalisable.

3.2.1.3. Troisième ensemble : changement de rôle

Si aucune configuration du deuxième ensemble ne propose une QdS meilleure que celle obtenue par la configuration exécutée, la plate-forme étudie les configurations ayant les mêmes composants que l'actuelle excepté pour le composant problématique. Pour ce dernier, nous proposons qu'elles n'utilisent pas le même rôle mais qu'elles reprennent tous les autres rôles pour lesquels elles ont les mêmes composants et les mêmes ordonnancement et déploiement. En revanche, elles peuvent utiliser de nouveaux rôles. Ces

contraintes permettent alors d'assurer une grande proximité de service entre la configuration actuelle et celles qui seront évaluées. La plate-forme change obligatoirement de décomposition fonctionnelle et doit étudier, au plus, F cas, où F est le nombre de décompositions fonctionnelles du Sous-Groupe.

À nouveau, deux cas de figure peuvent se présenter. Si supprimer le rôle problématique implique de changer de famille comme dans le cas d'un rôle critique, les notes des familles indiquent l'ordre d'étude et la plate-forme s'arrêtera dès qu'elle aura trouvé une configuration meilleure. Le choix de la famille indique donc s'il y a ou non un ou des rôles nouveaux à déployer et lesquels. Si supprimer le rôle problématique se fait au sein de la même famille, la plate-forme étudiera toutes les décompositions fonctionnelles dont le nombre — F — est relativement faible (cf. §3.1.1.3 p.156). Ce cas de figure correspond forcément à un rôle problématique non critique.

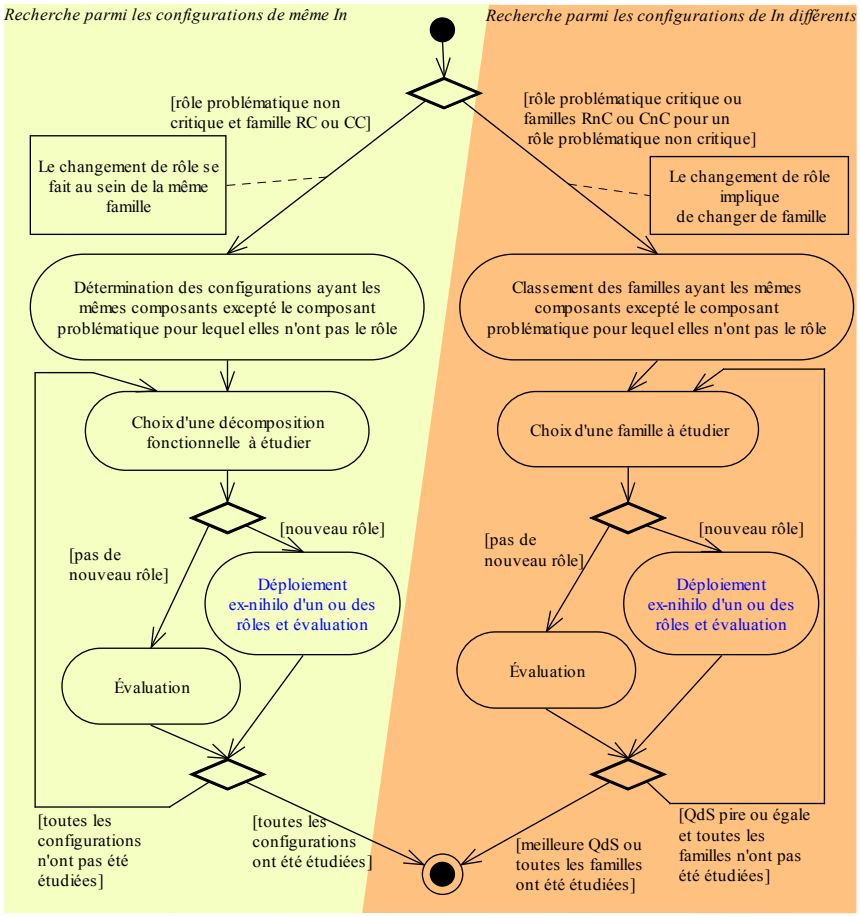


Figure 71 : Modification du rôle problématique

Dans les deux cas, la décomposition fonctionnelle peut être de deux types. Elle peut tout d'abord correspondre à la configuration en cours d'exécution dans laquelle seul le rôle problématique a été supprimé. La plate-forme peut alors évaluer sa QdS puisqu'elle connaît ses composants et leurs déploiements. Elle peut enfin nécessiter de déployer *ex-nihilo* de nouveaux rôles. Il faut alors définir comment choisir les composants qui les réalisent, comment les déployer et comment les ordonner. Ce problème très spécifique sera traité

dans le paragraphe 3.2.2 car il se pose à différents niveaux de notre étude que nous devons tout d'abord recenser pour proposer une solution globale cohérente. La Figure 71 page 181 résume alors comment la plate-forme modifie un rôle. Elle permet de vérifier que les recherches de configurations de même critère intrinsèque — fond clair — pour ce troisième ensemble succèdent bien à une recherche de même critère intrinsèque dans le deuxième ensemble (cf. Figure 70 p.180). La complexité de l'étude du troisième ensemble C_{omE3} dépend de la complexité du déploiement *ex-nihilo* de rôles C_{omR} qui sera étudiée plus loin. Elle est donnée par

Formule 14 : $C_{omE3} = F.C_{omR}$.

3.2.1.4. Quatrième ensemble : changement d'assemblage de Sous-Groupes

Si aucune configuration du troisième ensemble ne propose une QdS meilleure que celle obtenue par la configuration actuelle, la plate-forme étudie toutes les configurations du Groupe en cours d'exécution en modifiant l'assemblage de Sous-Groupes utilisé. Elle dispose pour cela au maximum des H assemblages que l'utilisateur a classé (cf. §3.1.2.2.d p.166). La plate-forme va donc évaluer l'assemblage le plus proche en service parmi ceux ne contenant pas le Sous-Groupe auquel appartient l'entité problématique. Puis elle passe au suivant si elle ne trouve pas de configuration meilleure. L'évaluation est faite en ne modifiant rien aux Sous-Groupes que cet assemblage partage avec la configuration actuelle. Pour les nouveaux Sous-Groupes, se pose à nouveau le problème de déploiement *ex-nihilo* concernant ici un Sous-Groupe en totalité et non plus seulement quelques rôles. Nous l'étudierons par la suite au paragraphe 3.2.2. La méthode de choix d'un assemblage de Sous-Groupes par la plate-forme est décrite par la Figure 72 p.183. Sa complexité C_{omE4} dépend de la complexité du déploiement *ex-nihilo* d'un Sous-Groupe C_{omSG} et elle est donnée par

Formule 15 : $C_{omE4} = H.C_{omSG}$.

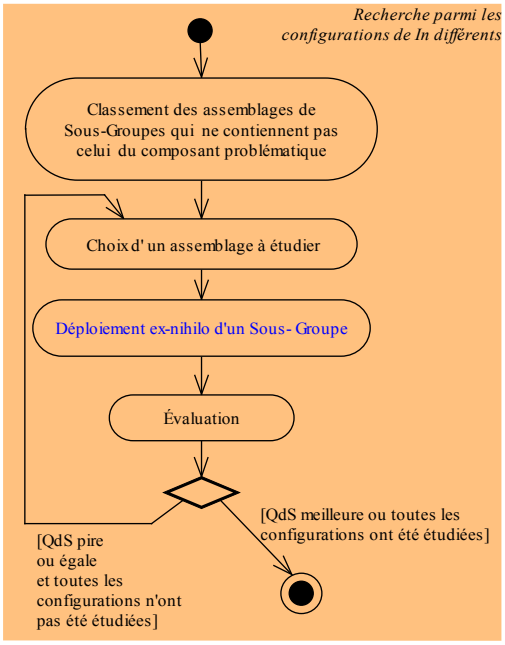


Figure 72 : Modification de l'assemblage de Sous-Groupes

3.2.1.5. Cas particulier des Conduits de synchronisation

Lorsque la reconfiguration est provoquée par un flux partageant un Conduit avec d'autres flux auxquels il est synchronisé, nous proposons que la plate-forme étudie tour à tour les différents flux du Conduit classés en fonction de la bande passante qu'ils occupent et de l'évolution souhaitée de la QdS. Elle interrompt sa recherche dès qu'elle trouve une configuration meilleure comme indiqué sur la Figure 73 page 184. La complexité de la recherche est alors multipliée par *Fl* qui représente le nombre maximal de flux dans un Conduit.

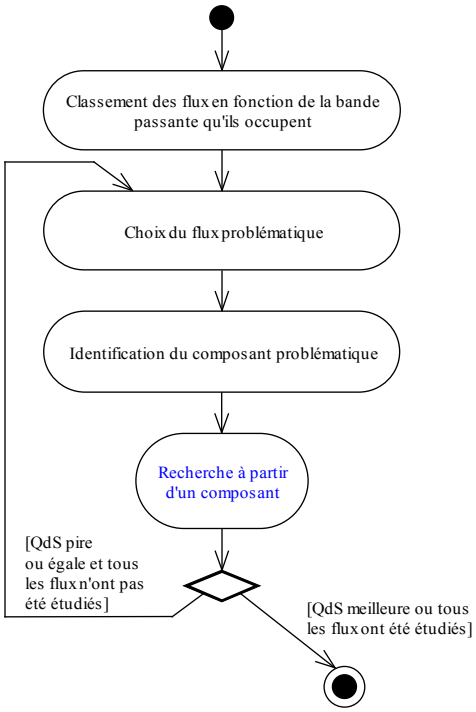


Figure 73 : Recherche à partir d'un événement issu d'un Conduit

3.2.1.6. Algorithme et complexité

L'algorithme global qui permet à la plate-forme de chercher une configuration meilleure lorsque c'est un composant qui est à l'origine de la reconfiguration est donné par la Figure 74 page 185. Sa complexité C_{om1} est la complexité maximale des différents ensembles (cf. Formule 12 p.178) (cf. Formule 13 p.180) (cf. Formule 14 p.182) (cf. Formule 15 p.182). Elle est donc définie par :

$$C_{om1} = Fl.\max\left(\mathrm{O}\left(P^2F\right);\mathrm{O}\left(CP^2F\right);FC_{omR};HC_{omSG}\right)$$

Formule 16 : $$\boxed{C_{om1} = \max\left(\mathrm{O}\left(Fl.CP^2F\right);Fl.F.C_{omR};Fl.H.C_{omSG}\right)}$$

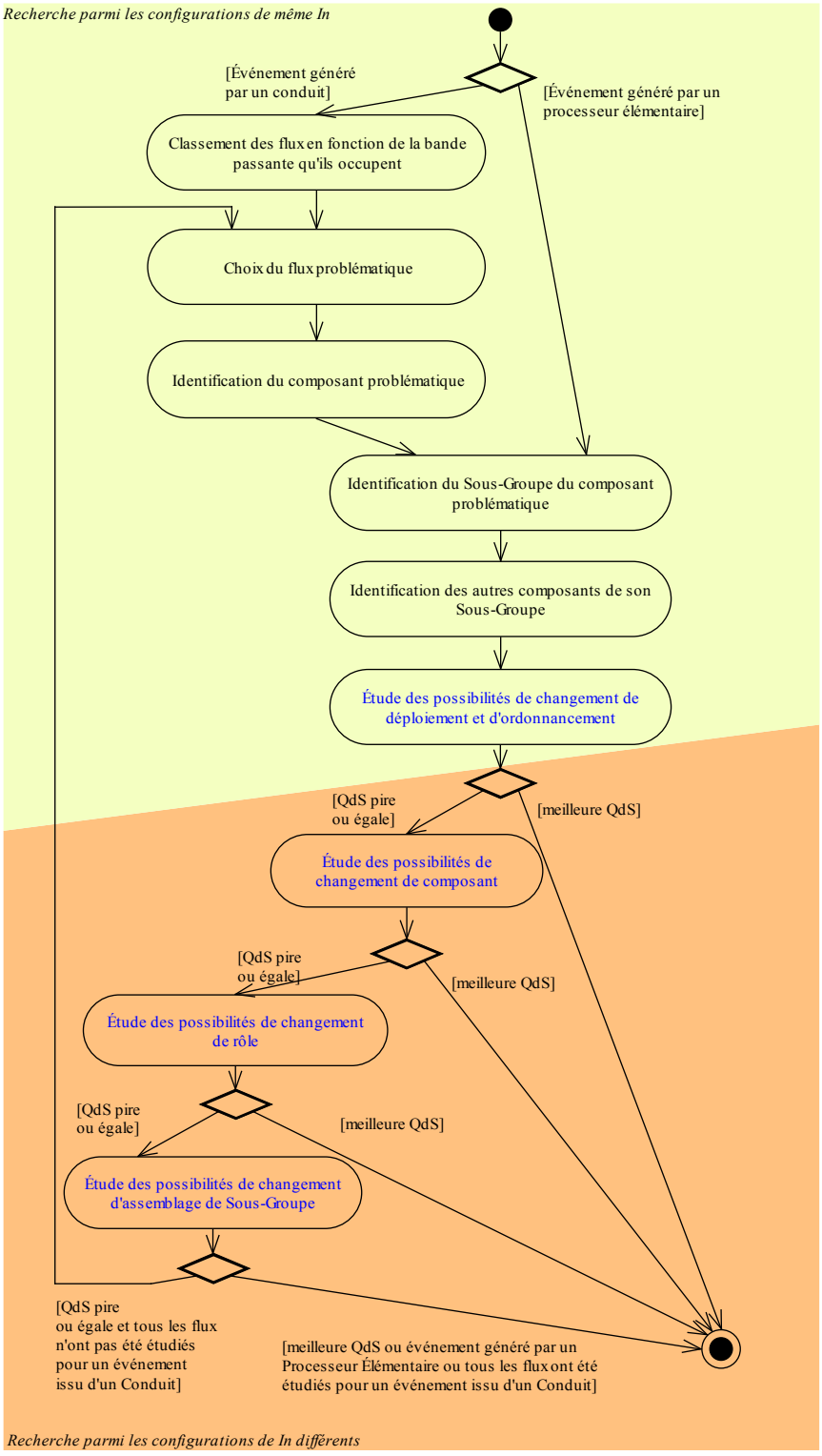


Figure 74 : Recherche provoquée par un composant de l'application

3.2.2. Reconfiguration provoquée par un composant espion

Lorsque ce sont les composants espions qui génèrent un événement de reconfiguration, ils peuvent soit demander la suppression soit imposer l'utilisation d'une entité. La suppression d'une entité ne pose pas de problème puisque la plate-forme supprime alors tous les composants de l'entité que celle-ci ne partage avec aucune autre. En revanche, les composants espions peuvent imposer le choix d'un Groupe — comme lors de l'arrivée d'un nouvel utilisateur dans l'application — ou d'un Sous-Groupe — comme lorsqu'il est nécessaire de fournir une fonctionnalité particulière — ou enfin d'un rôle — comme lorsqu'un traducteur est nécessaire. Dans tous les cas de figure, il s'agit de déployer *ex-nihilo* un Groupe ou une entité de l'application. Nous devons donc préciser comment la plate-forme s'acquitte de cette tâche d'autant plus que ce déploiement intervient également dans la recherche d'une configuration meilleure provoquée par un composant de l'application. Nous allons présenter tout d'abord les principes de déploiement *ex-nihilo* utilisés avant de décrire le déploiement d'un Groupe, d'un Sous-Groupe et de rôles.

3.2.2.1. Politique de déploiement

Le déploiement *ex-nihilo* consiste à déployer un Groupe, un Sous-Groupe, ou un rôle lorsque la configuration en cours d'exécution ne les utilise pas. Ils n'ont donc aucune réalité physique et, en particulier, pas de lien avec l'entité problématique. Nous ne disposons alors d'aucun critère de déploiement a priori. Nous allons donc définir avec soin les politiques utilisées pour ce déploiement car elles ont de grandes conséquences sur la convergence de l'algorithme. En effet, elles constituent une initialisation de la recherche par itération.

Ce problème peut être comparé aux problèmes de prédiction rencontrés dans la gestion des mémoires caches ou dans la prédiction des branchements. Dans les deux cas, il s'agit de faire un choix en réponse à un contexte donné en espérant qu'il soit le plus durable possible. Dans le cas des mémoires caches, le problème est de choisir un bloc de mémoire à libérer pour le réutiliser en faisant en sorte que ce choix perturbe le moins possible le fonctionnement en cours. Les deux méthodes utilisées se révèlent à peu près équivalentes.

La première, nommée L.R.U. — *Least Recently Used* — consiste à considérer que le contexte va probablement peu varier et que, par conséquent, le bloc de mémoire le moins récemment utilisé est probablement celui qui a le plus de chance de ne plus être utilisé du tout. La transposition de cette méthode à notre problème nous amènerait à choisir la dernière solution de déploiement qui a été implantée.

La seconde, nommée L.F.U. — *Least Frequently Used* — consiste à choisir la solution qui risque de poser le moins de problèmes. Elle consiste alors à remplacer le bloc de mémoire le moins souvent utilisé en supposant que c'est donc celui qui fera le moins souvent défaut s'il vient à manquer. La transposition de cette méthode à notre problème reviendrait à proposer le déploiement le plus souvent implanté en misant sur le fait que c'est celui qui donne visiblement le plus souvent satisfaction.

Or, dans notre cas, il est impossible de miser sur une stabilité à long terme du contexte et en particulier du réseau. Nous ne pouvons donc pas choisir la première solution

implantée. Nous pourrons en revanche utiliser des solutions de type LFU ce qui nous amènera à choisir la solution la plus fréquemment utilisée.

Cependant, lorsque l'entité à déployer n'a jamais été implantée, la méthode préconisée ne peut être utilisée faute de modèle à suivre. Nous pouvons alors opter pour une politique consistant à choisir de déployer :

- soit la configuration offrant la meilleure QdS possible ;
- soit la configuration offrant la moins bonne qualité ;
- soit la configuration offrant une qualité médiane.

La dernière proposition est à exclure puisque la complexité du problème — NP-complet — interdit de définir la médiane des qualités de service. Déployer la meilleure solution permet d'atteindre directement la qualité optimale si le contexte est bon. En revanche, dans le cas contraire, le risque est grand de perturber le service des autres utilisateurs. De plus, il nous paraît bien peu ergonomique de montrer quelque chose d'excellent à l'utilisateur pour le lui enlever aussitôt. Déployer la pire solution permet de s'assurer que l'utilisateur disposera d'un service qui fonctionne dès le départ ou qui ne fonctionnera de toute façon jamais. De plus, ce choix est celui qui perturbe le moins le fonctionnement des services des autres utilisateurs. Partant d'une mauvaise qualité, la plate-forme va rapidement améliorer le service quitte à utiliser un écran d'attente jusqu'à la stabilisation de la qualité. C'est pourquoi nous choisissons de déployer la pire solution. Cette idée nous encourage à partir de la solution à la fois la plus simple en écartant toute duplication de composant et à la fois la pire du point de vue de la QdS c'est-à-dire celle ayant la note intrinsèque la plus faible.

Ce choix se fait à partir des notes dont la plate-forme dispose uniquement au niveau des familles de configurations. Pour les autres niveaux, nous proposons que le concepteur définisse une configuration par défaut que la plate-forme utilisera faute de disposer d'autres critères. Nous allons présenter le fonctionnement de la plate-forme pour les différents cas de déploiement en partant de la granularité la plus grande pour aller vers la plus faible.

3.2.2.2. Déploiement *ex-nihilo* d'un Groupe

Pour déployer un Groupe, la plate-forme choisit le pire assemblage de Sous-Groupes et implante chacun d'entre eux (cf. Figure 75 p.188). La complexité de cette étape C_{omG} se déduit donc de la complexité du déploiement d'un Sous-Groupe C_{omSG} et du nombre maximal de Sous-Groupes dans un Groupe G :

Formule 17 : $C_{omG} = GC_{omSG}$.

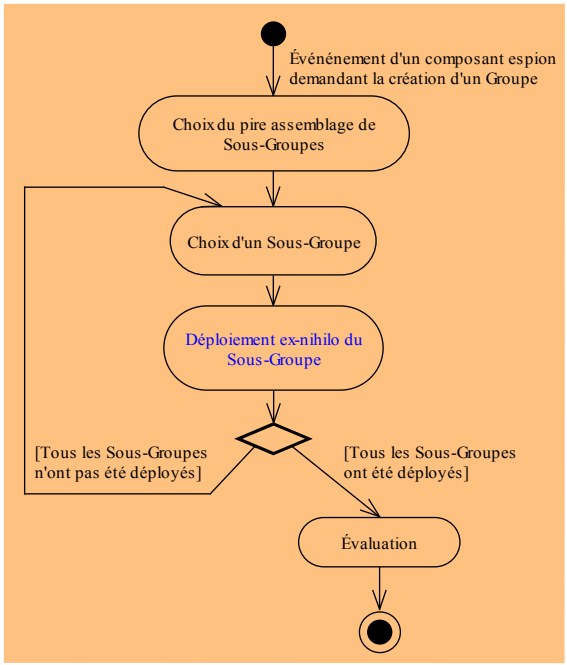


Figure 75 : Déploiement *ex-nihilo* d'un Groupe

3.2.2.3. Déploiement *ex-nihilo* d'un Sous-Groupe

Pour déployer un Sous-Groupe, les notes des familles de configurations permettent à la plate-forme de choisir la pire famille. En fonction du niveau de définition de celles-ci, seront alors connus :

- soit les composants du Sous-Groupe ;
- soit les composants critiques et les rôles mais pas les composants non critiques ;
- soit les composants critiques mais pas les rôles non critiques ;
- soit les rôles critiques mais ni les composants critiques ni les rôles non critiques.

Dans le premier cas, la plate-forme doit effectuer une étude du type de celle occasionnée par une reconfiguration liée à un composant quand elle recherche un déploiement et un ordonnancement (cf. §3.2.1.1 p.177). Cependant, l'absence de composant problématique ne permet pas d'utiliser le même algorithme pour ce problème non polynomial. Dans le second cas, la plate-forme se heurte à un problème semblable à celui du changement d'un composant (cf. §3.2.1.2 p.179). En revanche, comme il lui faut déployer plusieurs rôles, le problème est non polynomial et l'algorithme utilisé précédemment n'est pas adapté. Pour les deux derniers cas, le problème est également non polynomial.

Il est donc nécessaire d'utiliser les politiques de déploiement définies précédemment (cf. §3.2.2.1 p.177) pour traiter le déploiement d'un Sous-Groupe. Si la famille choisie a déjà été implantée, la plate-forme utilise le déploiement et la décomposition fonctionnelle le plus souvent utilisés et le problème se ramène au déploiement d'un ou de plusieurs rôles.

Si la famille n'a jamais été utilisée, la plate-forme implante la configuration par défaut qui précisera, en fonction du niveau de définition des familles :

- soit la décomposition fonctionnelle ;
- soit les composants non critiques et la décomposition fonctionnelle ;
- soit les rôles et composants non critiques et la décomposition fonctionnelle ;
- soit enfin les composants critiques, les rôles et composants non critiques et la décomposition fonctionnelle.

L'utilisation de la configuration par défaut permet donc de ramener le déploiement d'un Sous-Groupe complet à une unique problématique constituée par le déploiement d'un Sous-Groupe lorsque ses composants et sa décomposition fonctionnelle sont connus (cf. Figure 76). Bien entendu, étant donné que le concepteur ne connaît pas les postes utilisés par l'application, ceux-ci ne peuvent pas être définis dans la configuration par défaut.

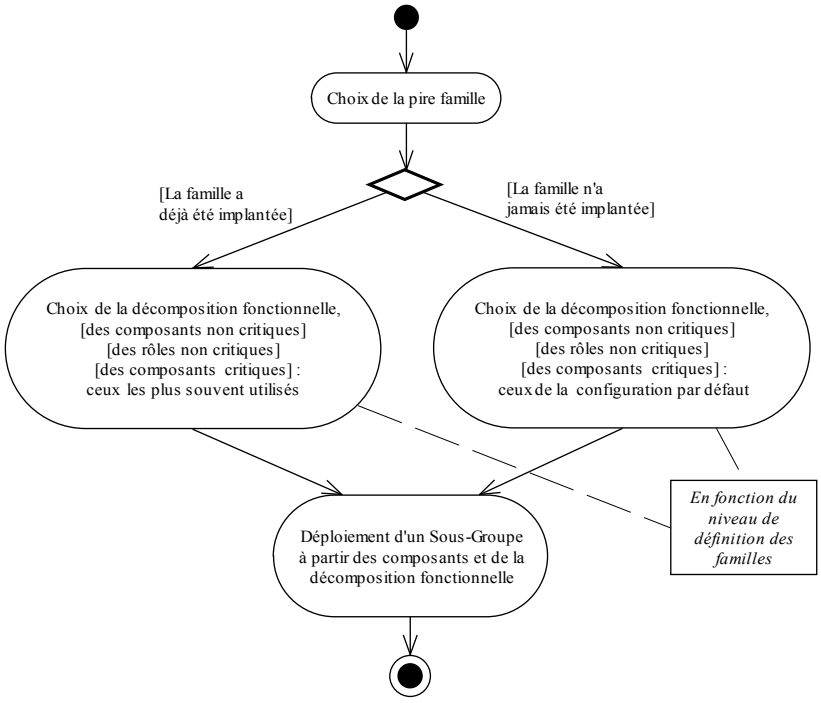


Figure 76 : Déploiement *ex-nihilo* d'un Sous-Groupe

La plate-forme doit alors définir la localisation et la possible duplication des composants. Si le Sous-Groupe a déjà été implanté, elle les choisit conformes au déploiement le plus souvent utilisé. Si ce n'est pas le cas, la plate-forme écarte toute duplication de manière à proposer la configuration la plus simple possible (cf. §3.2.2.1 p.186). Puis elle place les composants en cherchant à minimiser la charge du réseau induite par l'application et la charge des postes. Or tout Sous-Groupe contient un composant observable qui est sur le poste de l'utilisateur (cf. §2.2.3.3 p.105). Deux cas se présentent alors :

- tous les autres composants du Sous-Groupe peuvent être placés sur un poste quelconque et la plate-forme propose de les positionner sur le poste de l'utilisateur excepté si celui-ci est saturé. Elle les déploiera alors sur le poste le moins chargé par l'application.

- le Sous-Groupe contient des composants qui doivent être placés sur des postes précis autres que celui de l'utilisateur. Le déploiement peut alors se fractionner en plusieurs étapes consistant à placer une suite de composants entre deux composants indélocalisables. De manière à minimiser les transferts sur le réseau, la plate-forme choisit de placer les autres composants uniquement sur les deux postes des composants contraints. Elle positionne alors les liaisons sur le réseau de manière à minimiser le débit c'est-à-dire qu'elle les place là où les flux sont les plus légers. Ainsi, de proche en proche, en partant du composant observable, sont positionnés tous les composants du Sous-Groupe.

L'algorithme réalisé par la plate-forme pour déployer un Sous-Groupe est défini par la Figure 77 page 191. Lorsque ce déploiement est imposé par un événement issu d'un composant espion, le comportement de la plate-forme est décrit par la Figure 78 page 192. Il utilise les mêmes principes que l'algorithme utilisé pour rechercher une configuration en modifiant l'assemblage de Sous-Groupes lorsque l'événement de reconfiguration est provoqué par un composant de l'application (cf. §3.2.1.4 p.182).

Quel que soit le contexte de sa mise en œuvre, la complexité du déploiement d'un Sous-Groupe C_{omSG} par notre algorithme est liée au nombre de composants c'est-à-dire au nombre de rôles atomiques — R — que peut contenir un Sous-Groupe :

Formule 18 : $C_{omSG} = \mathrm{O}(R)$.

Dans le cas de la vidéoconférence que nous avons évoquée précédemment (cf. §2.1.2.1 p.84) (cf. §2.3.4 p.136), $R=6$ (cf. Tableau 12 p.156.) et donc le nombre de configurations à évaluer est de 6. La complexité est donc faible et ne pose pas de problème de mise en œuvre.

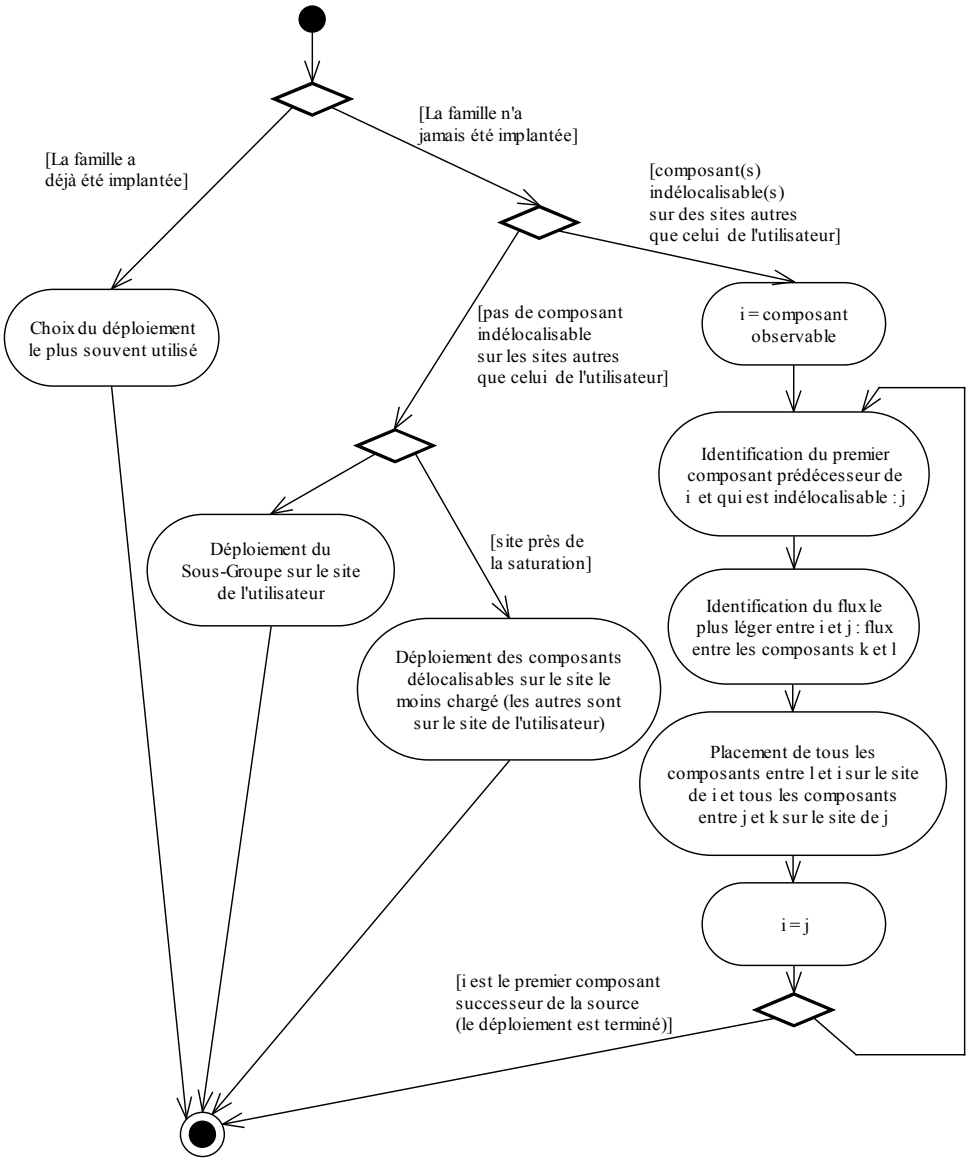


Figure 77 : Déploiement d'un Sous-Groupe à partir des composants et de la décomposition fonctionnelle

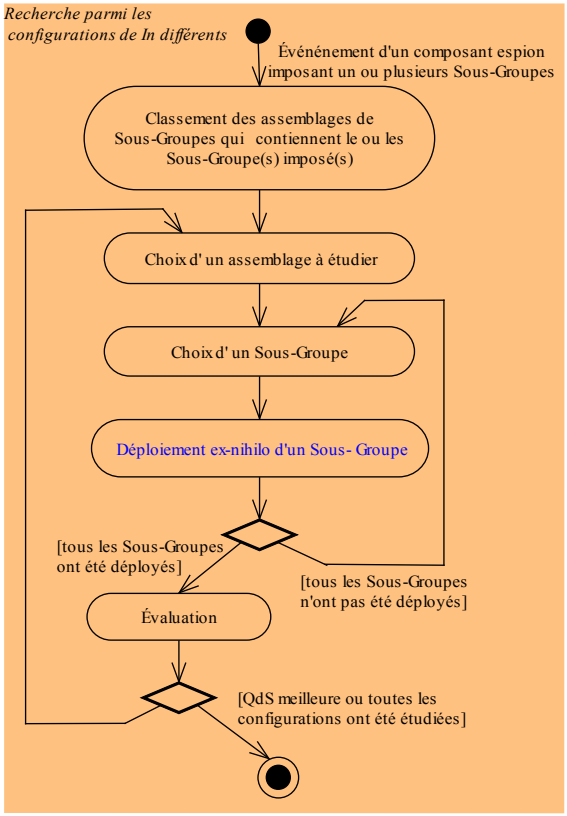


Figure 78 : Recherche provoquée par un composant espion lorsque l'événement impose un ou plusieurs Sous-Groupes

3.2.2.4. Déploiement *ex-nihilo* d'un ou de plusieurs rôles

La plate-forme doit déployer *ex-nihilo* un ou plusieurs rôles lorsqu'ils sont imposés par un événement généré par un composant espion ou lorsqu'elle recherche une configuration meilleure en modifiant le rôle d'un composant à l'origine d'une reconfiguration (cf. §3.2.1.3 p.180). La difficulté de ce déploiement vient du fait que le rôle à déployer n'a pas de lien avec le composant problématique et que la configuration par défaut n'est pas toujours utilisable. En effet, la configuration recherchée doit conserver les composants non problématiques de la configuration en cours d'exécution et modifier uniquement la décomposition fonctionnelle : rien n'assure que la configuration par défaut respecte ces conditions.

Comme précédemment, l'algorithme proposé dépend du niveau de définition des familles puisque, selon les cas, il doit permettre de choisir le déploiement, la décomposition fonctionnelle et les nouveaux composants non critiques. Le concepteur ne peut indiquer pour chaque rôle atomique un composant par défaut car ils sont trop nombreux. De manière à être cohérent avec notre politique de déploiement (cf. §3.2.2.1 p.186), nous proposons que la plate-forme choisisse le composant le plus utilisé si le rôle a déjà été implanté et le composant consommant le moins de ressources du système dans le cas contraire. Nous

utiliserons dans ce dernier cas les caractéristiques des composants fournies par leur concepteur pour l'identifier.

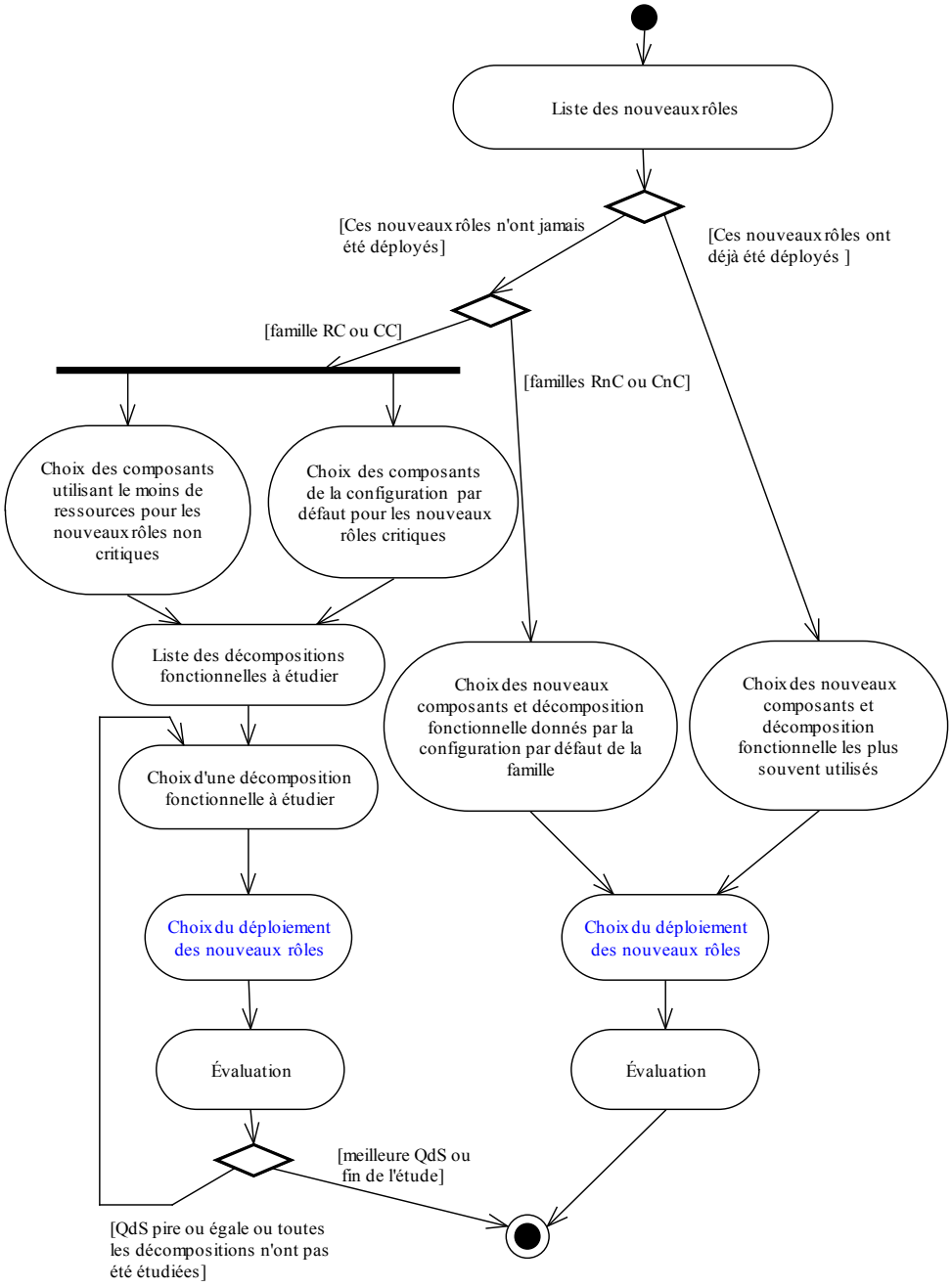
La complexité du déploiement *ex-nihilo* d'un ou de plusieurs rôles C_{omR} est définie à partir du nombre de décompositions fonctionnelles F et du nombre maximal de composants dans un Sous-Groupe R par

Formule 19 : $C_{omR} = \mathrm{O}(F.R)$ **.**

Dans le cas de la vidéoconférence (cf. §2.1.2.1 p.84) (cf. §2.3.4 p.136), F=6 et R=6 (cf. Tableau 12 page 156), nous avons donc 36 configurations à étudier ce qui correspond à une complexité relativement faible.

L'algorithme réalisé par la plate-forme est décrit par la Figure 79 page 194 et la Figure 80 page 195. Sa mise en œuvre dans le cas où le déploiement est imposé par un événement généré par un composant espion est précisée par la Figure 81 page 196. Elle repose sur les mêmes principes que l'algorithme utilisé pour rechercher une configuration en modifiant le rôle du composant problématique lorsque l'événement de reconfiguration est provoqué par un composant de l'application (cf. §3.2.1.3 p.180). Cependant les composants espions ne sont associés qu'à des caractéristiques intrinsèques de service que l'utilisateur peut percevoir (cf. §3.1.2.1.d p.161). Les rôles à déployer sont donc uniquement des rôles critiques et la recherche d'une configuration meilleure se fait en changeant de famille de configurations.

Ces algorithmes achèvent de décrire la manière dont la plate-forme déploie les Groupes, Sous-Groupes et composants. Sa plus grande originalité réside dans le fait que le déploiement de l'application à l'arrivée d'un utilisateur n'est rien d'autre qu'une reconfiguration provoquée par un composant espion qui demande le déploiement *ex-nihilo* d'un Groupe. Ce n'est donc pas un traitement particulier contrairement à d'autres travaux comme CADeComp [AYE 05] (cf. §1.2.3.4 p.55).



Famille RC : famille des configurations ayant les mêmes rôles critiques

Famille CC : famille des configurations ayant les mêmes rôles critiques et les mêmes composants critiques

Famille RnC : famille des configurations ayant les mêmes rôles critiques et non critiques et les mêmes composants critiques

Famille CnC : famille des configurations ayant les mêmes rôles critiques et non critiques et les mêmes composants critiques et non critiques

Figure 79 : Déploiement *ex-nihilo* d'un ou de plusieurs rôles

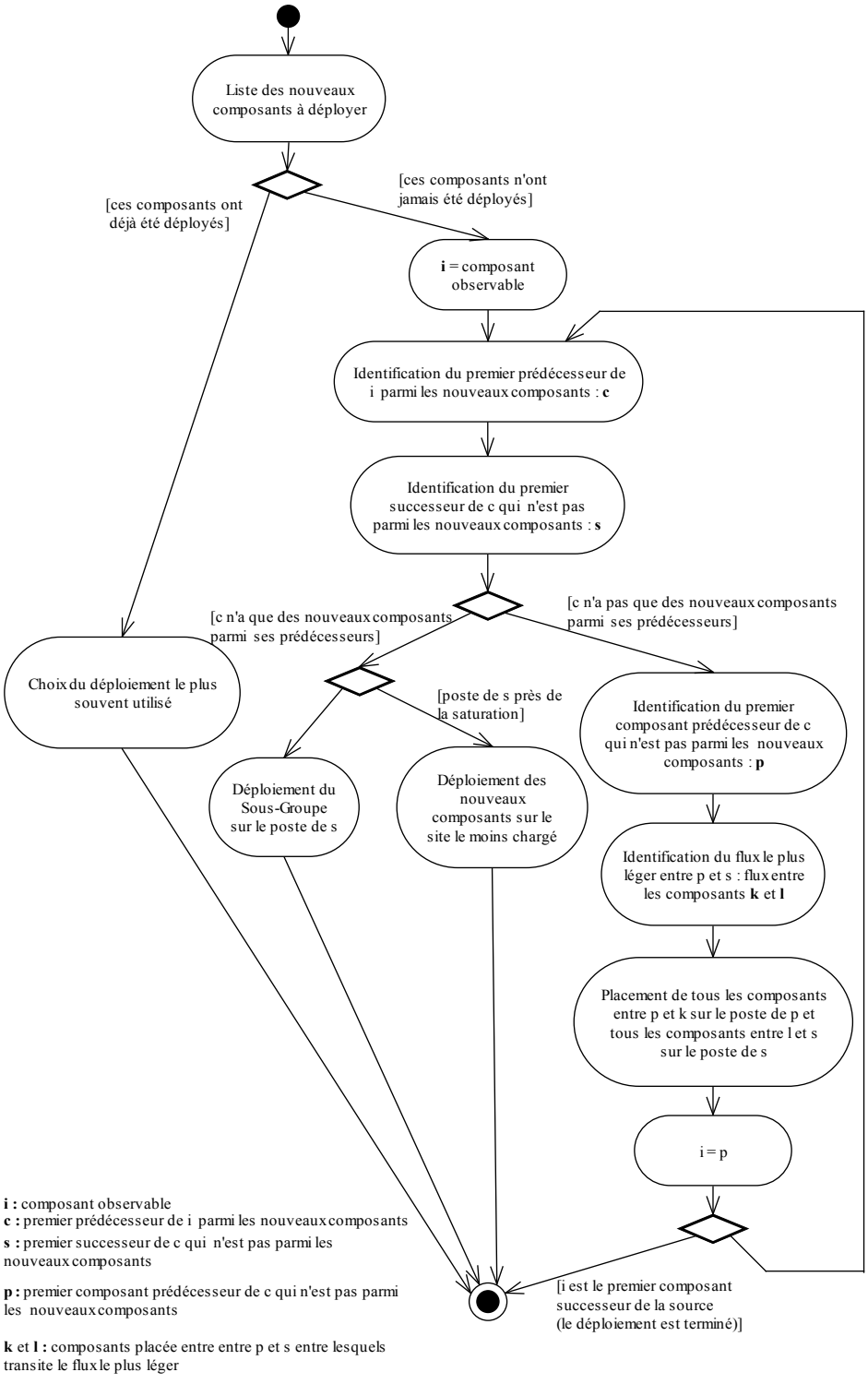


Figure 80 : Choix du déploiement des nouveaux rôles

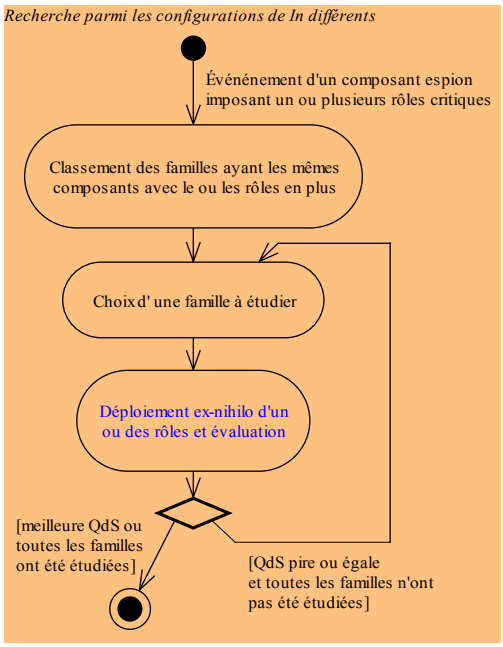


Figure 81 : Recherche provoquée par un composant espion lorsque l'événement impose un ou plusieurs rôles

3.2.2.5. Algorithme et complexité

L'algorithme global qui permet à la plate-forme de chercher une configuration meilleure lorsque c'est un composant espion qui est à l'origine de la reconfiguration est donné par la Figure 82 page 197. Sa complexité C_{om2} est la complexité maximale du déploiement :

- d'un Groupe $C_{omG} = GC_{omSG}$ (cf. Formule 17 p.187) où G est le nombre maximal d'instances de Sous-Groupes que peut contenir une instance de Groupe ;
- d'un Sous-Groupe $C_{omSG} = \mathrm{O}(R)$ (cf. Formule 18 p.190) où R est le nombre maximal de rôles atomiques pouvant intervenir dans la décomposition fonctionnelle d'un Sous-Groupe ;
- d'un ou de plusieurs rôles $C_{omR} = \mathrm{O}(F.R)$ (cf. Formule 19 p.193) où F est le nombre maximal de décompositions fonctionnelles possibles pour un Sous-Groupe.

Elle est définie par

Formule 20 : $\boxed{C_{om2} = \max\left(\mathrm{O}(G.R);\mathrm{O}(F.R)\right)}$.

Dans le cas de la vidéoconférence qui nous sert d'illustration (cf. §2.1.2.1 p.84) (cf. §2.3.4 p.136), G=3, F=6 et R=6 *(cf.* Tableau 12 p156.), 36 configurations doivent être évaluées ce qui correspond à une complexité relativement faible.

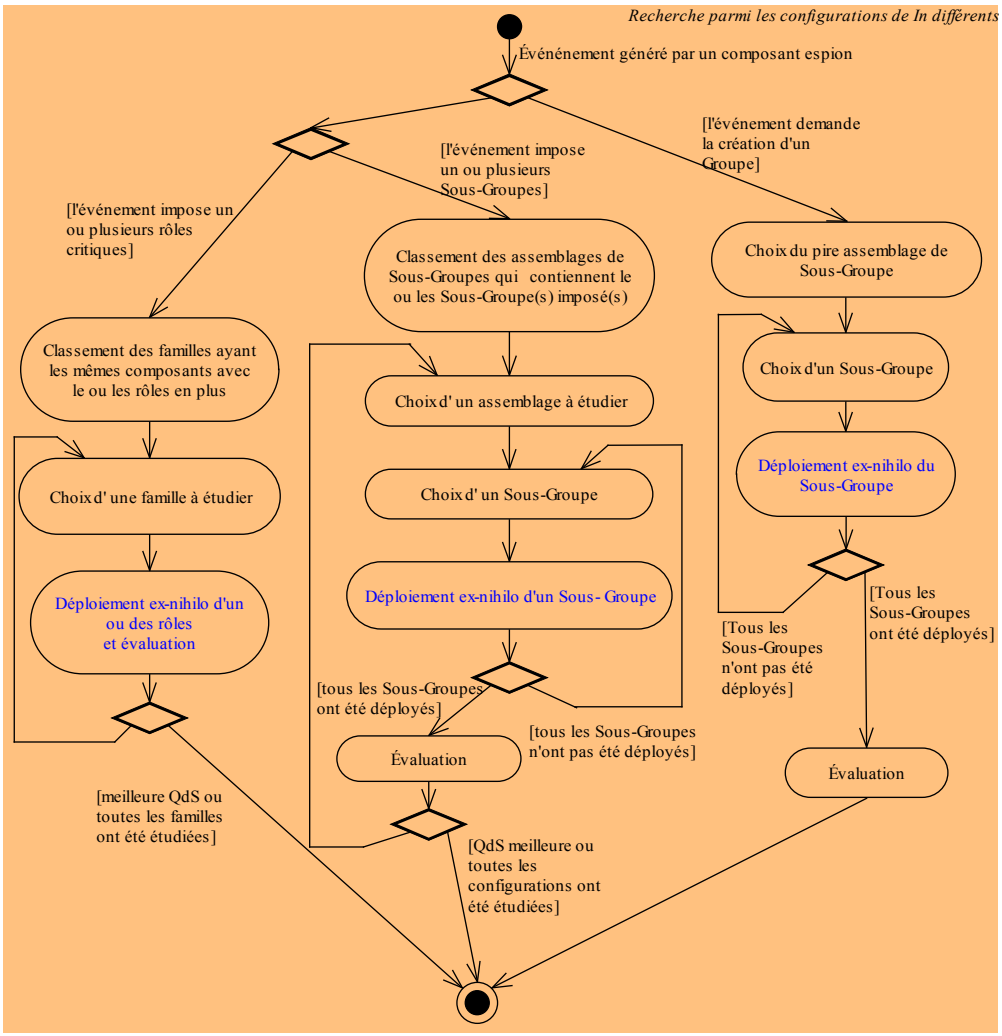


Figure 82 : Recherche provoquée par un composant espion

3.2.3. Complexité algorithmique de l'optimisation de la qualité de service

Notre algorithme de recherche d'une configuration meilleure est maintenant entièrement défini si bien que nous pouvons comparer sa complexité avec celle — non polynomiale — du problème initial permettant de définir la meilleure configuration.

3.2.3.1. Complexité obtenue pour la recherche d'une configuration meilleure

Nous avons déterminé la complexité de la recherche d'une configuration meilleure lorsqu'un composant de l'application a généré l'événement à l'origine de la reconfiguration comme C_{om1} telle que

$$C_{om1} = \max\left(\mathrm{O}\left(Fl.C.P^2.F\right); Fl.F.C_{omR}; Fl.H.C_{omSG}\right) \text{ (cf. Formule 16 page 184)}$$

où

Fl est le nombre maximal de flux présents dans un Conduit,
C le nombre maximal de composants réalisant un rôle atomique,
P le nombre maximal de postes,
F le nombre maximal de décompositions fonctionnelles possibles pour un Sous-Groupe,
H le nombre maximal d'assemblages différents de Sous-Groupes qui peuvent réaliser un Groupe donné,
C_{omR} la complexité du déploiement *ex-nihilo* d'un ou de plusieurs rôles,
C_{omSG} la complexité du déploiement d'un Sous-Groupe.

Nous avons ensuite évalué la complexité de la recherche d'une configuration meilleure lorsqu'un composant espion est à l'origine de la reconfiguration comme C_{om2} telle que

$$C_{om2} = \max\left(\mathrm{O}(G.R); \mathrm{O}(F.R)\right) \text{ (cf. Formule 20 page 196)}$$

où

G est le nombre maximal de Sous-Groupes que peut contenir un Groupe,
R le nombre maximal de rôles atomiques pouvant intervenir dans la décomposition fonctionnelle d'un Sous-Groupe.

La complexité algorithmique de l'optimisation de la QdS C_{om} peut donc être définie comme la complexité maximale de ces deux cas de figure soit

$$C_{om} = \max\left(C_{om1}; C_{om2}\right)$$

Formule 21 : $$\boxed{C_{om} = \max\left(\mathrm{O}\left(Fl.C.P^2.F\right); \mathrm{O}\left(Fl.F^2.R\right); \mathrm{O}(Fl.H.R); \mathrm{O}(G.R)\right)}$$

Dans le cas de la vidéoconférence qui nous sert d'illustration, *Fl*=2, *C*=2, P=4, *F*=6, *R*=6, *H=2, G*=3 (cf. Tableau 12 page 156), 432 configurations doivent être étudiées. Cette complexité est à comparer avec la complexité initiale.

3.2.3.2. Comparaison avec le problème initial

L'heuristique présente donc une complexité polynomiale ce qui assure que la plate-forme puisse l'implanter. Cette complexité doit être comparée avec celle de la définition de toutes les configurations nécessaire à une évaluation exhaustive des qualités de service possibles : $\left\{\left\lceil (C.P)^R F \right\rceil^G H\right\}^U$ (cf. Formule 11 p.155) avec *U* le nombre maximal d'utilisateurs simultanés de l'application. Le terme *U* n'apparaît plus car nous proposons de reconfigurer uniquement le Groupe dont le service est problématique et qui est identifié

grâce aux événements. Les exposants *R* et *G* n'apparaissent plus sous cette forme pour la même raison.

La disparition de ces termes signifie que l'application va pouvoir fonctionner quel que soit le nombre d'utilisateurs et quel que soit le nombre de fonctionnalités proposées à chacun c'est-à-dire quelle que soit la complexité de l'application. Les termes qui demeurent ou qui sont apparus traduisent :

- soit la complexité intrinsèque de l'application qui ne peut être réduite : *P* nombre de postes, *R* nombre rôles de la décomposition, *Fl* nombre de flux synchronisés, *G* nombre de fonctionnalités ;
- soit l'étendue de l'offre de QdS : *C* nombre de composants utilisables pour réaliser un rôle, *F* nombre de décompositions fonctionnelles donc nombre de façons de rendre un service.

Le concepteur devra donc réaliser un compromis entre la rapidité de l'heuristique et l'étendue de l'offre de QdS.

En pratique, cette complexité sera toujours réduite par les éléments rappelés au paragraphe 3.1.1.3. Le Tableau 17 page 200 permet de comparer numériquement la complexité du nombre maximal de configurations étudiées lors d'une itération au facteur *Fl* près — colonne "heuris" dans le tableau — avec la complexité du nombre de configurations possibles — colonne "max". Pour un Sous-Groupe, les configurations étudiées dans l'heuristique correspondent à l'étude des ensembles 1, 2 et 3 (cf. §3.2.1 p.177) ou aux configurations étudiées pour un déploiement *ex-nihilo* (cf. §3.2.2.3 p. 188) si bien que la complexité du déploiement d'un Sous-Groupe est donnée au facteur *Fl* près par $\max\left(\mathrm{O}\left(P^2.F\right);\mathrm{O}\left(C.P^2.F\right);\mathrm{O}\left(F^2.R\right);\mathrm{O}\left(R\right)\right)$.

Pour un Groupe, les configurations étudiées dans l'heuristique correspondent à l'étude des ensembles 1, 2, 3 et 4 (cf. §3.2.1 p.177) ou aux configurations étudiées pour un déploiement *ex-nihilo* (cf. §3.2.2.3 p.en page 188) si bien que la complexité du déploiement d'un Sous-Groupe est donnée au facteur *Fl* près par $\max\left(\mathrm{O}\left(P^2.F\right);\mathrm{O}\left(C.P^2.F\right);\mathrm{O}\left(F^2.R\right);\mathrm{O}\left(H.R\right);\mathrm{O}\left(G.R\right)\right)$.

Pour le déploiement de l'application, la complexité au facteur *Fl* près est donnée par $C_{om}\big|_{Fl=1} = \max\left(\mathrm{O}\left(C.P^2.F\right);\mathrm{O}\left(F^2.R\right);\mathrm{O}\left(H.R\right);\mathrm{O}\left(G.R\right)\right)$ qui est identique à la complexité du déploiement d'un Groupe puisque la plate-forme reconfigure l'application Groupe par Groupe.

Cette application numérique nous permet de vérifier que l'heuristique modifie fondamentalement l'ordre de grandeur des résultats indépendamment de la valeur de *Fl*. Ainsi la complexité de la reconfiguration de l'application est ramenée à celle de la reconfiguration d'un Sous-Groupe ou d'un Groupe même si le nombre de configurations étudiées dans chaque cas ne sera pas le même.

Nombre de	1er cas		2ème cas		3ème cas		Vidéoconférence	
Utilisateurs	10		7		5		4	
Postes	10		5		5		4	
Sous-Groupes	5		3		3		3	
Décompositions fonctionnelles	3		2		2		6	
Assemblages de Sous-Groupes	2		2		2		2	
Rôles pour une décomposition fonctionnelle	20		10		10		6	
Composants pour un rôle	5		3		2		2	
Déploiements d'un Sous-Groupe	3E+34	**1500**	1E+12	**150**	2E+10	**100**	1572864	**216**
Déploiements d'un Groupe	4E+172	**1500**	3E+36	**150**	2E+31	**100**	8E+18	**216**
Déploiements de l'application	$> 10^{307}$	**1500**	3E+255	**150**	1E+156	**100**	4E+75	**216**
	max	**heuris.**	max	**heuris.**	max	**heuris.**	max	**heuris.**

Tableau 17 : Comparaison entre la complexité initiale et la complexité de l'heuristique

Enfin, nous remarquons que pour notre exemple de vidéoconférence, la complexité est passée d'un nombre extrêmement élevé — 4.10^{75} configurations — à un nombre tout a fait implantable — 216 pour *Fl*=1 ou 432 pour *Fl*=2.

3.2.4. Synthèse

La plate-forme supervise l'application de manière à optimiser sa QdS en fonction du contexte d'exécution. Lorsqu'une reconfiguration est nécessaire, elle doit proposer une configuration offrant une meilleure QdS. Or il n'est pas possible de déterminer la configuration la plus adaptée à un contexte donné car ce problème est NP-complet. Nous avons donc proposé un modèle de plate-forme mettant en œuvre une heuristique itérative qui améliore la QdS à chaque itération. La recherche d'une configuration meilleure dans cette heuristique repose alors sur deux critères.

Le premier est imposé par les contraintes temporelles des applications multimédias : c'est l'efficacité de la plate-forme. Pour l'atteindre, nous proposons que la plate-forme utilise des événements décrivant le contexte. Ceux-ci demandent une reconfiguration en indiquant quelle entité de l'application pose problème.

Le second critère vient, lui aussi, des particularités des applications multimédias 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. Il est obtenu grâce à l'étude de la proximité de service. 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 de l'utilisateur.

Cependant ces deux critères se sont avérés insuffisants lorsque l'utilisateur ne peut indiquer ses goûts en raison de la complexité liée à un nombre trop élevé de possibilités ou lorsqu'une entité doit être déployée *ex-nihilo*. Dans ces cas-là, nous proposons trois solutions. La première consiste à prendre en compte les configurations par défaut définies par le concepteur lorsqu'elles sont disponibles. La deuxième repose sur l'évaluation par la plate-forme de toutes les possibilités lorsque cela est réalisable. Enfin, la troisième consiste

en l'implantation de la configuration la moins coûteuse en ressources systèmes ou la configuration la plus souvent utilisée.

A partir de tous ces critères nous avons construit une heuristique dont la complexité 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. Selon toute logique, cette complexité là est maintenant incompressible.

La plate-forme que nous proposons réalise ainsi une adaptation de l'application au contexte. Cette adaptation est entièrement dynamique puisque les politiques d'adaptation sont déterminées en cours d'exécution en fonction de la proximité de service et puisque son exécution est réalisée également en cours d'exécution. De plus, elle concerne à la fois la structure, les fonctionnalités et l'ordonnancement des applications ce qui la rend plus complète que ce qui est souvent proposé — JQoS, Agilos (cf. §1.2.3 p.51). Elle autorise aussi bien des améliorations que des dégradations de qualité. En effet, même si son objectif demeure à tout instant d'améliorer la QdS telle que nous l'avons définie, en pratique, ceci peut passer par une dégradation des caractéristiques intrinsèques du service — taille des images, nombre de couleurs, etc. — de manière à obtenir une amélioration de la qualité globale grâce à un compromis entre les critères intrinsèque et contextuel. Cette possibilité d'évolution dans les deux sens de la qualité constitue donc un progrès par rapport aux systèmes habituellement proposés tels que JQoS où seule la dégradation est réalisée de façon automatique. Enfin, la plate-forme utilise une heuristique pour déterminer la meilleure configuration. Elle propose donc une solution à un problème NP-complet non pas à partir d'une vision uniquement mathématique — comme le sont les solutions issues de l'étude des graphes pour les réseaux — mais en considérant l'utilisateur puisqu'elle se base sur la proximité de service. Même à ce niveau-là, les solutions que nous proposons sont donc centrées sur l'utilisateur.

3.3. Implantation de la plate-forme

Pour être implanté, le modèle de plate-forme que nous avons défini doit être complété par une étude de la gestion des événements. Nous pouvons alors définir comment la plate-forme peut être répartie pour réaliser ses différentes tâches que sont la gestion des événements et la recherche d'une configuration meilleure qui nécessite, en particulier, l'évaluation des configurations. Enfin, nous proposons une structure d'implantation de la plate-forme que nous utilisons dans un prototype permettant de valider notre modèle.

3.3.1. Gestion répartie des événements

Diverses entités de l'application ou de la plate-forme génèrent des événements que nous proposons de classer en différents types. Nous proposons qu'un gestionnaire d'événements décide à tout instant de l'événement que la plate-forme traitera. Nous présentons ici son modèle après avoir justifié son utilisation par l'étude du comportement de la plate-forme.

3.3.1.1. Nécessité de la gestion des événements

La plate-forme manipule différents types d'événements dont le traitement définit l'enchaînement des itérations de recherche d'une configuration meilleure. Le gestionnaire d'événements doit être capable d'ordonner les événements selon leur importance et l'urgence de leur traitement.

a. Caractérisation des événements

En fonction de leur origine et des informations qu'ils véhiculent, les événements peuvent être classés en différents types. Ils peuvent être générés par des Conduits liés à des flux, par des Processeurs Élémentaires encapsulant les composants métier de l'application ou par des composants espions (cf. §3.1.2.1.d p.161). Nous proposons qu'ils soient émis avec une périodicité que nous nommerons T et ceci jusqu'à ce que leur cause disparaisse — saturation du réseau, buffer vide, création d'un Groupe, etc. Il est donc inutile de les acquitter puisque leur cause disparaît dès qu'ils sont traités avec succès.

Ces événements peuvent être classés en deux types. Tout d'abord, les événements notés $\mathbf{E_1}$ qui indiquent que des constituants de l'application ne remplissent plus correctement leur rôle comme lors de la saturation d'un Conduit ou d'un composant ou lorsqu'un locuteur utilise un idiome que l'utilisateur ne comprend pas. Dans ces cas, l'événement avertit la plate-forme que la QdS est en train de se dégrader. Ces événements indiquent donc qu'il faut reconfigurer car la configuration actuelle n'est plus adaptée au contexte.

Le second type d'événement $\mathbf{E_2}$ indique que des constituants de l'application sont sous-utilisés. C'est le cas lorsqu'il existe une grande marge de manœuvre sur le réseau ou sur les composants. Ils préviennent donc la plate-forme que la QdS peut être améliorée. Ces événements indiquent donc qu'il est possible de reconfigurer car la configuration actuelle n'est pas optimale pour le contexte actuel. Ils sont issus des Conduits et des Processeurs Élémentaires.

Quel que soit le type de l'événement de reconfiguration qu'elle a reçu, la plate-forme utilise l'heuristique que nous avons présentée pour trouver une configuration meilleure et l'implanter.

b. Enchaînement des itérations

A la réception d'un événement, la plate-forme recherche puis implante une configuration qu'elle pense être meilleure. Si, après cela, aucun nouvel événement n'est généré, son estimation est exacte et la configuration choisie est la meilleure pour le contexte actuel. La plate-forme peut donc rester en attente. En revanche, si un nouvel événement est généré, son apparition peut être interprétée de différentes façons selon son type.

Si l'événement est de type E_1, la QdS est en train de se dégrader ce qui signifie que l'estimation de la plate-forme est erronée. Cette erreur peut être causée par une variation du contexte survenue entre l'instant de l'évaluation et la reconfiguration. Elle peut également résulter d'imprécisions d'estimation liées aux caractéristiques décrivant le fonctionnement des composants : en effet, ces dernières sont définies avec une marge d'erreur.

Si l'événement est de type E_2, la configuration implantée n'est pas la meilleure pour le contexte actuel. Il y a donc à nouveau une erreur d'estimation soit parce que le contexte a changé soit parce que la plate-forme n'a pas trouvé la meilleure configuration mais juste une meilleure.

Quel que soit le type d'événement, la plate-forme doit à nouveau reconfigurer l'application.

c. Gérer les événements

La question est alors de savoir si la plate-forme doit poursuivre la recherche en cours ou reprendre une nouvelle recherche à partir du nouvel événement ou à partir de l'événement initial. Le comportement que doit avoir la plate-forme diffère selon les cas.

Si le contexte a varié, quel que soit le type d'événement, c'est une nouvelle recherche qui commence car les configurations écartées pour un contexte peuvent être les meilleures pour un autre. De plus, la plasticité impose de considérer le dernier événement reçu de manière à conserver la continuité ergonomique de l'application.

Si des imprécisions d'estimation sont à l'origine de l'échec de la première reconfiguration, rien ne peut y remédier. L'étude des configurations étant basée sur la proximité de service, la plate-forme entame une nouvelle recherche à partir du dernier événement généré. En effet, c'est ainsi qu'elle est le plus susceptible de trouver une configuration correspondant au service le plus proche possible et présentant une estimation plus juste.

Si la configuration trouvée par la plate-forme n'est pas la meilleure, la recherche doit continuer. A priori, il semble inefficace de reprendre une recherche depuis le début. En réalité, chaque nouvel événement décrit un aspect du contexte et la plate-forme doit donc l'utiliser en reconfigurant à partir de l'entité problématique qu'il désigne. Cependant, certains problèmes sont plus aigus que d'autres. C'est pourquoi il est nécessaire de définir un ordre de priorité entre les événements et d'utiliser un gestionnaire qui le mette en œuvre. Ainsi la plate-forme ne s'enfermera pas dans des solutions de type redéploiement ou dans des améliorations peu importantes.

3.3.1.2. Priorités des événements

La plate-forme doit traiter, en priorité, les événements qui traduisent une perturbation dans le service fourni. Ici aussi, nous proposons un ordre de priorité entre les événements induit par l'importance de la perception que l'utilisateur a du service rendu. Nous définissons ensuite comment gérer ces différentes priorités.

a. Distinctions entre événements

Les deux types d'événements définis précédemment (cf. §3.3.1.1.a p.203) doivent donc être qualifiés en fonction de leur répercussion sur le service.

Nous discernerons, tout d'abord, les événements qui modifient un service actuellement fourni à un utilisateur. Ils peuvent le modifier parce que le contexte s'améliore et dans ce cas, ils appartiennent au type $\mathbf{E_2}$. Ils peuvent également le modifier parce que les constituants de l'application ne remplissent plus correctement leur rôle et appartiennent donc au type E_1. Ils peuvent alors être issus des composants espions ce que nous appelons le type $\mathbf{E_{1A}}$ et, dans ce cas, ils provoquent forcément des variations du critère intrinsèque. Ils peuvent également être générés par des Composants de l'application ou des Conduits, ce que nous nommerons le type $\mathbf{E_{1B}}$. Dans ce cas, ils provoquent des variations du critère contextuel ou des deux critères.

Nous discernerons enfin les événements qui créent un service pour un utilisateur. Ils sont issus des composants espions et appartiennent au type E_1, nous les nommerons $\mathbf{E_{1C}}$.

b. Ordre des événements

Nous proposons alors de classer ces quatre types d'événements par ordre de priorité décroissante (cf. Tableau 18 p.206).

La plus forte priorité est donnée aux événements E_{1A}. En effet, si un utilisateur a un service fortement dégradé, la plate-forme doit intervenir vite donc les événements E_1 doivent être traités avant les événements E_2. De plus, une configuration propose un service plus éloigné du service antérieur si elle en diffère par le critère intrinsèque que si elle en diffère par le critère contextuel (cf. §3.1.2.2.a p.163). Or un événement E_{1A} provoque forcément une modification du critère intrinsèque contrairement à un événement E_{1B} qui peut se contenter de modifier le critère contextuel. La plate-forme doit donc traiter les événements E_{1A} avant les E_{1B}. Notons également qu'une modification du critère intrinsèque étend ou restreint les qualités qu'une application peut potentiellement atteindre lors de variations du contexte ce qui n'est pas le cas du critère contextuel. La priorité de rang deux est alors octroyée aux événements E_{1B} qui traitent une dégradation du service contrairement aux E_{1C}.

Ces événements, E_{1C}, ont la priorité de rang trois car un utilisateur peut patienter avant de disposer de l'application, le temps que la plate-forme améliore le service des autres utilisateurs. L'arrivée de ce nouvel utilisateur peut améliorer la situation puisque l'application dispose d'un nouveau poste ou la dégrader puisque la charge du réseau sera localement augmentée ainsi que celle de certains postes. Cependant l'attente ne peut durer trop longtemps, c'est pourquoi nous proposons d'introduire une temporisation : lorsqu'un événement E_{1C} est généré, la plate-forme attend un délai fixé par le concepteur au terme duquel sera généré un événement noté $\mathbf{E_T}$ de priorité exceptionnelle — plus haute que celle des événements E_{1A}. L'événement E_T est généré avec la même périodicité T que les autres

événements. Il contient, à chaque fois, le temps écoulé depuis le début de la temporisation. Lorsque l'événement E_{1C} initial disparaît, la temporisation et l'événement E_T sont également supprimés.

Enfin la priorité la plus basse est accordée aux événements E_2 qui indiquent qu'une amélioration peut avoir lieu.

Reste à définir l'ordre de traitement des événements appartenant à une même catégorie. Les événements E_T sont classés en fonction du temps écoulé depuis la création de la temporisation ce qui revient à fournir le service aux utilisateurs en fonction de l'ordre d'arrivée de leur demande. De plus, un événement de type E_T ne peut pas interrompre le traitement d'un autre événement E_T puisque les services ne peuvent pas être partiellement fournis.

Les événements E_{1A} sont classés en fonction du niveau structurel de leur répercussion sur le service de manière à respecter la continuité ergonomique de celui-ci donnée par l'ordre établi au paragraphe 3.1.2.2.c : Sous-Groupe, rôle critique, composant critique, rôle non critique et composant non critique.

Les événements E_{1B} sont classés en fonction du niveau de saturation donné en pourcentage. Ainsi ces événements permettent à la plate-forme de réagir avant une saturation totale du système puisqu'ils sont réitérés avec une priorité croissante.

Les événements E_{1C} sont classés en fonction de leur ordre d'arrivée et les événements E_2 en fonction de l'amplitude de la marge de manœuvre observée en pourcentage obtenue grâce à des mesures actives.

L'ordre de priorité de traitement des différents événements par la plate-forme est résumé dans le Tableau 18.

Type	Signification	Nom	Influence sur le service	Origine	Priorité	Interclassement
E_T	Attente liée à E_{1C}	E_T	Création	Temporisation	Exception	par l'ordre d'arrivée
E_1	Dégradation en cours de la qualité de service	E_{1A}	Modification	Composant espion	1	C_{SG}> C_{RC}>C_{CC}>C_{RnC} >C_{CnC}
		E_{1B}	Modification	Processeur Élémentaire	2	par le pourcentage de saturation
				Conduit		
		E_{1C}	Création	Composant espion	3	par l'ordre d'arrivée
E_2	Amélioration possible de la qualité de service	E_2	Modification	Processeur Élémentaire	4	par le pourcentage de marge de manœuvre
				Conduit		

Tableau 18 : Définitions et priorités des différents événements

c. Gestion des priorités entre événements

Nous proposons qu'un gestionnaire réparti d'événements mette en œuvre les priorités précédemment présentées. Il reçoit les événements et compare les priorités pour indiquer à la plate-forme l'événement qu'elle doit traiter et, le cas échéant, lui demander d'interrompre une recherche. Il lui fournit également les informations nécessaires au traitement de l'événement et héberge les temporisations liées aux événements de type E_T.

Il gère le cas particulier où la plate-forme n'a pas trouvé une configuration meilleure après l'arrivée d'un événement E_i. Cet événement verra alors sa priorité déclassée et mise au niveau le plus bas. Lorsque cet événement sera à nouveau généré, il ne sera traité qu'à condition qu'il n'y ait pas d'autre événement en attente. En effet, le traitement des autres événements génère un nouveau contexte qui peut se révéler plus favorable à la résolution du problème à l'origine de cet événement. La Figure 83 résume alors l'algorithme implanté par le gestionnaire d'événements.

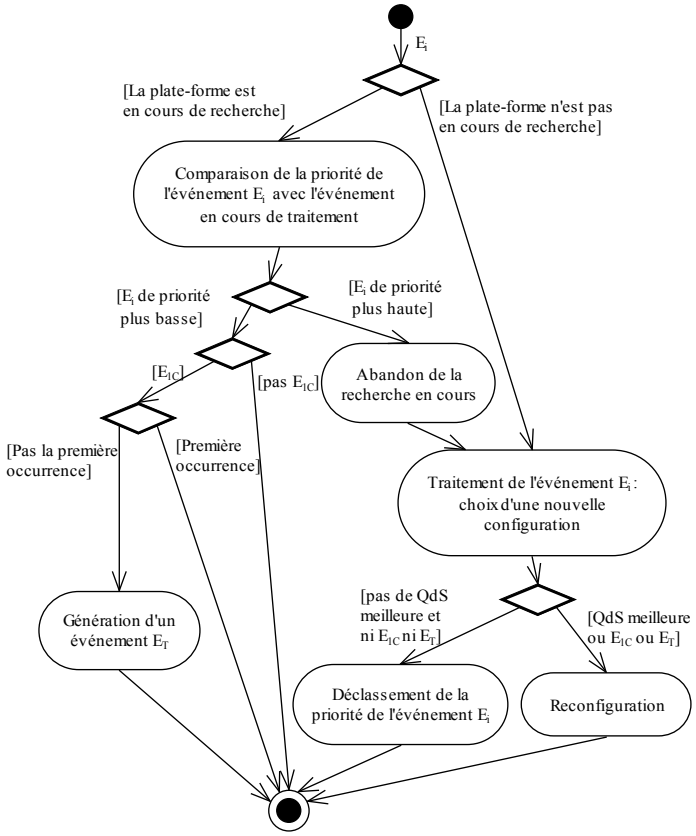


Figure 83 : Gestion des priorités entre événements

La plate-forme va ainsi reconfigurer l'application jusqu'à ce que la QdS soit stable c'est-à-dire jusqu'à ce qu'il n'y ait plus d'événements E_1 et jusqu'à ce que l'application n'ait plus de marge de manœuvre ni sur le critère intrinsèque ni sur le critère contextuel c'est-à-dire jusqu'à ce qu'il n'y ait plus d'événements E_2. Or la QdS relève d'un compromis entre le contextuel et le non contextuel. Il est donc fort probable qu'il sera rare d'avoir une configuration qui optimisera les deux critères à la fois, d'autant plus que nous savons que le

problème initial est NP-complet. Le danger est alors d'avoir une plate-forme reconfigurant sans cesse en vain. L'utilisation des événements pour orienter la recherche permet alors de s'assurer que la plate-forme n'étudie pas toujours les mêmes solutions. Nous évitons ainsi qu'elle cherche à améliorer la QdS uniquement d'une seule façon ou en un seul endroit, qui est déjà dans son état optimal pour le contexte donné, alors qu'un autre point pourrait être amélioré.

3.3.1.3. Gestionnaire d'événements

Nous avons proposé que la plate-forme soit répartie. Il nous reste à déterminer à quel niveau elle l'est. Cette répartition doit lui permettre de traiter en parallèle plusieurs événements et ainsi d'être plus rapide et plus efficace. La seule condition est que leur traitement soit non conflictuel.

a. Conflits potentiels dans la gestion des événements

Le traitement simultané de plusieurs événements est limité, d'une part, par notre volonté d'assurer la plasticité de l'application et, d'autre part, par l'impossibilité d'implanter un même composant de deux manières différentes en dehors de toute duplication.

En effet, nous avons proposé une heuristique de recherche de la future configuration qui respecte la continuité ergonomique de l'application (cf. §3.1.2.1 p.159). Pour les événements issus des Processeurs Élémentaires et des Conduits — E_{1A} et E_2 — l'étude des différentes configurations possibles débute par une évaluation des différentes possibilités de déploiement et d'ordonnancement d'un Sous-Groupe puis est élargie à l'étude de différents assemblages possibles de Sous-Groupes pour un Groupe donné. Si la plate-forme traite un événement, elle propose de modifier la QdS fournie par le Sous-Groupe de manière à résoudre le problème en s'éloignant le moins possible du service fourni. Pour deux événements, elle ne peut plus assurer cette continuité ergonomique car chacun aura des répercussions sur la QdS de l'autre. C'est pourquoi nous proposons que la plate-forme n'étudie qu'une modification possible au sein d'un Sous-Groupe à la fois.

De la même façon, les événements issus d'un composant espion notés E_{1A} peuvent, soit imposer de déployer un ou des rôles dans un Sous-Groupe, soit imposer de déployer un ou plusieurs Sous-Groupes. Nous proposons que la plate-forme ne traite qu'un événement de ce type par Sous-Groupe lorsqu'il impose un ou des rôles.

En revanche, au sein d'un Groupe, des événements liés à des Sous-Groupes différents peuvent être traités en parallèle tant qu'ils ne concernent pas des composants ou des flux communs et que la recherche n'en arrive pas à proposer de modifier les différents assemblages de Sous-Groupes. De la même façon, plusieurs Groupes peuvent être reconfigurés en même temps s'ils ne partagent pas de composants.

D'autre part, les événements de type E_{1c} et E_T créent un service pour un utilisateur. Or celui-ci peut avoir besoin d'utiliser des composants de l'application déjà implantés qui seront donc partagés avec d'autres utilisateurs. Nous avons proposé que ce déploiement ne modifie pas les services déjà rendus. Les événements de type E_{1C} et E_T pourront donc être traités en parallèle avec tout type d'événement car ils ne modifient pas les autres services. En revanche, les événements de ce type ne pourront être traités simultanément que si les Groupes ne partagent pas de composants.

En fonction de ces critères, le traitement parallèle des événements par la plate-forme est réalisable à deux niveaux, Groupe et application. Au sein d'un Groupe, plusieurs événements peuvent être traités simultanément si les Sous-Groupes concernés n'ont pas de composant en commun et que la recherche se limite à modifier le Sous-Groupe puisqu'un seul événement par Sous-Groupe peut être traité. Au sein de la plate-forme, plusieurs événements peuvent être traités en parallèle si les Groupes concernés n'ont pas de composants en commun et qu'un seul événement par Groupe est considéré. La Figure 84 illustre ces possibilités de traitement en parallèle des événements lors de la recherche de la future configuration que ce soit un composant de l'application (cf. Figure 74 p.185) ou un composant espion qui soit à l'origine de la reconfiguration (cf. Figure 82 p.197).

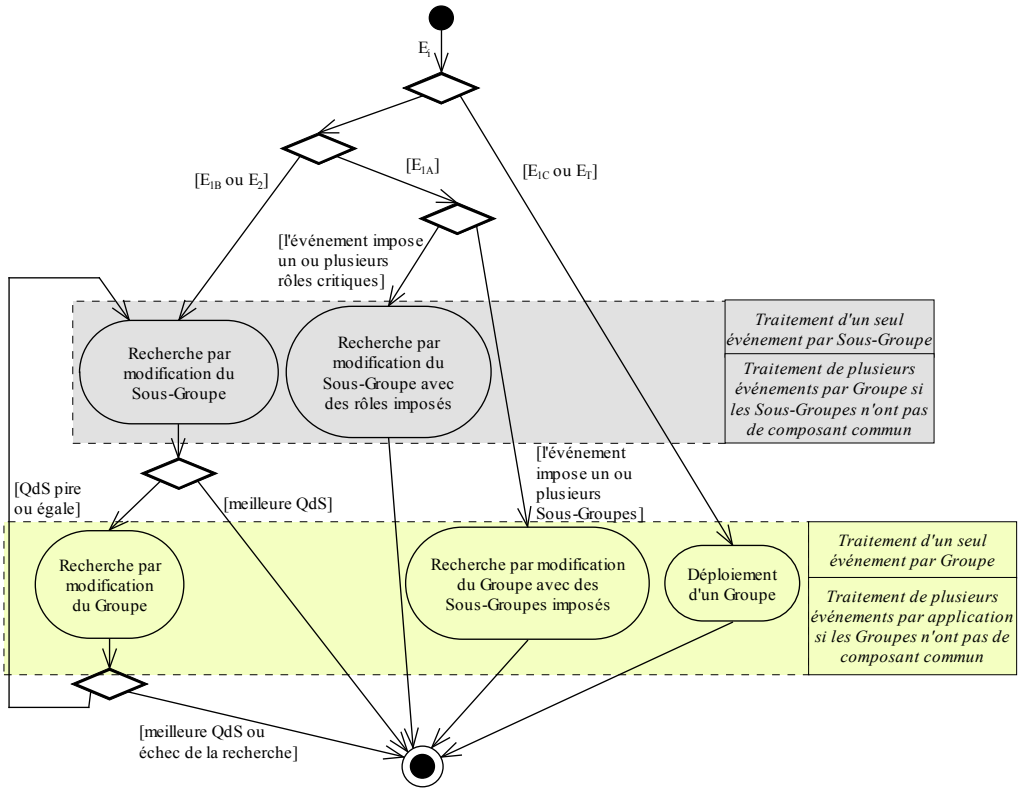


Figure 84 : Choix d'une nouvelle configuration et traitements parallèles des événements

b. Définition et répartition du gestionnaire d'événements

Le Groupe est le niveau le plus bas où des événements peuvent être traités en parallèle. Or ce parallélisme permet à la plate-forme de réagir plus rapidement. C'est pourquoi nous proposons de répartir la gestion des événements entre tous les Groupes. Ainsi chaque Groupe sera associé à un gestionnaire d'événements. Nous proposons de le placer sur le poste de l'utilisateur puisque c'est le seul poste que le Groupe utilise obligatoirement car il contient les composants observables de chaque Sous-Groupe. Ce gestionnaire sera déployé préalablement à l'application puisqu'il doit pouvoir prendre en compte la demande de création d'un Groupe.

La Figure 85 page 211 décrit le comportement d'un gestionnaire d'événements de Groupe. Celui-ci respecte la gestion des événements décrite par la Figure 83 page 207 et utilise le parallélisme possible mis en exergue par la Figure 84 page 209. Cette gestion des conflits nous oblige donc à imposer qu'une recherche soit abandonnée lorsque l'événement qui l'a causée disparaît. Nous remarquons cependant que lorsque le niveau de saturation d'un composant ou d'un flux augmente, l'événement représentatif sera généré et le niveau de saturation augmente. Ainsi la plate-forme a toutes les chances de réagir avant le collapsus. Nous notons que tout le fonctionnement de la plate-forme est interruptible par un événement de priorité supérieure excepté la reconfiguration et le déclassement d'un événement.

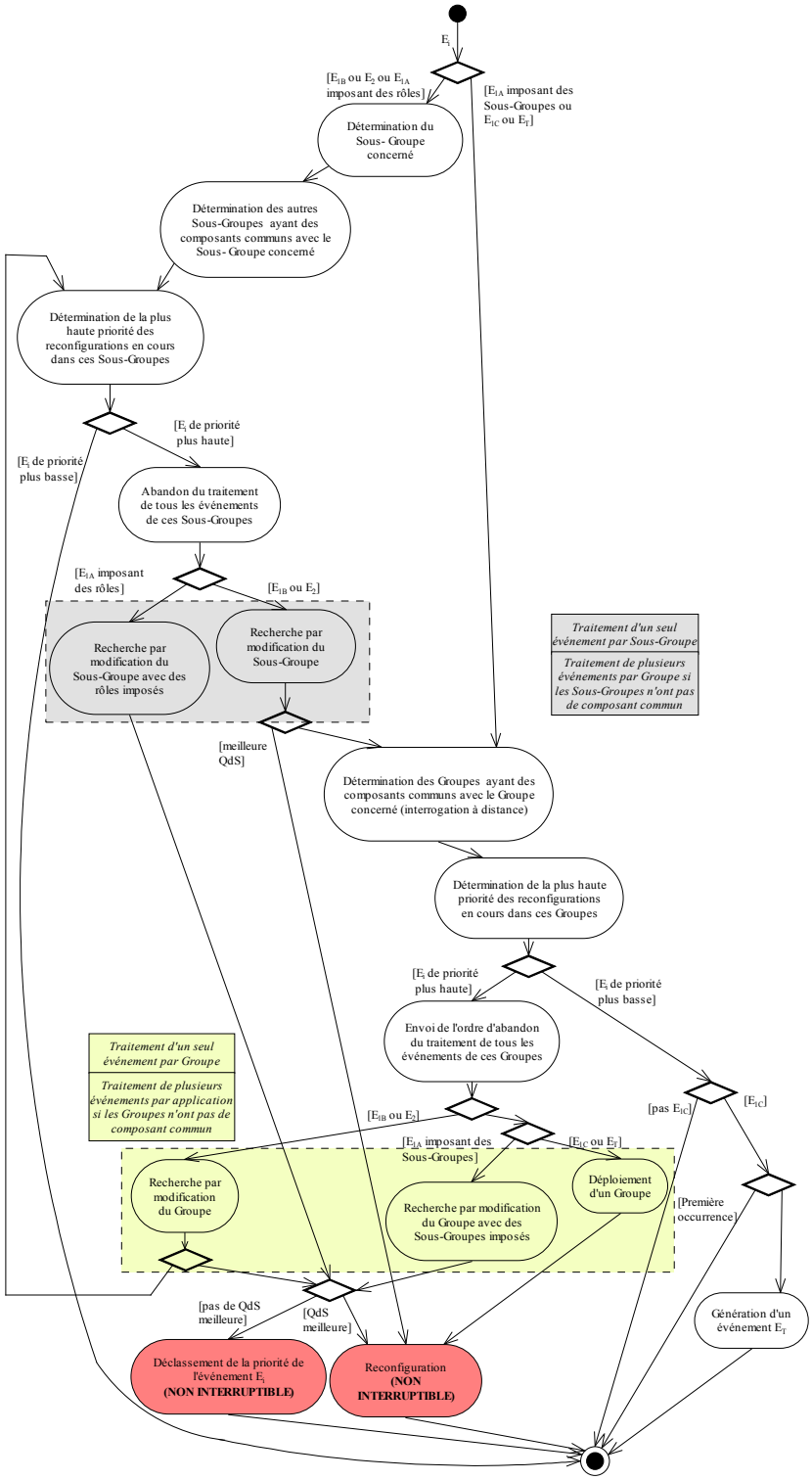


Figure 85 : Gestionnaire d'événements d'un Groupe

Le gestionnaire d'événements réalise trois tâches principales représentées sur la Figure 86 ainsi que leurs différents enchaînements :

- le choix de l'événement à traiter ;
- le choix de la configuration à évaluer ;
- le choix de reconfigurer ou pas en fonction du résultat de l'évaluation.

Nous proposons une structure de la plate-forme telle que ce gestionnaire ne réalise ni l'évaluation des configurations potentielles ni la mise en œuvre des reconfigurations.

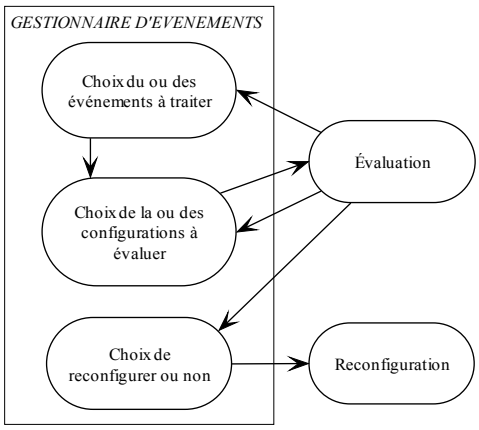


Figure 86 : Trois tâches principales du gestionnaire d'événements

3.3.2. Structure de la plate-forme

Comme nous venons de le voir, la plate-forme propose une gestion répartie des événements. Toutefois leur traitement peut être centralisé ou réparti et il consiste principalement à évaluer des configurations potentielles. Nous avons tout d'abord à choisir entre une évaluation répartie ou centralisée. Puis, comme les évaluations permettent de choisir la configuration à implanter, nous étudions la mise en œuvre des reconfigurations. Nous pouvons alors proposer un modèle structurel pour notre plate-forme.

3.3.2.1. Gestion répartie de l'évaluation d'une configuration

a. Choix d'une évaluation parallèle et répartie

Plus la plate-forme évalue rapidement les configurations potentielles, plus elle trouve rapidement la configuration à implanter. En outre, sa rapidité de réaction à un événement assure à l'utilisateur une dégradation moindre de sa QdS. Le temps d'exécution de l'algorithme doit donc être réduit au maximum. Pour cela, nous proposons d'utiliser une

programmation parallèle et répartie que notre modélisation à base de flots de données autorise (cf. §1.4.1 p.71).

Nous proposons cependant que les informations nécessaires à cette évaluation — comme les documents décrivant les caractéristiques de fonctionnement des composants de l'application — soient initialement centralisées sur un poste privilégié défini par le concepteur de l'application. Ce poste central pourra également héberger un dépôt de composants dans lequel la plate-forme pourra puiser les composants logiciels à déployer.

Enfin, l'évaluation nécessite également de récolter, au préalable, les vœux de l'utilisateur ce qui ne pourra se faire que sur le poste de celui-ci.

b. Mise en œuvre de l'évaluation

De manière à minimiser les échanges de données entre postes, nous proposons que les informations décrivant le contexte soit utilisées là où elles sont récoltées. Ainsi un composant potentiel de l'application est évalué sur le poste où il sera implanté. C'est pourquoi nous proposons que chaque poste soit doté d'un gestionnaire d'évaluation.

Lorsque le gestionnaire d'événements du poste d'un utilisateur décide qu'un événement doit être traité, il en informe le gestionnaire d'évaluation qui va mesurer la QdS de la configuration en cours d'exécution et celle des configurations potentielles (cf. Figure 87 p.214). Pour cela, il prévient le gestionnaire d'évaluation du (des) poste(s) du (des) premier(s) successeur(s) de la source dans le graphe de la configuration à évaluer. Il lui (leur) indique la configuration à étudier. Le gestionnaire du composant définit le vecteur décrivant la QdS que le composant fournirait et l'envoie sur le poste hébergeant le composant suivant. Le gestionnaire d'évaluation joint à ce vecteur les étiquettes décrivant la configuration en cours d'évaluation de manière à ce que le poste récepteur sache quelle configuration et quels composants évaluer. Ces étiquettes sont définies relativement à la configuration en cours d'exécution. Chaque poste dispose d'une description de cette dernière. En effet, à la suite de chaque reconfiguration, le poste à l'origine de celle-ci indique les modifications effectuées sur l'application à tous les autres postes et bloque le traitement des événements en cours tant que la mise à jour n'a pas été faite.

Chaque composant de l'application est ainsi évalué. L'évaluation du composant observable est réalisée sur le poste de l'utilisateur où se trouve également le gestionnaire d'événements à l'origine de cette évaluation. La plate-forme locale peut alors définir la note de cette configuration et décider de l'implanter ou non en la comparant à la note de la configuration en cours d'exécution.

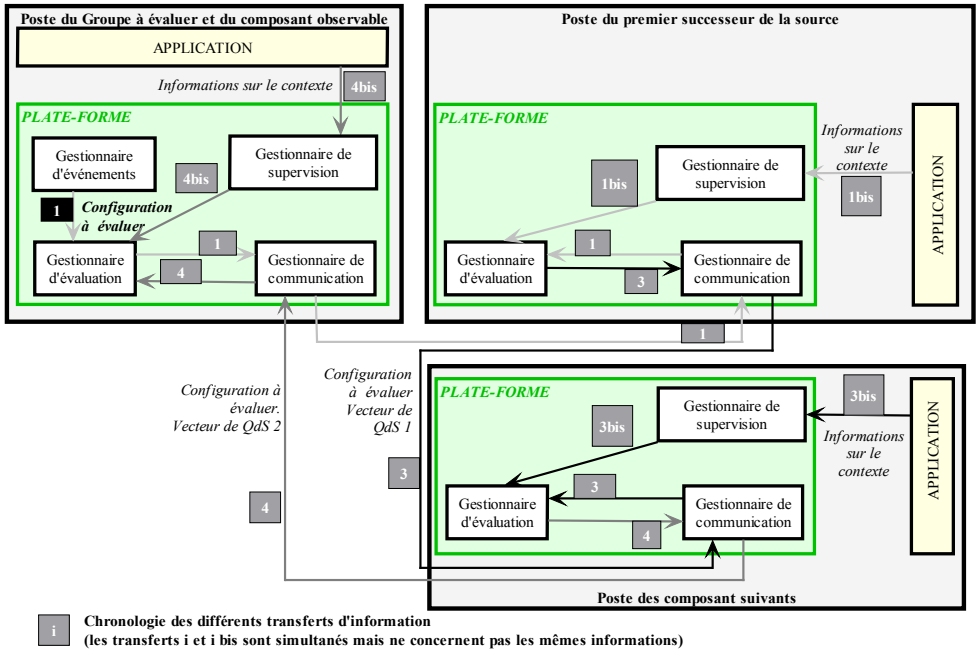


Figure 87 : Représentation de l'évaluation répartie dans un cas simple

3.3.2.2. Modèle structurel de la plate-forme

a. Les cinq gestionnaires

La plate-forme réalise à la fois une gestion répartie des événements de reconfiguration et une gestion répartie de l'évaluation des configurations potentielles. Nous avons proposé d'utiliser pour cela deux gestionnaires :

- un gestionnaire d'événements associé à chaque Groupe et donc à chaque utilisateur ;
- un gestionnaire d'évaluation associé à chaque poste de l'application réalisant l'évaluation locale des composants et des flux.

Nous proposons d'y ajouter :

- un gestionnaire de communication associé à chaque poste qui assure la communication entre tous les postes utilisés par l'application et donc par la plate-forme ;
- un gestionnaire d'utilisateur associé à chaque Groupe permettant de recueillir les vœux de celui-ci ;
- un gestionnaire de supervision associé à chaque poste de l'application : il héberge les composants espions du poste et gère la reconfiguration des composants locaux de l'application.

Or, dès lors que l'application utilise un poste, il est possible qu'un utilisateur souhaite s'en servir ce qui impose la présence d'un gestionnaire d'événements susceptible de recevoir une demande de création de Groupe et d'un gestionnaire d'utilisateur. Ces deux entités demeurent cependant liées au Groupe et non au poste. Finalement, la plate-forme sera donc répartie sur tous les postes de l'application et comprendra sur chacun les cinq gestionnaires cités (cf. Figure 88).

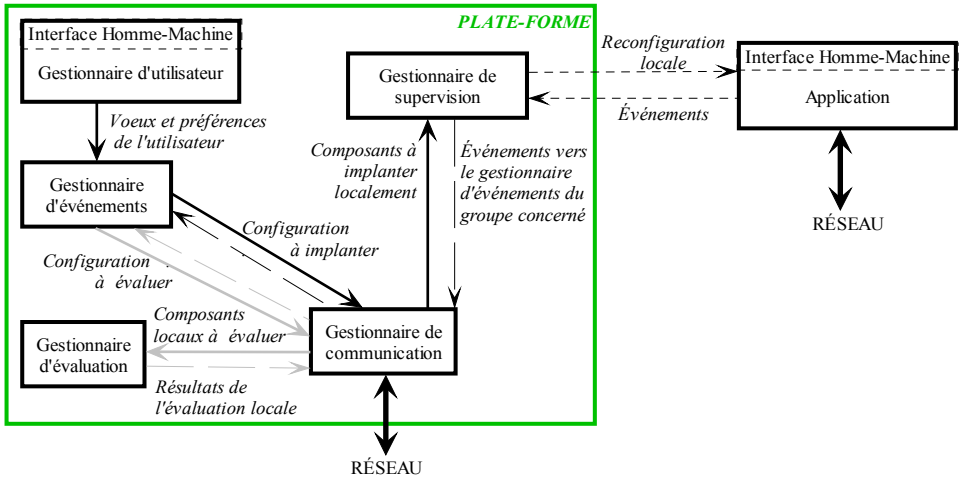


Figure 88 : Modèle structurel de la plate-forme

b. Fonctionnement

Les Conduits et les Processeurs Élémentaires de la partie locale de l'application émettent des événements vers le gestionnaire de supervision du poste. Celui-ci héberge les composants espions qui émettent également des événements. Tous ces événements sont transmis via le gestionnaire de communication aux gestionnaires d'événements des Groupes concernés.

En fonction des vœux et des préférences de l'utilisateur, le gestionnaire d'événements d'un Groupe donné détermine l'événement à traiter puis la configuration à évaluer en plus de la configuration en cours d'exécution. Il en informe les gestionnaires d'évaluation des postes des premiers composants des configurations à évaluer qui déterminent les caractéristiques des services avant de transmettre l'ordre d'évaluation aux postes suivants. En bout de graphe, le gestionnaire d'événements initiateur de l'évaluation reçoit les caractéristiques de service de la configuration à évaluer et détermine sa note de QdS.

En fonction de la note obtenue, le gestionnaire d'événements décide d'une nouvelle évaluation, du traitement d'un nouvel événement ou d'une reconfiguration. Dans ce dernier cas, il transmet la nouvelle configuration aux gestionnaires de supervision de tous les postes concernés soit par la suppression soit par l'ajout de composants.

3.3.2.3. Mise en œuvre des reconfigurations

La plate-forme partage avec l'application les ressources informatiques que ce soit le réseau ou les unités centrales. Or les reconfigurations ont un coût en terme d'utilisation de

ces ressources si bien qu'elles réduisent les ressources disponibles pour l'application. Comme ces ressources ne sont pas illimitées, nous proposons une mise en œuvre des reconfigurations qui perturbe le moins possible l'application de manière à ce que les améliorations de service obtenues par une reconfiguration ne soient pas masquées par les perturbations engendrées par cette même reconfiguration. Nous présentons ici les points clés de la mise en œuvre de la plate-forme.

a. Espacement des reconfigurations

L'espacement des reconfigurations doit être choisi en fonction de deux problématiques. La première est liée aux surcharges du réseau et du matériel que peuvent entraîner des reconfigurations incessantes de l'application. En effet, le risque est alors de perturber l'application au lieu de l'améliorer. La seconde est la nécessité de proposer une évolution graduelle de l'application de manière à conserver la continuité ergonomique de celle-ci. La plate-forme doit donc trouver un juste milieu entre reconfigurer sans cesse et perturber le service et reconfigurer trop peu et provoquer alors des sauts de QdS qui gêneront l'utilisateur.

Nous pensons que l'algorithme que nous proposons répond à cette problématique puisque à chaque niveau de proximité de service — en partant du plus proche pour aller vers le plus éloigné — il étudie un ensemble de configurations avant de reconfigurer. Il évite ainsi de reconfigurer dès qu'il trouve une configuration meilleure et attend la fin de l'étude de l'ensemble considéré. La mise en œuvre de la plate-forme n'a donc pas à se préoccuper de ce problème comme nous le vérifierons avec le prototype.

b. Seuillage

Le coût d'une reconfiguration en terme de charge du réseau et des machines est à comparer avec l'amélioration de QdS obtenue après la reconfiguration. Dans un souci d'efficacité, nous proposons de ne mettre en œuvre une nouvelle configuration qu'à condition qu'elle apporte une amélioration notable de la QdS de façon à éviter de perturber l'application pour un gain de QdS insignifiant aux yeux de l'utilisateur. Cette amélioration minimale sera définie en pourcentage relatif et constitue un seuil noté S que le prototype permettra de fixer.

A la réception d'un événement de reconfiguration, la plate-forme recherche des configurations meilleures que celle en cours d'exécution dont la QdS est notée QdS_a. A la fin de l'étude d'un ensemble, elle implante la meilleure configuration trouvée de qualité de service QdS_r uniquement si $\frac{QdS_r - QdS_a}{QdS_a} \geq S\%$. Dans le cas contraire, la plate-forme poursuit sa recherche par l'étude de l'ensemble suivant.

c. Stabilité du système

Nous avons proposé un algorithme à base d'itérations où les événements permettent de juger de la réussite d'une reconfiguration. En fonction de la répercussion d'une reconfiguration, la plate-forme ajuste son comportement. Ce système constitue donc un système bouclé tel qu'il est défini en automatique. Il est donc nécessaire de vérifier que ce système est stable et en particulier qu'il n'oscille pas malgré les imprécisions dans l'évaluation et la dynamicité de la définition des politiques d'adaptation.

Erreurs d'évaluation

Une erreur d'évaluation du contexte peut être provoquée soit par une variation du contexte survenue entre sa perception et son évaluation soit par des imprécisions dans la caractérisation du comportement des composants. Cette erreur peut empêcher une reconfiguration d'améliorer la QdS voire même la lui faire dégrader. Dans ce cas, comme nous l'avons vu, la plate-forme reprendra sa recherche à partir de cette nouvelle configuration dans le but d'améliorer la QdS. Si elle y parvient, une oscillation de qualité sera ressentie par l'utilisateur en raison de l'erreur d'évaluation initiale.

Or les erreurs d'estimation dues aux variations du contexte sont inévitables. Elles sont liées au temps de réponse de la plate-forme comparable au phénomène appelé "retard pur" en automatique. Il est impératif que ce retard soit négligeable par rapport au comportement global du contexte de manière à ce que la dégradation soit la plus imperceptible possible pour l'utilisateur. Il ne s'agit donc pas véritablement d'oscillation, ce que le prototype nous permet de vérifier.

Les erreurs d'estimation dues aux imprécisions dans la définition du comportement des composants sont accidentelles et ne peuvent pas être prises en compte. Les concepteurs devront veiller à ce que la description des composants qu'ils fournissent soit aussi précise et exacte que possible.

Dynamicité de la définition des politiques d'adaptation

La plate-forme définit la politique d'adaptation qu'elle met en œuvre en fonction de l'événement et du contexte. Cette dynamicité de la définition des politiques d'adaptation peut, elle aussi, amener à des comportements oscillatoires. Or nous avons proposé que la recherche d'une configuration meilleure modifie la configuration en cours d'exécution à partir du composant problématique. Il est donc peu probable que l'heuristique propose la configuration de départ — configuration ayant généré l'événement — d'autant plus que le comportement de la plate-forme n'est pas symétrique : la recherche ne part pas de la même configuration pour améliorer la QdS la première fois ou après une dégradation due à cette première reconfiguration. De plus, l'utilisation d'un seuil en deçà duquel une configuration meilleure n'est pas implantée évite que deux configurations trop proches par le service ne soient implantées alternativement. La plate-forme ne peut donc pas présenter des oscillations liées à la dynamicité de l'adaptation.

La Figure 89 page 218 illustre le comportement théorique de la plate-forme en fonction de l'évolution du contexte. Elle décrit l'évolution de la QdS d'une application en comparaison avec la QdS optimale à tout instant ainsi que les événements provoquant les reconfigurations. Le traitement de certains événements provoquent une amélioration de la qualité alors que d'autres sont suivis d'une dégradation liée à l'évolution du contexte. Cette dégradation est détectée immédiatement par des événements de type E_1 permettant ainsi à la plate-forme de réagir au plus vite et de reconfigurer l'application.

L'objectif du prototype, que nous allons présenter maintenant, est de confirmer ces prévisions.

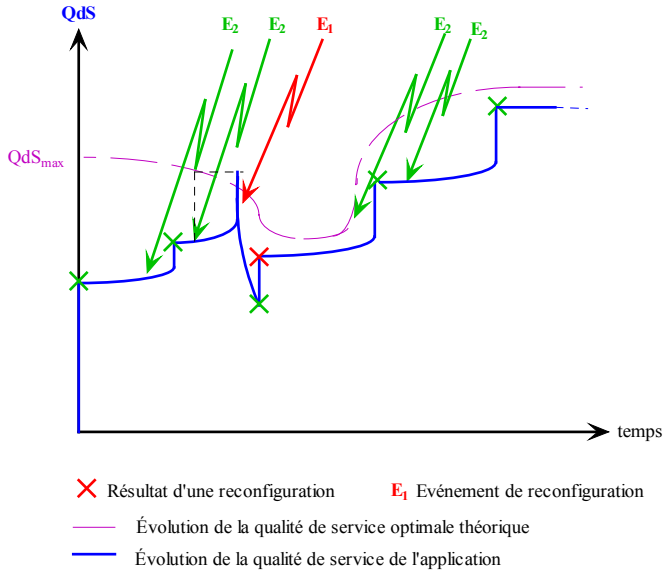


Figure 89 : Exemple d'évolution de la qualité de service

3.3.3. Prototype

Nous avons validé nos propositions à l'aide d'un prototype que nous allons tout d'abord définir. Puis nous expliquons comment nous l'avons réalisé et quels sont les résultats que les expérimentations nous ont permis d'obtenir.

3.3.3.1. Définition

Un prototype est un « Modèle ou mise en œuvre préliminaire permettant l'évaluation de la conception d'un système, de sa réalisation et de son potentiel d'exploitation, ou encore une meilleure identification et compréhension des besoins. » [OFF 96]. Dans notre cas, il s'agit de valider la conception de la plate-forme d'exécution que nous avons proposée. Nous allons définir plus précisément les objectifs de notre prototype puis nous présenterons les résultats attendus de manière à justifier la méthode et les outils employés pour sa réalisation.

a. Objectifs du prototype

Le prototype a pour but d'évaluer le modèle de plate-forme de gestion de la QdS que nous avons proposé pour les applications multimédias distribuées. Il nous permet de valider le comportement global de la plate-forme c'est-à-dire le fait qu'elle améliore la QdS instantanée tout en respectant la plasticité nécessaire.

Conformément à notre modèle, les actions réalisées avant l'exécution ne seront pas prises en compte : définition des Groupes, des Sous-Groupes, des familles de configurations, des caractéristiques participant aux critères intrinsèque et contextuel.

b. Méthode

Nous souhaitons disposer le plus tôt possible et à tout instant d'un prototype validant une partie de plus en plus complète de nos propositions. Pour cette raison, nous ne suivons pas un cycle de vie linéaire – en cascade, en chute d'eau, en V, etc. — mais un modèle non linéaire de développement et plus précisément un cycle de vie itératif.

Nous utilisons une méthode de développement adaptée à notre prototype de plate-forme qui s'inspire du modèle incrémental car celui-ci permet de disposer à tout instant d'un incrément livrable.

Chaque fonctionnalité principale de la plate-forme va constituer un incrément :

- l'évaluation répartie réalisée par le gestionnaire d'évaluation ;
- la recherche d'une meilleure configuration grâce à l'heuristique réalisée par le gestionnaire d'événements ;
- la reconfiguration réalisée par le gestionnaire de supervision ;
- le choix de l'événement à traiter réalisé par le gestionnaire d'événements ;
- l'exploitation des vœux de l'utilisateur réalisée par le gestionnaire d'utilisateur.

Selon sa complexité, chaque incrément est développé selon un cycle de vie différent, incrémental ou non. Puis, son intégration est réalisée grâce à la programmation du gestionnaire de communication. Ainsi, lorsque tous les incréments sont implantés, le prototype permet de valider le modèle de plate-forme.

Nous organisons la programmation en trois grandes étapes de manière à valider tout d'abord une itération de l'heuristique puis, la convergence de l'heuristique et enfin, le comportement global de la plate-forme. La Figure 90 page 220 décrit la méthode de programmation utilisée en précisant :

- les tests unitaires qui valident chaque module du logiciel — le test unitaire j de l'étape i est noté E_iU_j ;
- les tests d'intégration qui vérifient que l'intégration des modules n'a pas altéré leur comportement après leur composition progressive — le test d'intégration j de l'étape i est noté E_iI_j ;
- les tests du système qui constituent les tests finals — le test du système j de l'étape i est noté E_iS_j.

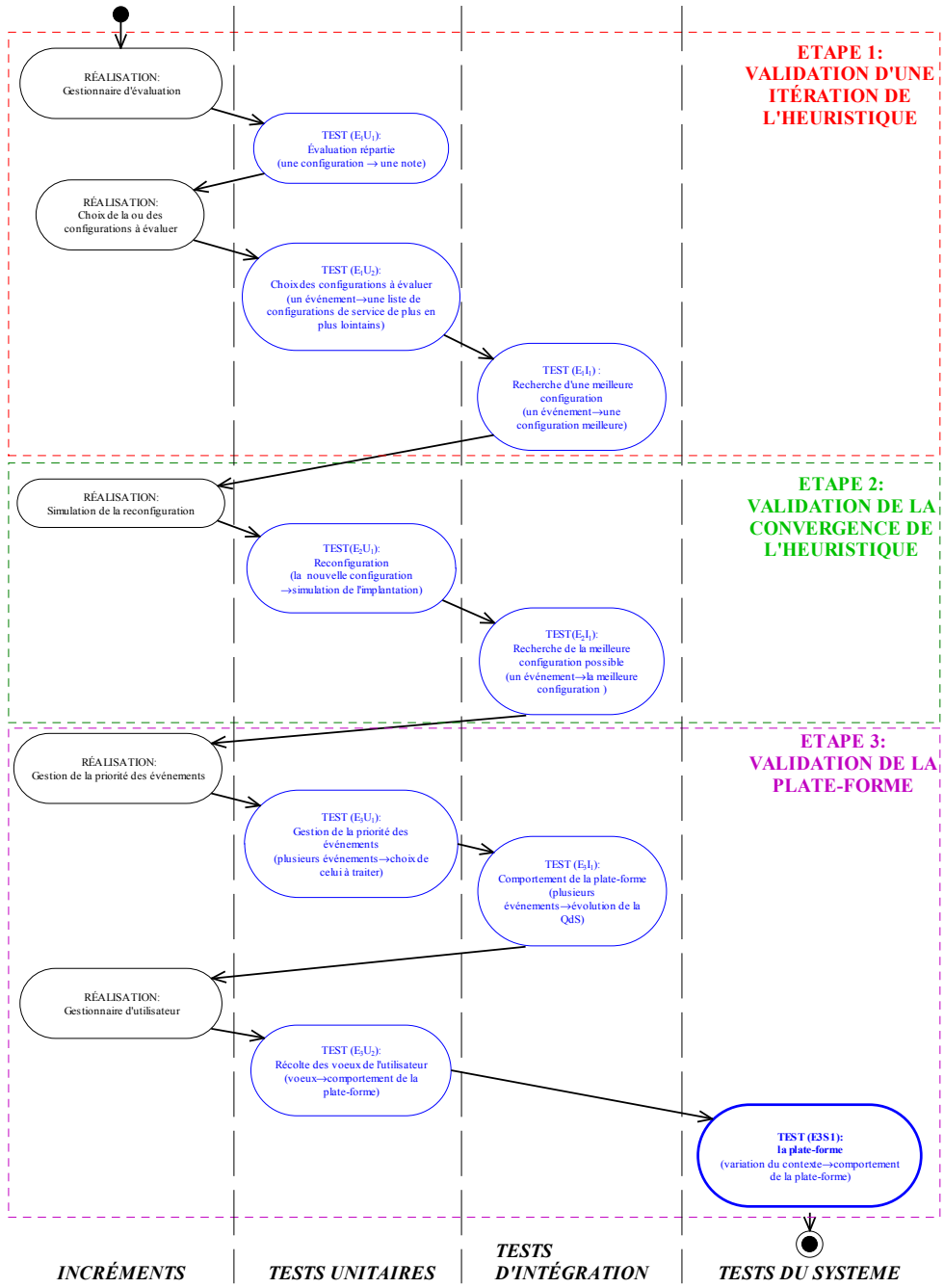


Figure 90 : Programmation du prototype

c. Résultats attendus

Le prototype valide le modèle de plate-forme. Pour cela, il doit confirmer qu'une itération de l'heuristique permet de trouver une configuration ayant une meilleure QdS. Puis il doit vérifier que l'heuristique converge en un nombre d'itérations relativement faible

lorsque le contexte est stable. Il doit également permettre d'identifier les solutions de reconfiguration proposées par la plate-forme : délocalisation, changement de composant, etc. Enfin, il est nécessaire de valider le comportement global de la plate-forme et surtout sa plasticité principalement caractérisée par l'absence d'oscillation et l'obtention d'une évolution graduelle du service.

d. Outils utilisés

Le prototype doit nous permettre de démontrer que l'heuristique que nous proposons permet de réduire la complexité d'un problème NP-complet tout en maintenant une QdS optimale dans un contexte où la plate-forme est vue comme un outil de supervision d'application et l'évaluation est basée sur les flux de données. Pour valider le modèle de plate-forme, nous avons donc réalisé un prototype simulant le comportement de celle-ci. Pour cela, nous avons choisi un outil de simulation axé supervision et flux de données : le logiciel LabVIEW version 6.1 de National Instruments.

LabVIEW est un logiciel de développement d'applications utilisant le langage graphique G pour réaliser des applications de traitement scientifique, d'acquisition, de test et de mesure électronique et enfin de supervision de processus industriels. Il propose des bibliothèques de fonctions et des outils de développement spécialement conçus pour les applications de contrôle d'instruments et d'acquisition de données. Un programme LabVIEW est appelé *Virtual Instrument — V.I.* pour instrument virtuel. Il est composé d'une Face Avant — *Panel* — réalisant éventuellement l'interface interactive avec l'utilisateur et d'un Diagramme — *Diagram* — constituant le code source en langage graphique orienté flux de données. Les programmes ont une structure hiérarchique et modulaire constituée de sous-*V.I.* correspondant à des sous-programmes.

Plusieurs raisons justifient notre choix. Tout d'abord, LabVIEW est un outil permettant de réaliser des applications de supervision or la plate-forme supervise l'application. Puis, la programmation proposée se fait selon le modèle des flux de données et nous avons vu que les contraintes temporelles du multimédia imposent l'utilisation de ce type de programmation. Ensuite, LabVIEW permet un cycle de prototypage rapide qui nous est particulièrement utile puisque nous avons choisi un cycle de vie incrémental pour notre prototype. Enfin, la programmation par *V.I.* constitue une programmation par composants logiciels très facilement composables. Ainsi, la plate-forme et l'application sont programmées selon la même méthode.

3.3.3.2. Réalisation

Pour programmer le prototype nous avons effectué des choix concernant la simulation de l'application et de la plate-forme que nous présentons en premier lieu. Nous décrivons ensuite la structure puis le fonctionnement de notre prototype.

a. Simulation

L'objectif de ce prototype n'est de réaliser ni plate-forme ni des applications multimédias mais d'étudier la QdS. C'est pourquoi l'application et le contexte sont simulés.

Simulation de l'application

L'application est construite à l'aide de composants qui ne réalisent aucune fonction sur les données multimédias. Leur exécution est simulée par l'évolution des caractéristiques de QdS des flux en sortie à partir des caractéristiques de QdS des flux reçus en entrée.

Ainsi la Figure 91 propose l'interface — la Face-Avant — d'un composant permettant de réduire la taille des images à 64 pixels sur 98. Il reçoit en entrée les caractéristiques de QdS d'un flux FC2a et fournit en sortie les caractéristiques de ce même flux après traitement par un tel composant de réduction de taille. En particulier, notons que la taille des images — Taille Im — a été modifiée ainsi que la bande passante utilisée — appellée ici Débit. De plus ce composant permet d'accéder à ses caractéristiques de fonctionnement telles que le concepteur les a définies.

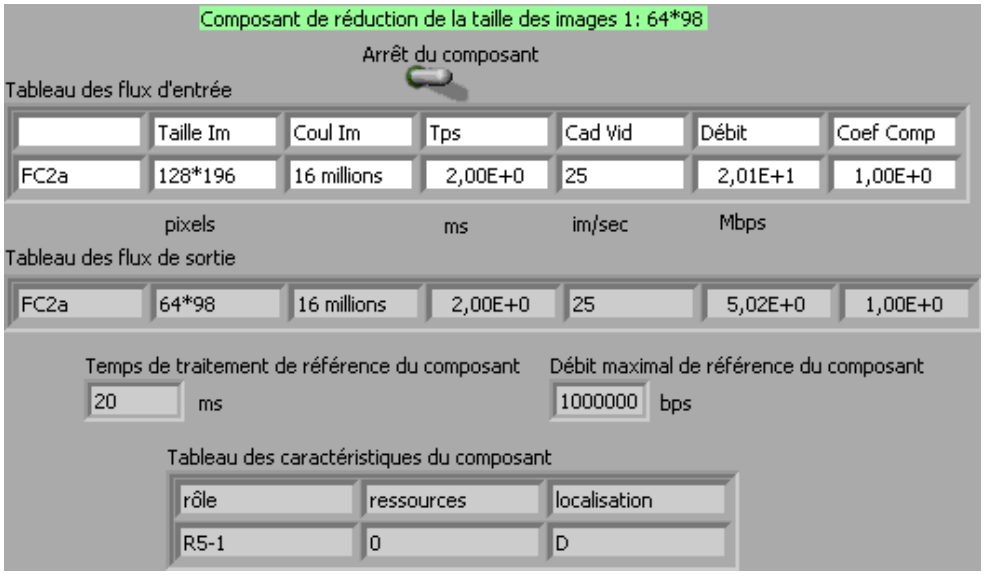


Figure 91 : Résultats de la simulation d'un composant de réduction de la taille des images

L'évaluation de la QdS est alors simulée par la définition des caractéristiques de QdS des flux fournis en sortie de l'application à un instant donné ainsi que par leur notation. La simulation de l'exécution d'une application est obtenue par l'estimation en continu des caractéristiques de QdS de l'ensemble des flux présents dans l'application comme indiqué sur la Figure 92.

Tableau des flux de l'application

	Taille Im	Coul Im	Tps	Cad Vid	Débit	Coef Comp
FA			0,00E+0		0,00E+0	
FC			0,00E+0		0,00E+0	
FA1			1,00E+0		1,00E+0	1,00E+0
FC1	128*196	16 millions	1,00E+0	25	2,01E+1	1,00E+0
FC2a	128*196	16 millions	2,00E+0	25	2,01E+1	1,00E+0
FC3a	64*98	16 millions	2,20E+1	25	5,02E+0	1,00E+0
FA2a			1,00E+3		1,00E+0	1,00E+0
FC4a	64*98	256	1,05E+3	1,99E+1	1,00E+0	1,00E+0
FA3			1,00E+3		1,00E+0	1,00E+0
FC5	64*98	256	1,05E+3	1,99E+1	1,00E+0	1,00E+0
FA4			1,00E+3		1,00E+0	1,00E+0
FC6	64*98	256	1,05E+3	1,99E+1	1,00E+0	1,00E+0

Figure 92 : Simulation de l'exécution d'une application

Simulation du contexte

Le prototype permet de simuler l'état du réseau et des postes informatiques utilisés par l'application. En cours de simulation, il est ainsi possible de faire varier la bande passante disponible entre deux postes ainsi que le temps de transmission associé. La saturation d'un poste est simulée par des seuils représentant le débit maximal des flux locaux et le délai minimal de transmission en local. L'utilisateur du prototype peut également définir un coefficient utilisé pour multiplier le temps de traitement de référence des composants d'un poste et pour diviser le débit de référence de ces composants — ce coefficient est supérieur ou égal à un car cette valeur correspond à la fréquence d'horloge du poste. L'interface qui permet de faire manuellement varier le contexte de chaque poste— des contextes prédéfinis étant également disponibles —est présentée sur la Figure 93.

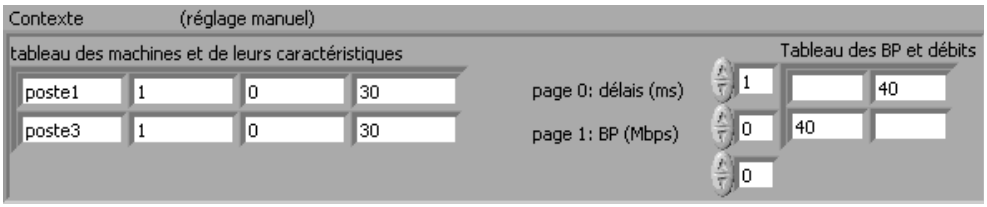


Figure 93 : Simulation du contexte

Constitution de l'application

Chaque composant et chaque rôle atomique correspondent à un *V.I.* Le *V.I.* d'un rôle agit sur un ou plusieurs flux de données comme le montre le code graphique du rôle de réduction de la taille des images sur la Figure 94.

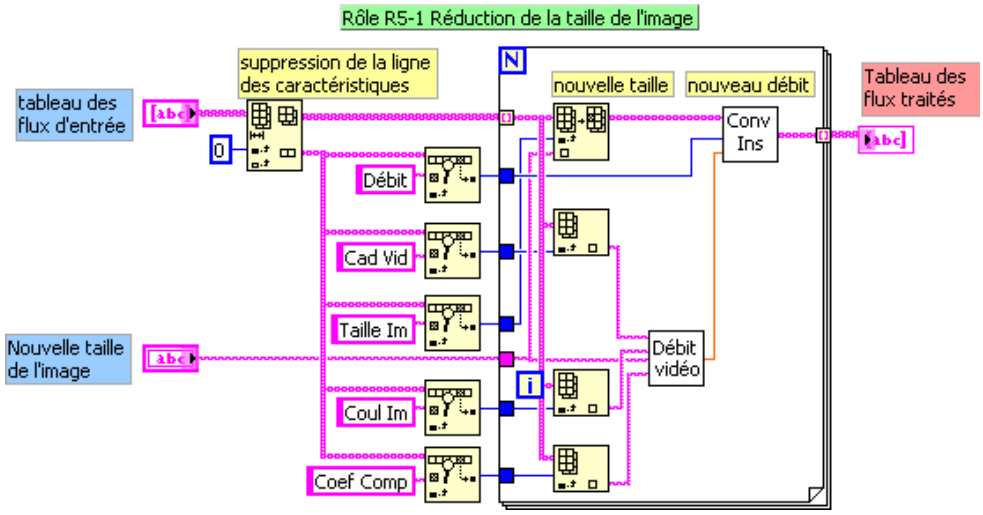


Figure 94 : Exemple de rôle atomique : réduction de la taille des images

Un rôle est paramétrable de manière à être utilisé pour réaliser les *V.I.* des composants réalisant ce rôle, les paramètres permettant d'obtenir des qualités intrinsèques différentes. Ainsi le composant de réduction de la taille des images dont le code est présenté sur la Figure 95 utilise le rôle précédent (cf. Figure 94) pour une taille d'image particulière — 64 pixels sur 98.

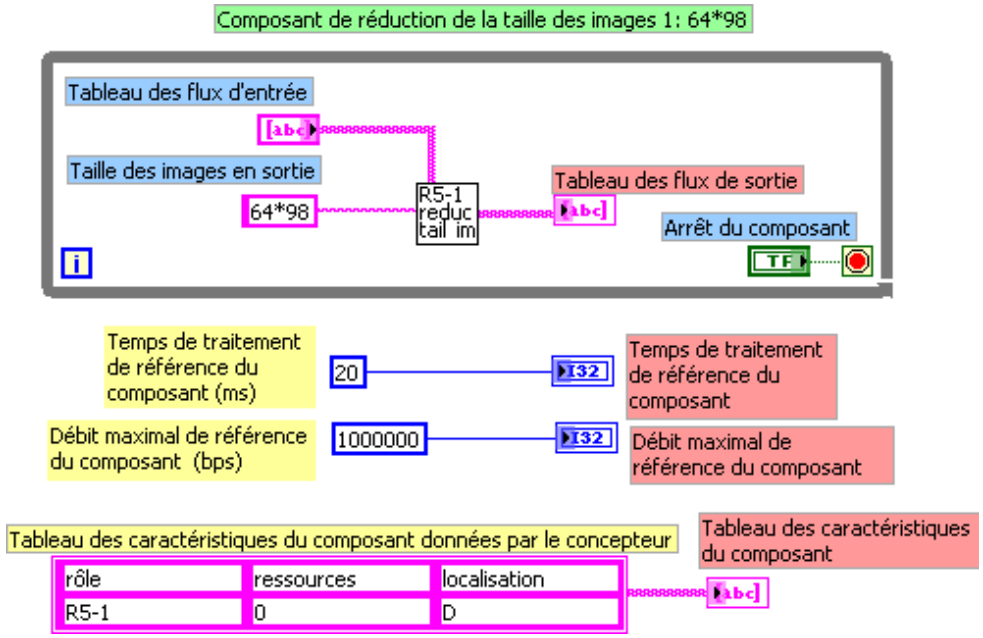


Figure 95 : Simulation d'un composant de réduction de la taille des images

L'appel dynamique des composants peut être réalisé grâce à l'utilisation d'un modèle unique de composant pour programmer tous les composants de l'application. Celui-ci permet de définir les entrées et sorties des composants comme indiqué par la Figure 96.

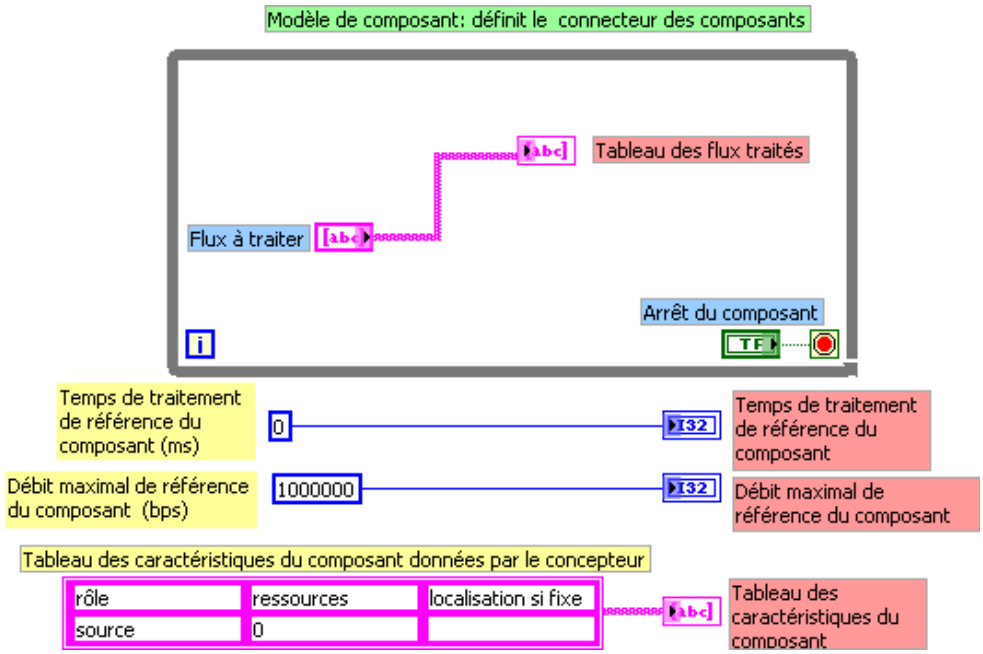


Figure 96 : Modèle de composant de l'application

De manière à optimiser l'écriture du code, les traitements génériques aux composants ont été codés dans un *V.I.* réalisant la fonction du Processeur Élémentaire (cf. Figure 97 p.226) alors que ce qui est propre à un composant et à sa logique métier est implanté dans le *V.I.* de ce composant. Ainsi, les valeurs du temps de traitement et des débits maximaux sont implantés dans le *V.I.* du composant alors que leur influence sur les flux est matérialisée par le *V.I.* Processeur Élémentaire. De plus, différents noms sont attribués aux flux de l'application. C'est le *V.I.* Processeur Élémentaire qui se charge alors de renommer les flux en sortie d'un composant. Ce *V.I.* permet également de simuler l'influence des performances de la machine hébergeant les composants.

Un *V.I.* Conduit simule l'utilisation des Conduits pour transmettre les données et conserver leur synchronisation (cf. Figure 98 p.227). Il permet en particulier de tenir compte de l'état du réseau et des machines pour définir les flux transmis en local ou de manière répartie. Des opérateurs de duplication de flux sont utilisés pour dédoubler un flux issu d'un seul Conduit mais devant être envoyé à plusieurs composants.

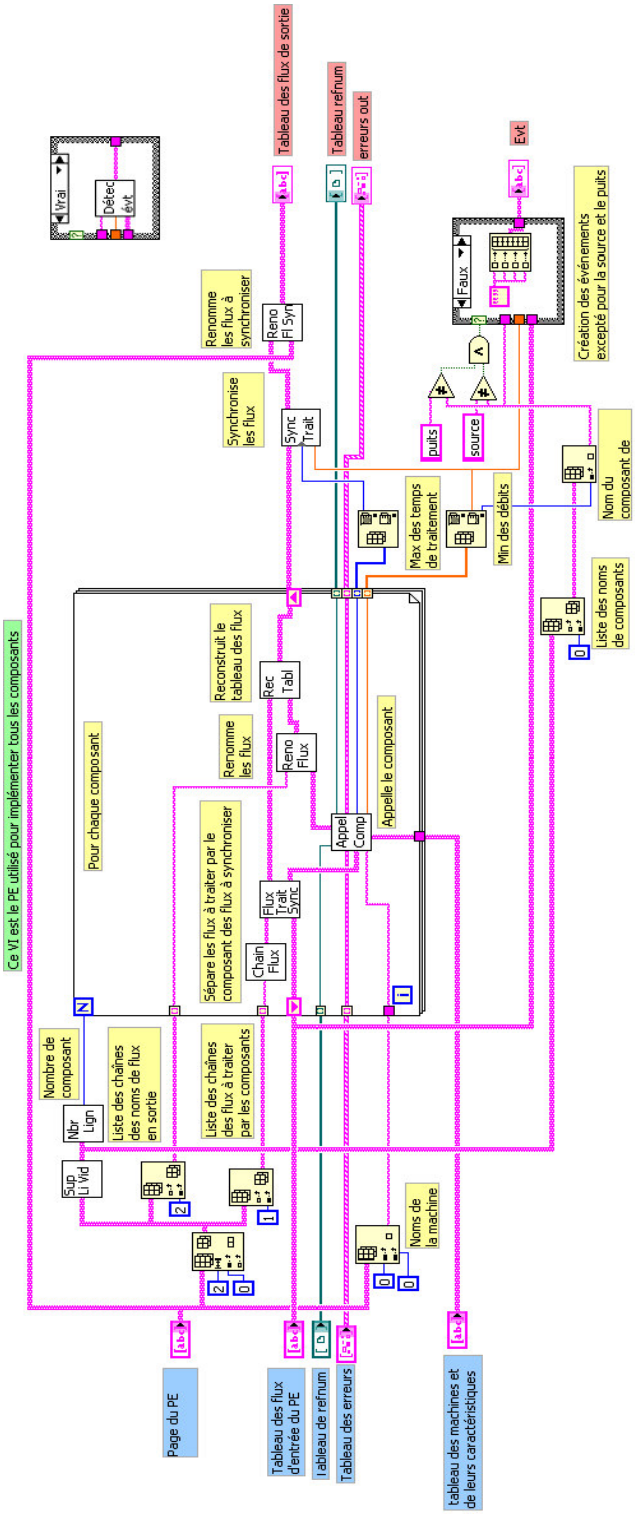


Figure 97 : Code du *V.I.* Processeur Élémentaire

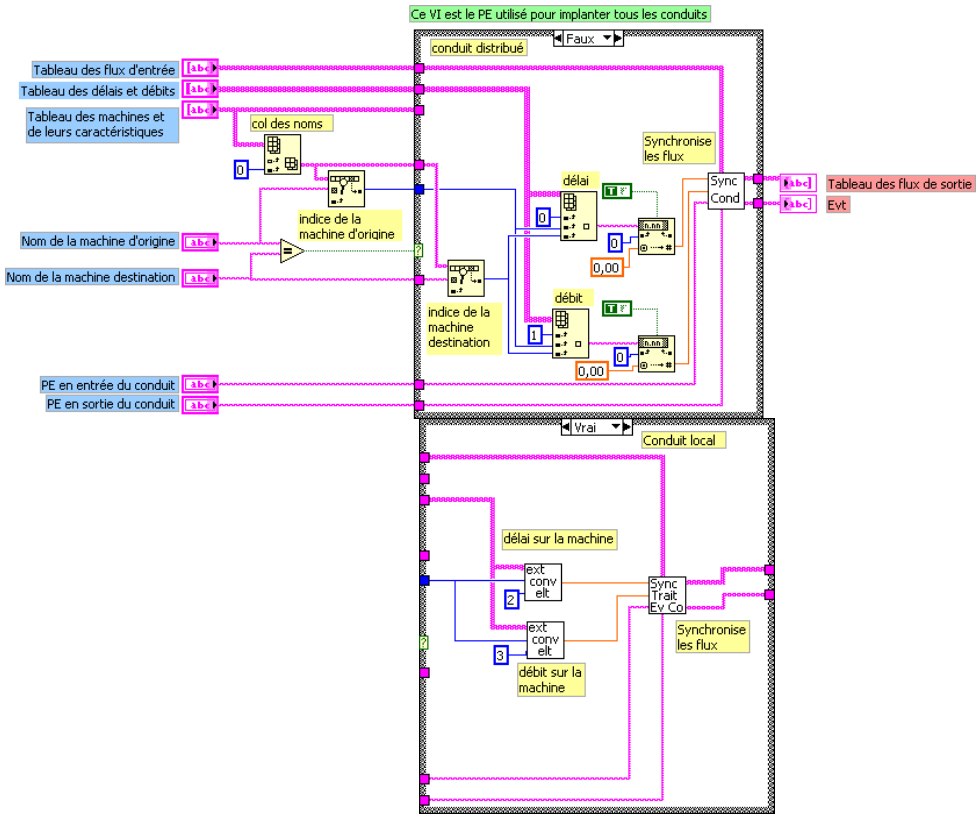


Figure 98 : Code du *V.I.* Conduit

Structure de l'application

Dans le prototype, les graphes représentant une ou toutes les décompositions fonctionnelles ainsi que ceux décrivant les configurations sont représentés par différents tableaux dont une matrice d'adjacence :

- le tableau des Processeurs Élémentaires qui identifie les Processeurs Élémentaires utilisés, les rôles des composants — pour les décompositions fonctionnelles —ou les composants contenus dans chaque Processeur Élémentaire — pour les configurations — et les flux en entrée et en sortie ;
- le tableau des Conduits qui indique quels flux transitent par quel Conduit et à quel lien de synchronisation ils doivent obéir ;
- la matrice d'adjacence qui décrit le graphe de la configuration ou de la décomposition fonctionnelle en définissant les Processeurs Élémentaires à l'origine et à la destination de chaque Conduit ;
- le tableaux des opérateurs de duplication.

Une configuration est également décrite par un tableau permettant de l'identifier en donnant son nom, les Groupes, les assemblages de Sous-Groupes et les décompositions fonctionnelles qu'elle utilise. Ces éléments de définition sont regroupés dans une structure

appelée *cluster* — groupement de données de types différents dans le langage G — qui contient également la note de QdS ainsi que les notes des critères intrinsèque et contextuel. La Figure 99 décrit ainsi le *cluster* définissant une configuration de l'application de vidéosurveillance que nous allons utiliser comme application de test par la suite (voir sa définition au §3.3.3.3.a p.231).

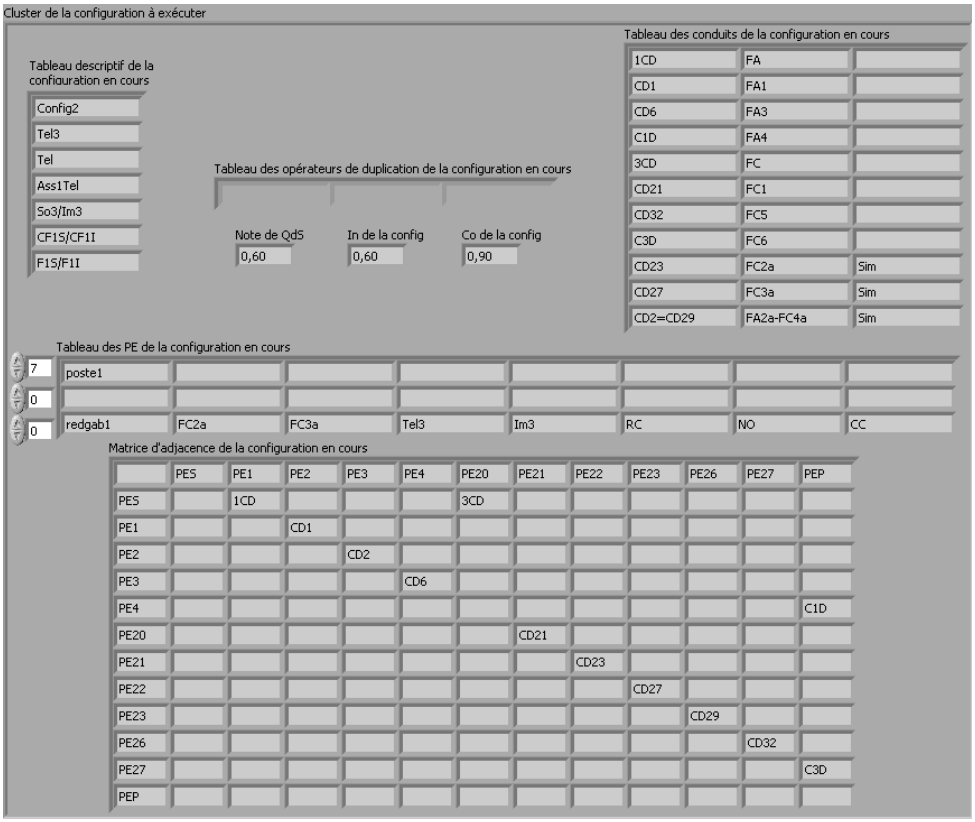


Figure 99 : Représentation d'une configuration de vidéosurveillance

b. Structure du prototype

Le prototype est composé de quatre sous-programmes regroupés dans le *V.I.* principal de la Figure 100 page 230.

Le premier se charge du déploiement de la plate-forme sur les différents postes utilisables par une application ainsi que de sa suppression en fin d'exécution. Il définit les identifiants — références numériques — utilisés par LabVIEW pour assurer la communication entre les parties de la plate-forme présentes sur les postes et en particulier les gestionnaires d'évaluation et de supervision.

La deuxième partie simule partiellement la plate-forme. Elle se charge de la gestion des événements, de l'évaluation et de la communication permettant une évaluation répartie entre les différents gestionnaires.

La troisième partie simule l'application et le gestionnaire de supervision de manière à répartir la réalisation d'une reconfiguration entre les différents gestionnaires.

La dernière partie permet de visualiser sur un chronogramme l'évolution de la QdS de l'application en fonction du temps. Elle fournit également une représentation des différentes QdS atteintes par l'application lors de son exécution sous la forme de l'ensemble des points de coordonnées (Co ; In) atteints. Cette partie constitue donc le tableau de bord grâce auquel nous pouvons suivre les résultats obtenus par la plate-forme.

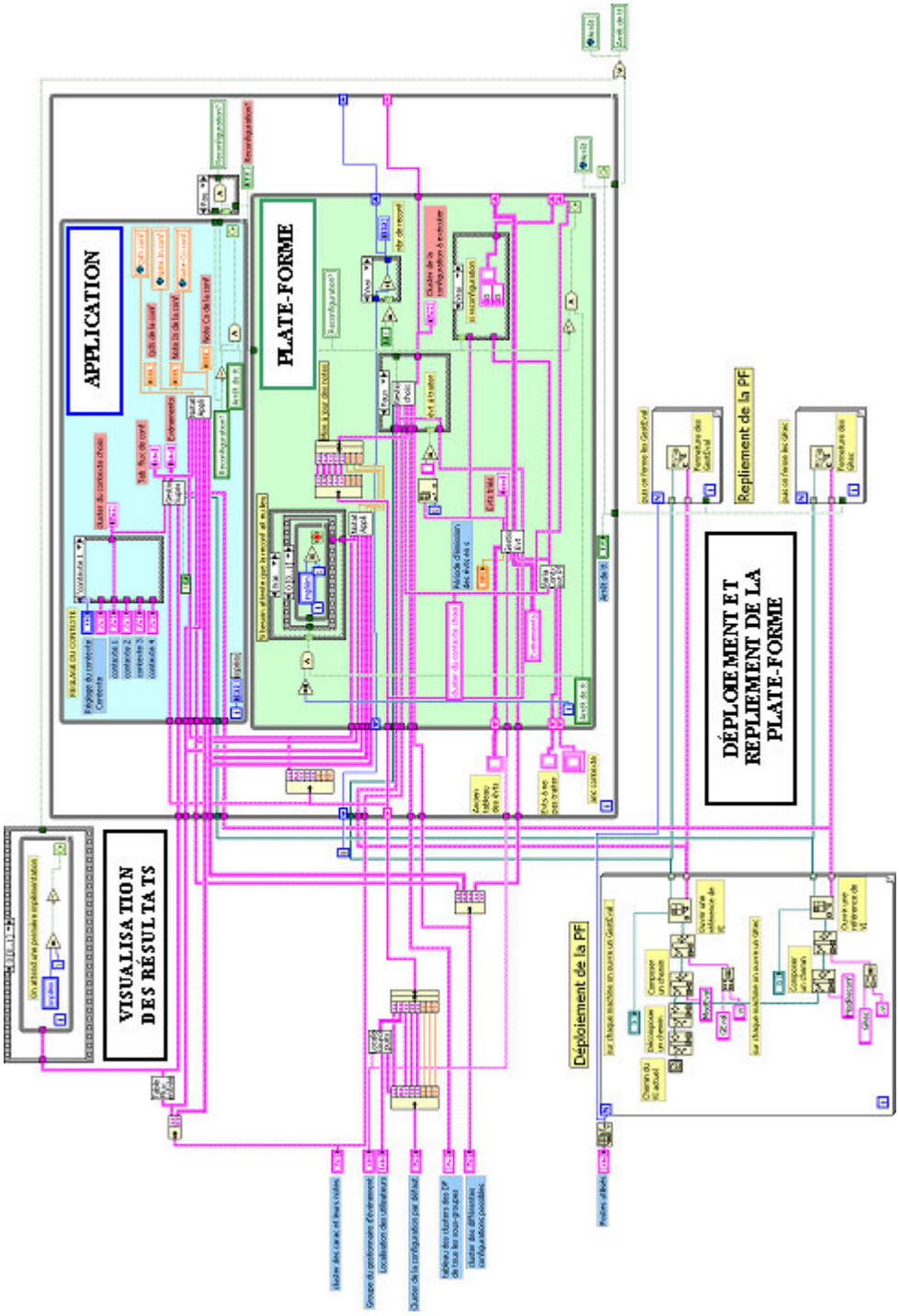


Figure 100 : Code du programme principal du prototype

c. Fonctionnement du prototype

La plate-forme dispose d'informations fournies par le concepteur de l'application et par les utilisateurs.

Le concepteur de l'application a préalablement défini :

- les Groupes, les Sous-Groupes, les assemblages possibles de Sous-Groupes ;
- les différentes décompositions fonctionnelles disponibles pour chaque Groupe et les familles qu'ils constituent ;
- les composants disponibles et les rôles atomiques associés ainsi que leurs caractérisations — critique, délocalisable etc ;
- les caractéristiques de QdS qui seront utilisées pour l'évaluation de la QdS et des critères intrinsèque et contextuel ;
- la configuration par défaut de chaque Groupe et Sous-Groupe.

Chaque utilisateur a choisi son Groupe et classé les assemblages de Sous-Groupes disponibles. Il a exprimé ses souhaits de qualité en donnant un poids aux Sous-Groupes et des notes aux caractéristiques de QdS proposées.

En fonction de la conception de l'application, des vœux des utilisateurs et des informations décrivant le contexte, le prototype simule le fonctionnement de l'application et de la plate-forme.

Il simule l'implantation des parties locales de la plate-forme puis déploie dynamiquement l'application en utilisant les configurations par défaut. L'application va alors s'exécuter en continu jusqu'à ce qu'une reconfiguration soit nécessaire. Le prototype permet de visualiser à tout instant la note de QdS de l'application.

En parallèle, la plate-forme recueille les événements générés par les Processeurs Élémentaires et les Conduits. Elle identifie celui de plus haute priorité et recherche, si nécessaire, une configuration meilleure. Si elle échoue dans sa quête, elle étudie un autre événement. Dans le cas contraire, elle envoie un ordre de reconfiguration au gestionnaire de supervision et l'application est reconfigurée.

Lorsque le prototype est interrompu par l'utilisateur, l'application est arrêtée, les composants sont supprimés ainsi que les parties de la plate-forme sensées se situer sur les différents postes.

3.3.3.3. Expérimentation

a. Application de test

L'application de test est une vidéosurveillance permettant de transmettre les images et le son captés sur un site à un utilisateur distant.

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 identiques à celles définies pour l'exemple de la vidéoconférence et présentées sur la Figure 101.

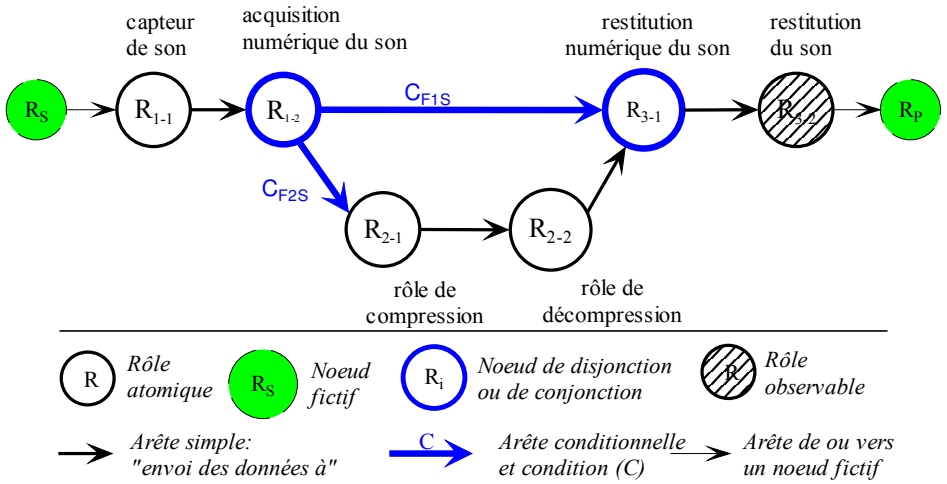


Figure 101 : Différentes décompositions fonctionnelles du Sous-Groupe *So*

Pour transmettre les images, le Sous-Groupe *Im* propose six décompositions fonctionnelles identiques à celles définies pour l'exemple de la vidéoconférence et présentées sur la Figure 102.

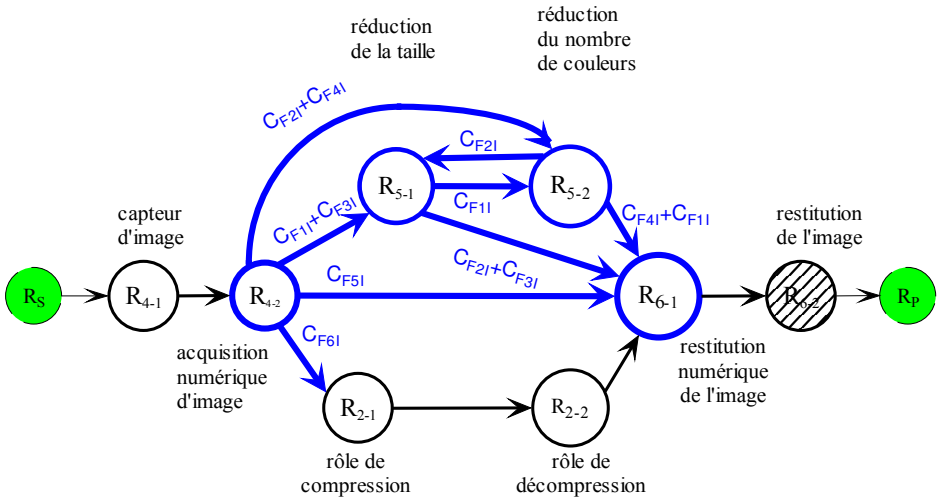


Figure 102 : Différentes décompositions fonctionnelles du Sous-Groupe *Im*

La plate-forme dispose d'un composant pour chaque rôle atomique utilisé à l'exception de la réduction de la taille des images qui peut être réalisée par deux composants différents : le premier pour une réduction au format de 64 pixels sur 98 et le second au format de 48 sur 64.

Le Sous-Groupe *So* ne contient ni composant critique ni rôle critique. Les différents niveaux de définition des familles permettent d'obtenir soit une seule famille pour RC — rôle critique — et CC — composant critique —, soit deux familles pour RnC — rôle non critique — et CnC — composant non critique. Dans cet exemple, nous supposons que le concepteur a défini une unique famille regroupant les deux configurations possibles et qu'il a choisi de ne pas utiliser de caractéristiques intrinsèques de QdS. En revanche, le critère contextuel regroupera deux caractéristiques : le débit du flux sonore et le temps de traitement nécessaire à sa restitution.

Le Sous-Groupe *Im* contient deux rôles critiques — pour les rôles $R_{5\text{-}1}$ et $R_{5\text{-}2}$ — et un composant critique réalisant la réduction de la taille des images — pour le rôle $R_{5\text{-}1}$. Les différentes manières de constituer les familles ont été définies préalablement (cf. Tableau 15 p.170). Nous supposons que le concepteur a choisi de définir une famille comme un ensemble de configurations ayant les mêmes rôles critiques — RC — ce qui permet de disposer des quatre familles suivantes :

- la première contenant les deux décompositions fonctionnelles avec réduction de la taille des images et du nombre de bits de codage des couleurs soit CF1I et CF2I ;
- la deuxième constituée par la décomposition fonctionnelle réduisant uniquement la taille des images soit CF3I ;
- la troisième composée de la décomposition fonctionnelle réduisant uniquement le nombre de bits de codage des couleurs soit CF4I ;
- enfin la quatrième regroupant les deux décompositions fonctionnelles qui n'affectent ni la taille ni le nombre de couleurs des images soit CF5I et CF6I.

Les caractéristiques de QdS permettant d'évaluer le Sous-Groupe transmettant les images sont alors la taille et le nombre de couleurs des images pour le critère intrinsèque ainsi que la cadence vidéo et le temps de restitution pour le critère contextuel.

Nous supposons que l'utilisateur a exprimé les préférences décrites par la Figure 103 et la Figure 104.

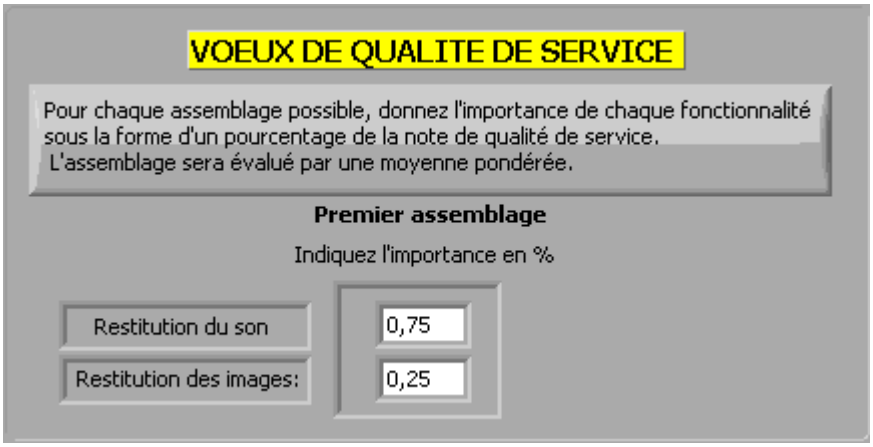


Figure 103 : Importance des Sous-Groupes définie par l'utilisateur

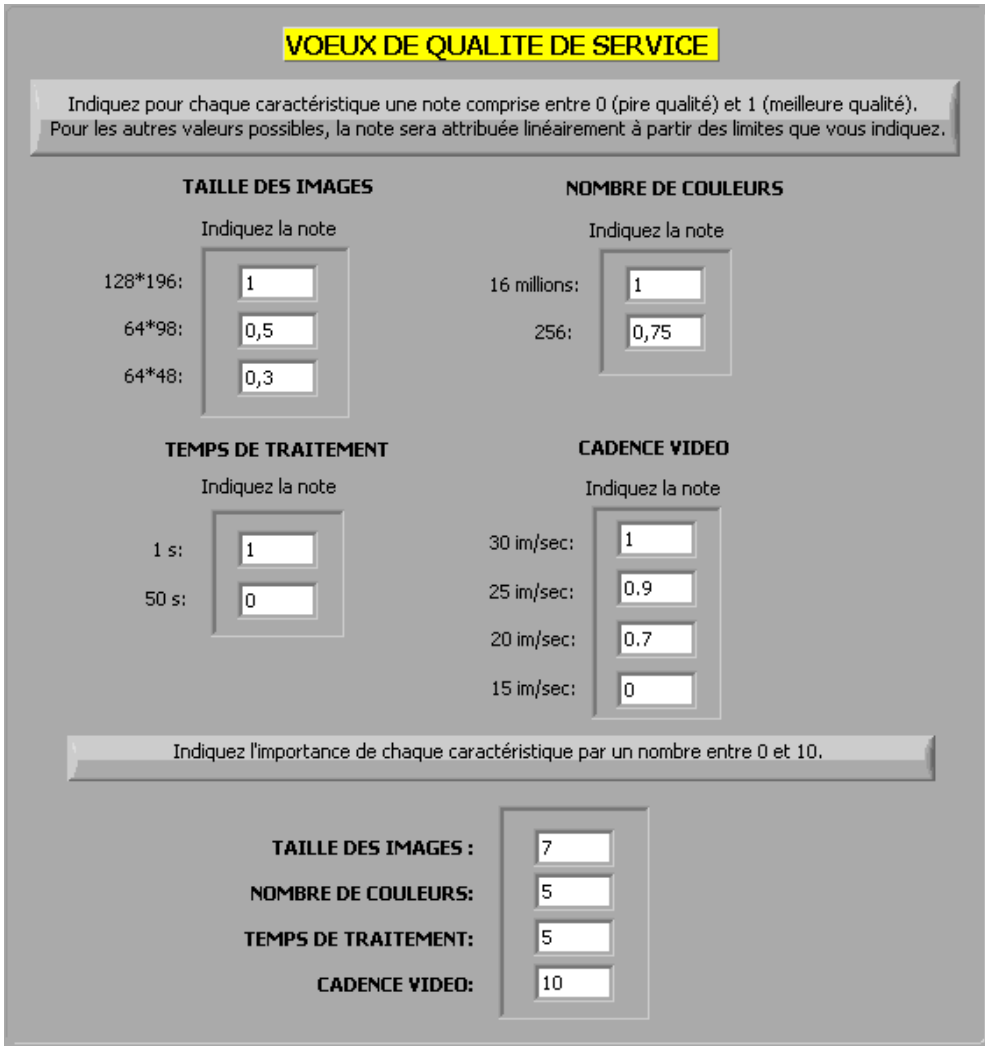


Figure 104 : Vœux de l'utilisateur pour les caractéristiques de QdS

La complexité théorique de cette application peut être évaluée par le nombre maximal de configurations donné par $\left[\left(C.S^{R}.F\right)^{G}.H\right]^{U}$ qui, dans ce cas volontairement très simple, vaut tout de même 768 car U=1, H=1, F=6, C=2, S=2, G=2 et R=6. Elle ne tient compte ni des contraintes matérielles ni des contraintes fonctionnelles.

La complexité réelle est inférieure à ce nombre car elle tient compte des décompositions fonctionnelles possibles, des P postes disponibles et des différents choix de composants. Le nombre réel de configurations possibles est de 135 (nous excluons la duplication de composant qui a peu d'intérêt lorsqu'un seul utilisateur est considéré).

En effet, l'application contient un Groupe réalisable par un seul assemblage composé des deux Sous-Groupes — *So* et *Im*. Le Sous-Groupe *So* peut être réalisé soit par la décomposition fonctionnelle CF1S où tous les composants sont indélocalisables, soit par la décomposition fonctionnelle CF2S où deux composants sont délocalisables sur P postes ce

qui donne P^2 combinaisons : le Sous-Groupe *So* dispose donc de $1+P^2$ configurations pour être implanté.

Les six décompositions fonctionnelles du Sous-Groupe *Im* peuvent utiliser quatre composants délocalisables et un rôle réalisable par deux composants différents. Par le même raisonnement, on démontre que ce Sous-Groupe peut être implanté par $5.P^2+3.P+1$ configurations.

L'application dispose donc de $(5.P^2+3.P+1)(1+P^2)$ où P=2 soit 135 configurations.

La configuration par défaut (cf. Figure 105), choisie par le concepteur, utilise les décompositions fonctionnelles CF1S pour le Sous-Groupe *So* et CF1I pour le Sous-Groupe *Im*. Le composant de réduction de la taille des images est $C_{5\text{-}1a}$ fournissant 64 pixels sur 48. Il est placé sur le poste émetteur de même que le composant de réduction du nombre de couleurs $C_{5\text{-}2}$.

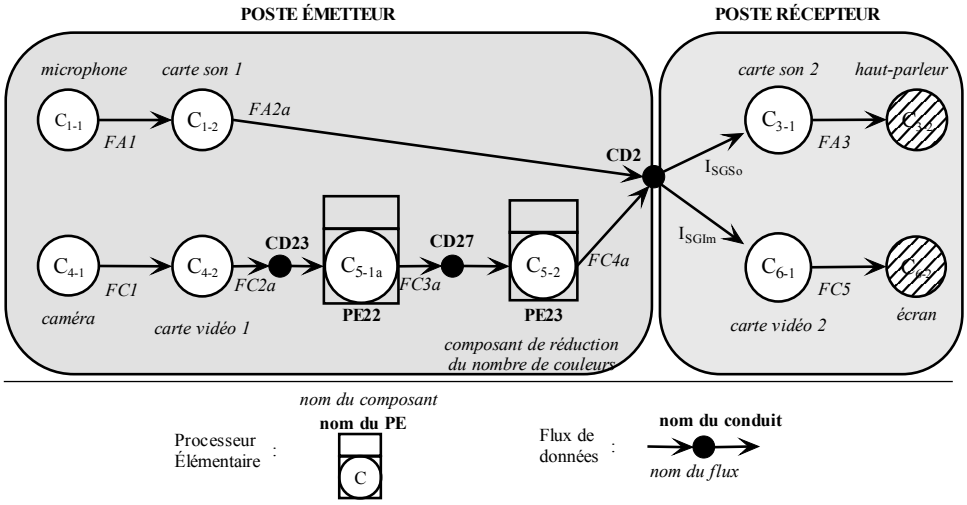


Figure 105 : Configuration par défaut utilisée dans les tests

b. Tests

Trois tests principaux ont été effectués soit dans un contexte statique soit dans un contexte dynamique. Tout d'abord, le déploiement de l'application a été étudié dans le cas d'un contexte stable qu'il soit favorable comme dans le premier test ou défavorable comme dans le deuxième. Puis l'étude en dynamique a permis de caractériser le comportement de la plate-forme et de l'application lorsqu'un contexte favorable se dégrade puis revient à son état antérieur.

Test n°1 : Déploiement dans un contexte favorable

Le déploiement de l'application a tout d'abord été étudié dans un contexte favorable c'est-à-dire dans lequel le temps de transmission entre les deux postes est très faible — 1 seconde —, la bande passante est élevée — 40Mbps — et les deux postes ne sont pas saturés — coefficient des temps de traitement de 1, temps de transmission de 0s et débit maximum de 30 Mbps — ce qui correspond à la Figure 106.

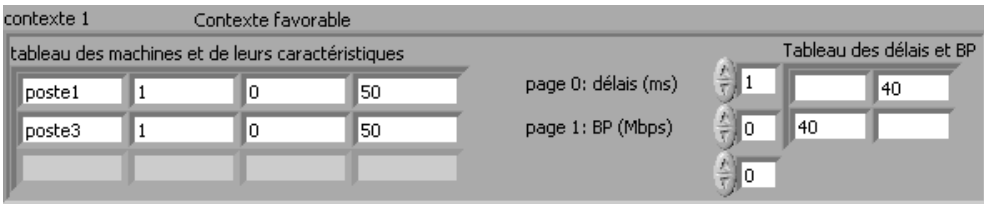


Figure 106 : Contexte favorable de déploiement utilisé

Dans ces conditions la note de QdS obtenue par la configuration par défaut est QdS1=0,49 pour une note du critère intrinsèque de 0,49 et une note du critère contextuel de 0,99. La plate-forme propose alors une reconfiguration suite à l'étude d'un événement généré par le Conduit CD27 placé en sortie du composant de réduction de la taille $C_{5\text{-}1a}$ qui détecte que la qualité peut être améliorée — événement de type E2 — et désigne donc ce composant comme le composant problématique. La nouvelle configuration est obtenue par le remplacement de ce composant de réduction de la taille des images par celui fournissant une image plus grande soit $C_{5\text{-}1b}$ qui fournit 64 pixels sur 98 au lieu de 64 pixels sur 48 (cf. Figure 107). La nouvelle note de l'application est alors de QdS2=0,60. Elle est identique à la note du critère intrinsèque alors que la note du critère contextuel est de 0,90. L'évolution de la QdS en fonction du temps est définie par les courbes de la Figure 108 et de la Figure 109.

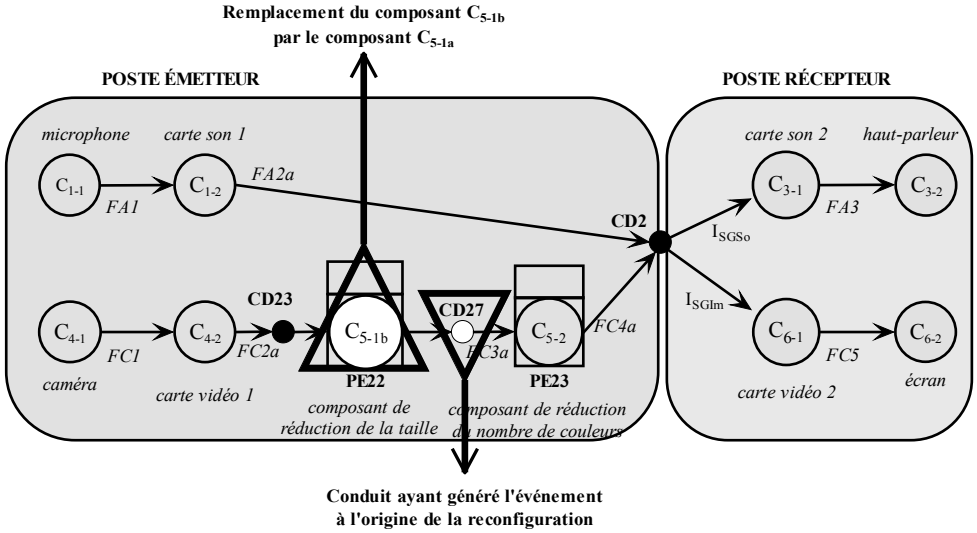


Figure 107 : Première reconfiguration dans un contexte favorable

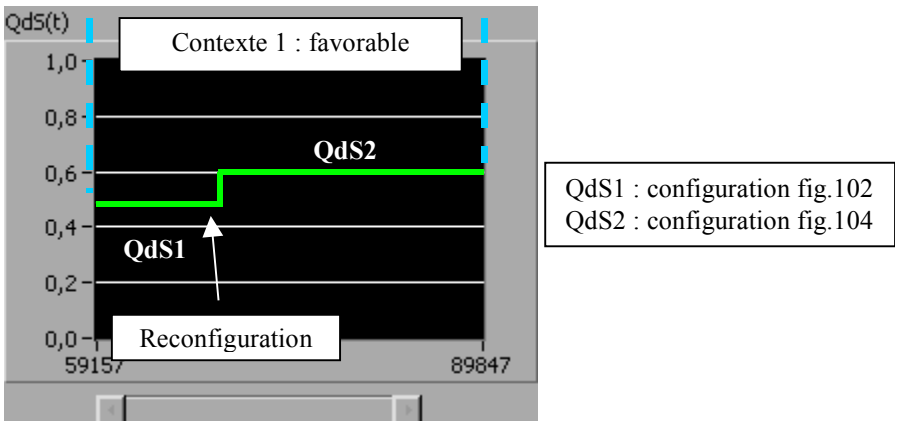


Figure 108 : Évolution de la QdS en fonction du temps lors du déploiement dans un contexte favorable

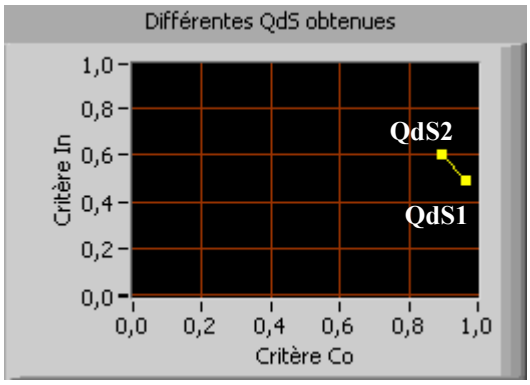


Figure 109 : Évolution des critères de QdS lors du déploiement dans un contexte favorable

En partant d'une configuration par défaut minimaliste, dans le sens où le concepteur sait qu'elle n'est pas optimale, la plate-forme permet donc d'améliorer la QdS. Elle trouve alors une meilleure configuration et, dans notre exemple, la meilleure configuration possible est atteinte en une seule reconfiguration. Son comportement est alors stable puisque ni la QdS ni la configuration atteinte ne fluctuent lorsque le contexte reste stable.

Test n°2 : Déploiement dans un contexte défavorable

Le deuxième test réalisé consiste à déployer la même configuration (cf. Figure 105 p.235) dans un contexte rendu défavorable par une charge élevée sur le poste émetteur. Cette saturation est simulée par un fort coefficient d'exécution des composants, un temps de transmission de 1s et un débit maximal de 2 Mbps.

La QdS de la configuration par défaut est alors QdS1=0,48 ce qui correspond également à la note du critère contextuel. En revanche, la note du critère intrinsèque est de 0,49. La plate-forme réalise alors une première reconfiguration (cf. Figure 110 p.238) à l'issue de l'étude d'un événement généré par le Processeur Élémentaire PE23 contenant le composant $C_{5\text{-}2}$ réduisant le nombre de couleurs qui détecte une dégradation de qualité liée au contexte — événement de type E1B — et indique que $C_{5\text{-}2}$ est le composant problématique. L'adaptation consiste à déplacer ce composant sur le poste récepteur qui n'est pas saturé.

La QdS obtenue est alors QdS2=0,49 ce qui est également la note du critère intrinsèque alors que la note du critère contextuel a atteint 0,54.

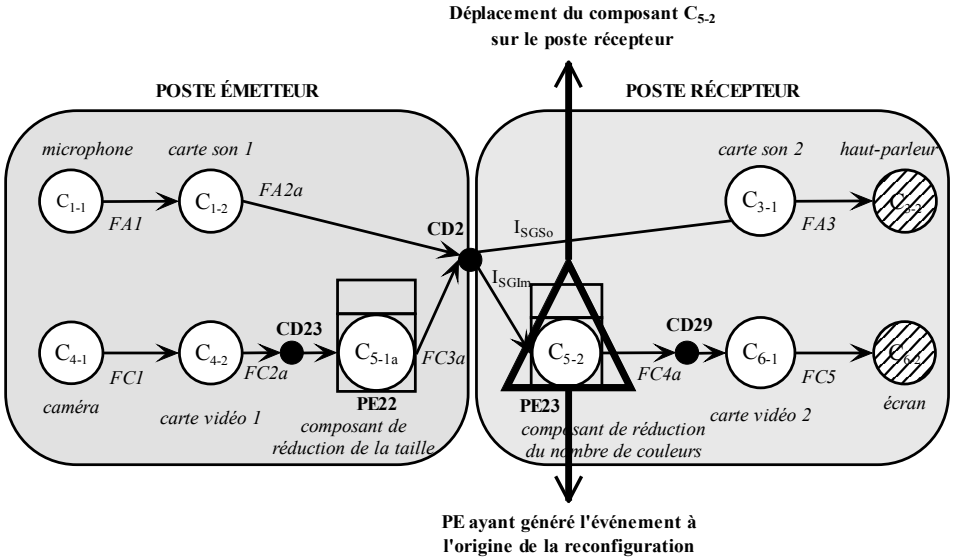


Figure 110 : Première reconfiguration dans un contexte défavorable

Puis la plate-forme reconfigure une seconde fois l'application en remplaçant le composant de réduction de la taille des images $C_{5\text{-}1a}$ par le second composant disponible $C_{5\text{-}1b}$ et en le déplaçant sur le poste récepteur (cf. Figure 111). L'événement à l'origine de cette délocalisation a été généré par le même Processeur Élémentaire PE23 que précédemment mais celui-ci a indiqué que le composant problématique était le composant de réduction de la taille $C_{5\text{-}1a}$ car le composant $C_{5\text{-}2}$ lui succédant était sous-employé et disposait donc d'une marge de manœuvre — événement de type E2. La qualité obtenue est alors QdS3=0,60. C'est également la note du critère intrinsèque alors que le critère contextuel vaut 0,64. L'évolution de la QdS en fonction du temps est définie par les courbes de la Figure 112 et de la Figure 113.

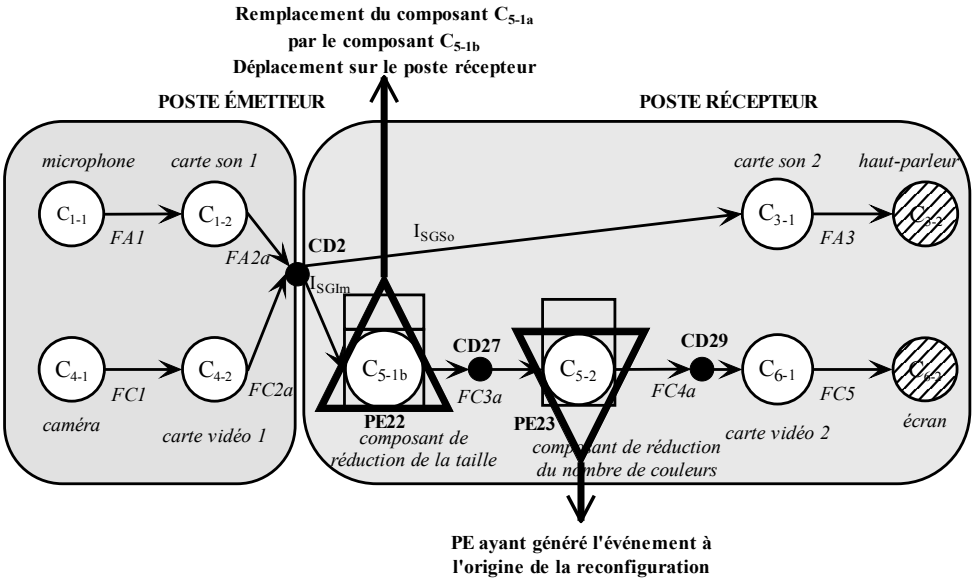


Figure 111 : Seconde reconfiguration dans un contexte défavorable

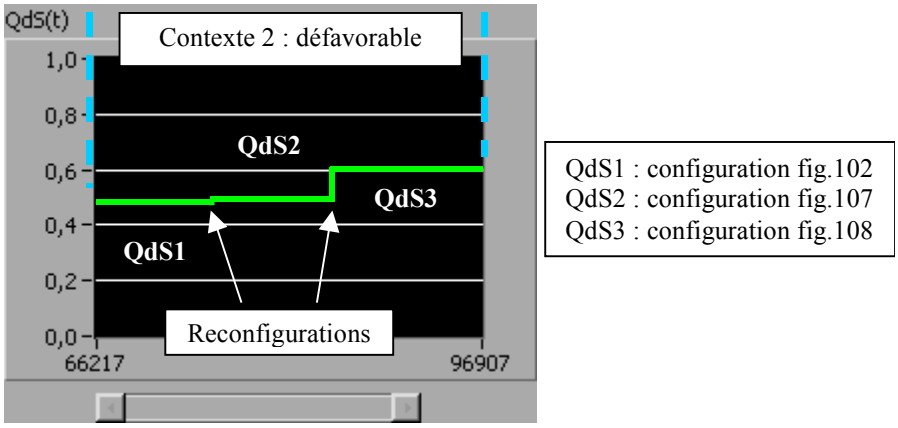


Figure 112 : Évolution de la QdS en fonction du temps lors du déploiement dans un contexte défavorable

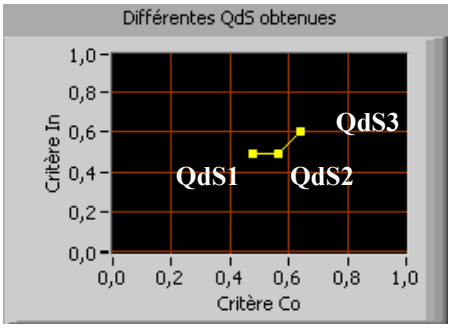


Figure 113 : Évolution des critères de QdS lors du déploiement dans un contexte défavorable

Lorsque le contexte l'exige, la plate-forme utilise donc différentes manières de modifier l'application pour améliorer sa QdS. Dans notre exemple, il s'agit d'une part, du remplacement d'un composant par un autre et d'autre part, du déplacement de composants. Les reconfigurations successives permettent alors de converger vers la meilleure configuration grâce à des événements issus aussi bien des Conduits que des Processeurs Élémentaires. Nous remarquons que, conformément aux algorithmes proposés (cf. Figure 69 p.178), la plate-forme déploie un nouveau composant sur le poste qui permet d'obtenir la meilleure QdS puisque, dans le cas étudié, le nouveau composant de réduction de la taille des images est directement implanté sur le poste récepteur alors que le composant qu'il remplace était sur le poste émetteur.

Test n°3 : Évolution de l'application lors d'une dégradation suivie d'une amélioration du contexte

Ce test permet d'étudier le comportement de la plate-forme lorsque le contexte oscille c'est-à-dire qu'il se dégrade puis s'améliore puis se dégrade à nouveau. Au départ, l'application est déployée dans une configuration stable (QdS1 obtenue après un déploiement avec une qualité de QdS0). Le contexte de départ est celui du test 1 (cf. Figure 106 p.236). Nous le faisons évoluer par une saturation de l'émetteur comparable à celle utilisée dans le test 2 (cf. p.237) puis revenir au contexte du test 1 et enfin reprendre les caractéristiques du test 2. La QdS initiale est QdS1=0,60. L'application utilise le composant $C_{5\text{-}1b}$ de réduction de la taille des images à 64 pixels sur 98 placé sur le poste émetteur comme le composant $C_{5\text{-}2}$ de réduction du nombre de couleurs (configuration identique à celle de la Figure 107 page Figure 107).

La dégradation du contexte provoque une baisse de la QdS à QdS2=0,48 ce qui correspond à la note du critère contextuel alors que la note du critère intrinsèque Rest à 0,60. La plate-forme reconfigure une première fois pour atteindre une qualité de service QdS3 de 0,57 pour un critère intrinsèque à 0,60 et un critère contextuel à 0,57 grâce au déplacement du composant de réduction du nombre de couleurs $C_{5\text{-}2}$ sur le poste récepteur.

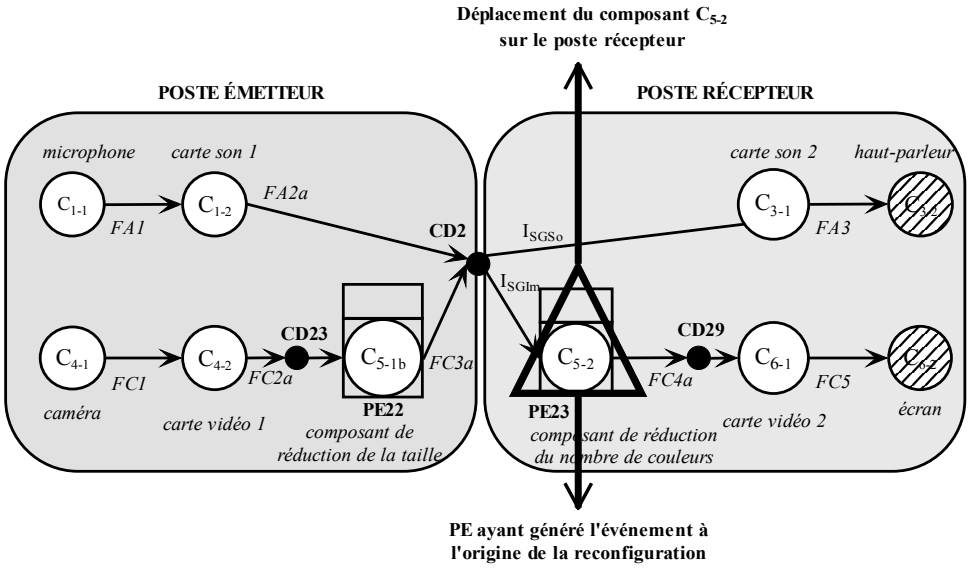


Figure 114 : Première reconfiguration lors du troisième test

Puis elle reconfigure une seconde fois pour améliorer le critère contextuel à 0,64 ce qui permet d'atteindre une qualité de service QdS4 de 0,60 en déplaçant le composant $C_{5\text{-}1b}$ de réduction de la taille des images sur le poste récepteur (configuration identique à celle de la Figure 111 page 239). La situation est alors stable. Grâce aux reconfigurations, la qualité atteinte est égale à celle obtenue dans le contexte antérieur qui était pourtant meilleur.

Dans une seconde phase, nous modifions le contexte afin qu'il retrouve son état initial. La QdS demeure à QdS5=0,60 ce qui était sa valeur initiale. C'est également la note intrinsèque alors que la note contextuelle est de 0,67. La plate-forme ne propose alors plus de reconfiguration car la configuration utilisée est la meilleure.

Enfin lorsque le contexte se dégrade à nouveau la QdS ne varie pas et la plate-forme ne reconfigure pas 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 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 de l'application justifie donc a posteriori l'algorithme utilisé par la plate-forme.

La Figure 115 et la Figure 116 décrivent l'évolution de la QdS lors de ces variations du contexte.

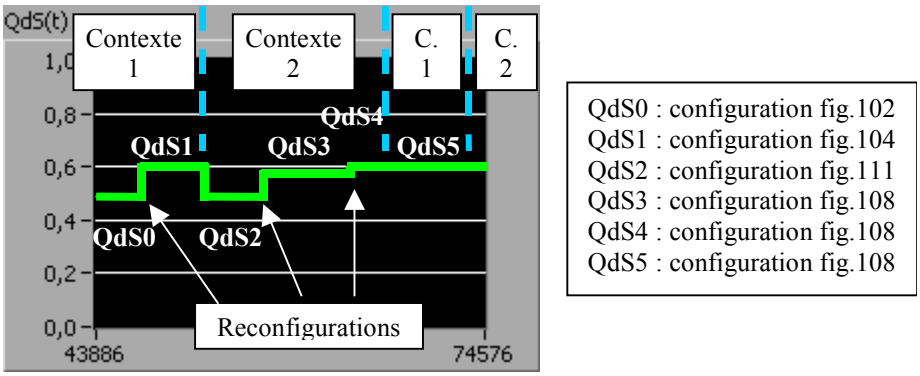


Figure 115 : Évolution de la QdS en fonction du temps lors d'une dégradation du contexte

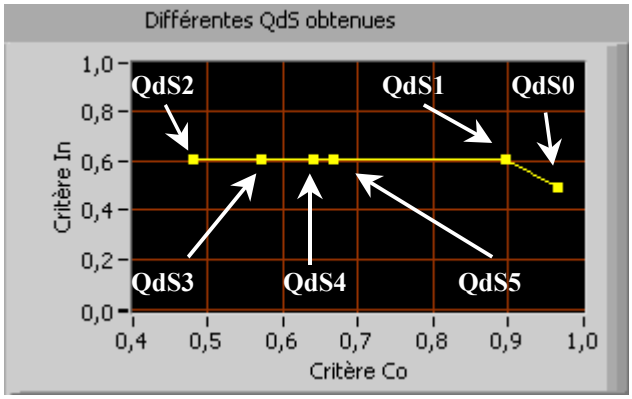


Figure 116 : Évolution des critères de QdS lors d'une dégradation du contexte

Ce troisième 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 identique mais en utilisant une configuration différente puisque les deux composants délocalisables restent sur le poste récepteur. Ceci illustre bien le fait que seul le résultat perceptible par l'utilisateur a de l'importance. De plus, en ne cherchant pas l'optimum, la plate-forme stabilise le service et donne une plasticité meilleure et donc un comportement meilleur à l'application. Une recherche exhaustive de la meilleure configuration — si elle était possible — ne présenterait pas cette plasticité ni ce pouvoir stabilisant.

c. Résultats

Dans tous les tests effectués, chaque reconfiguration a permis d'améliorer la QdS. L'enchaînement de ces reconfigurations a abouti a une stabilisation de l'application dans une configuration donnée lorsque le contexte est stable et même dans certains cas lorsqu'il oscille. Nous pouvons donc affirmer qu'une itération de l'heuristique permet d'améliorer la QdS. De plus, l'heuristique converge en un nombre d'itérations qui s'est révélé faible — au maximum 2 —pour l'exemple simple étudié. Enfin, les différentes possibilités de reconfiguration proposées ont effectivement été mises en œuvre par la plate-forme — délocalisation, changement de composant, etc. Enfin, l'évolution de la QdS obtenue est graduelle conformément à nos vœux.

3.3.4. Synthèse

L'efficacité de l'heuristique que nous avons proposée pour définir les reconfigurations dépend en partie du choix de l'événement qui permet d'identifier le composant problématique. C'est pourquoi nous avons élaboré un modèle de gestionnaire d'événements qui permet à la plate-forme d'intervenir sur la partie de l'application la plus critique pour l'utilisateur. Nous avons ensuite déterminé que le traitement en parallèle des événements pouvait être réalisé soit au niveau d'un Sous-Groupe soit au niveau d'un Groupe en fonction des composants communs.

Puis nous avons proposé un modèle structurel de la plate-forme permettant la gestion répartie des événements mais également de l'évaluation dans le souci de respecter les contraintes temporelles des applications multimédias. La partie locale de la plate-forme est composée de cinq gestionnaires — gestionnaire d'événements, gestionnaire d'évaluation, gestionnaire de communication, gestionnaire d'utilisateur et gestionnaire de supervision — et est répartie sur tous les postes utilisés par l'application.

Enfin, notre prototype nous a permis de vérifier la validité du modèle de plate-forme proposé et en particulier l'obtention d'une évolution graduelle de la QdS. En effet, lorsque la détérioration du contexte provoque une dégradation de la QdS, l'heuristique permet de converger vers une meilleure QdS.

3.4. Conclusion

La conception d'une plate-forme permettant d'optimiser la QdS pour une application répartie à base de composants se heurte au problème de la complexité algorithmique d'une telle tâche. Nous avons montré que celle-ci est NP-complète. Nous avons donc proposé une heuristique permettant de définir dynamiquement les politiques de reconfiguration. Le choix de la future configuration repose ainsi sur une approche itérative guidée par la perception du contexte et le souci de maintenir la plasticité de l'application. Les configurations sont alors regroupées en fonction de la proximité de leur service avec le service fourni à un moment donné.

La plate-forme débute sa recherche d'une solution par l'étude des configurations les plus proches du service en cours avant d'évaluer la qualité potentielle des configurations plus éloignées. Cependant, lorsqu'elle ne dispose pas d'informations permettant de comparer les configurations potentielles avec celle en cours d'exécution, elle utilise une configuration par défaut définie par le concepteur ou la configuration la moins coûteuse en ressources systèmes. Cette heuristique a permis de réduire la complexité du problème à celle dépendant des caractéristiques intrinsèques de l'application combinée à l'étendue de l'offre de service choisie par le concepteur.

Après avoir défini le fonctionnement de la plate-forme, nous avons proposé une structure d'implantation permettant une gestion répartie des événements et des évaluations. La gestion des événements est ainsi guidée par leur importance pour l'utilisateur. Sur chaque poste utilisé par l'application, la plate-forme est constituée de cinq gestionnaires réalisant le choix de l'événement à traiter, l'évaluation des configurations, la supervision de la plate-forme, la saisie des vœux de l'utilisateur et enfin assurant la communication entre les différentes parties de la plate-forme.

Nous avons ensuite validé ce modèle grâce à un prototype qui nous a permis de constater la convergence de l'heuristique vers une configuration stable. L'évolution de la QdS obtenue est graduelle et respecte la plasticité nécessaire aux applications orientées vers l'utilisateur. Nous obtenons alors un système informatique qui, s'il subit des dégradations de qualité lorsque le contexte se détériore, réagit en améliorant cette qualité grâce à une adaptation dynamique.

Conclusion et Perspectives

Posséder les connaissances et les moyens techniques pour fournir un service informatique ne suffit pas : si la qualité obtenue par l'utilisateur est médiocre, ce service restera inutilisable. Or cette qualité ne dépend pas uniquement des talents du concepteur, mais également du contexte d'exécution de l'application, en particulier lorsque celui-ci est variable et à plus forte raison s'il varie de façon imprévisible. Il existe alors deux moyens d'assurer une certaine qualité de service à l'utilisateur : adapter le contexte à l'application, ou du moins contraindre le contexte en fonction de l'application, et adapter l'application au contexte, y compris à l'utilisateur. Or il n'est pas toujours possible d'agir sur le contexte surtout lorsque l'on y inclut, comme nous le faisons, les utiliseurs eux-mêmes. En outre, les méthodes de réservation et de négociation de ressources, qui traitent uniquement du contexte purement informatique, ne sont pas applicables dans des réseaux de type " meilleur effort" tel l'Internet grand public. En revanche, si dès sa conception et sa réalisation, l'adaptation de l'application au contexte a été prise en compte, l'utilisateur pourra bénéficier de la meilleure qualité de service possible.

Ce constat nous a amenés à proposer dans cette thèse une méthode de conception d'application et un support d'exécution qui permettent d'adapter dynamiquement une application aux variations du contexte de manière à fournir et à maintenir la meilleure qualité de service possible aux utilisateurs, qualité définie individuellement.

Pour nos travaux, nous avons choisi comme domaine d'étude celui qui nous semble le plus représentatif de cette problématique : les applications multimédias réparties sur Internet. En effet, elles sont caractérisées à la fois par de grandes exigences de qualité, par une grande sensibilité au contexte et par la forte variabilité de ce dernier. Elles utilisent un grand volume de données soumis à des contraintes temporelles aussi bien de délai que de synchronisation — vidéosurveillance, médecine augmentée, vidéoconférence. Ces exigences se concilient mal avec l'Internet grand public car ce réseau a des performances très variables et ne propose pas de véritable réservation de ressources. Or, pour assurer une qualité de service suffisante aux applications multimédias dans un contexte variable, les solutions habituellement proposées utilisent soit l'allocation de ressources soit une forme d'adaptation dynamique au contexte qui peut concerner le réseau, l'intergiciel, les flux multimédias, l'application, les composants de l'application ou plusieurs de ces éléments à la fois.

D'autre part, les intergiciels constituent un outil efficace pour maintenir une certaine qualité de service. Ainsi CAliF Multimédia propose un intergiciel pour applications coopératives multimédia en utilisant l'allocation de ressources du réseau et l'adaptation des besoins de l'application aux ressources disponibles. L'intergiciel Argilos permet un contrôle hiérarchique de la qualité de service en utilisant non seulement la réservation de ressources mais aussi la configuration de composants. JQoS propose une application de vidéoconférence sur Internet avec adaptation des flux multimédias selon l'état des performances de qualité de service du système. Enfin, QuO permet la spécification et la gestion de la qualité de service dans des applications construites à base de composants répartis : la plate-forme et l'application sont adaptées.

Pour toutes ces raisons, nous avons proposé un intergiciel permettant d'adapter dynamiquement les applications multimédias réparties à leur contexte d'exécution afin de maintenir une qualité de service optimale pour les utilisateurs. Cet intergiciel, lui-même réparti, ajoute ou supprime des composants et reconfigure ou restructure les assemblages de composants formant l'application.

Nos travaux ont été guidés tout du long par l'idée qu'il était nécessaire que notre démarche soit en permanence centrée sur la perception que les utilisateurs auront de l'application. Ainsi notre modèle d'application est structuré autour des notions de service et de fonctionnalité. Le service représente ce qui est offert à l'utilisateur tandis que la fonctionnalité représente tout élément constitutif de ce service qui peut être évalué unitairement par l'utilisateur. De la même façon, notre modèle de qualité de service se structure autour de critères perceptibles et évaluables par l'utilisateur et mesurables par la plate-forme.

C'est pourquoi nous avons défini un modèle d'application à base de composants logiciels qui peut être utilisé aussi bien pour décrire les aspects fonctionnels et structurels que pour évaluer la qualité de service. Pour atteindre ce but, notre modèle d'architecture d'application utilise deux niveaux structurels correspondant aux points de vue de l'utilisateur : le "Groupe" qui représente le service global fourni à l'utilisateur et le "Sous-Groupe" qui exprime les fonctionnalités constituant ce service. Les Groupes fournissent une vision de haut niveau qui permet de faire abstraction des contraintes matérielles et logicielles pour se concentrer sur la réalisation d'une application viable et utilisable. De la même façon, les Sous-Groupes permettent de traiter l'adaptation au contexte en s'affranchissant des contraintes de localisation. Ce modèle issu de l'analyse descendante permet donc de traiter en amont la gestion de la qualité de service puis de s'intéresser aux fonctionnalités de bas niveau avant de procéder à l'implantation. C'est pourquoi notre conception est centrée sur l'utilisateur et s'appuie sur les principes de la programmation par aspects.

En parallèle avec ce modèle d'application et en cohérence avec lui, nous avons défini une manière originale de prendre en compte la satisfaction de l'utilisateur puisque la qualité de service s'entend ici comme l'adéquation entre le service fourni et celui désiré. Le modèle de qualité de service que nous avons proposé est alors basé sur deux critères, l'un appelé "contextuel " regroupant les caractéristiques variant en fonction du contexte et l'autre appelé "intrinsèque" regroupant les caractéristiques ne dépendant pas du contexte. Ces deux critères sont évalués en fonction des désirs de l'utilisateur. En nous inspirant de la notion d'utilité en microéconomie, nous avons choisi d'évaluer la qualité de service par une note définie comme la pire note des critères intrinsèque et contextuel de l'application.

De plus, que ce soit pour l'évaluation de la qualité de service ou pour la conception de l'application, nous avons utilisé un modèle et une représentation à base de graphes de flux orientés. En effet, l'une des principales caractéristiques des applications multimédias est

l'existence de contraintes temporelles dans et entre les flux. Or la qualité de service doit être évaluée en permanence et en temps réel, il est donc nécessaire de trouver un moyen d'évaluation efficace et rapide. Nous avons proposé une évaluation répartie qui utilise les principes des machines à flots de données afin d'optimiser le parallélisme et, dans ce domaine, la représentation par graphes de flux est la plus efficace. Nous avons alors une application orientée flux de données par sa nature multimédia et un support de gestion de cette application orienté flux de données pour accroître sa réactivité.

Ces différents modèles nous ont ensuite permis de proposer une méthode de conception d'application multimédia intégrant la qualité de service et un modèle de plate-forme qui supervise l'application lors de son exécution.

Cependant, nos travaux ont montré que la recherche d'une configuration optimale du point de vue de la qualité de service revient à une combinaison de problèmes d'ordonnancement et de routage connus pour être NP-complets. Nous avons donc proposé une méthode itérative de recherche qui converge vers la meilleure configuration dans un contexte donné. Cette recherche est guidée à chaque étape par le souci de perturber le moins possible l'utilisateur et d'assurer ainsi la plasticité de l'application. De plus, la combinatoire est réduite par la prise en compte des vœux des utilisateurs qui permettent de classer les différentes structures en grandes familles de service et par l'utilisation des informations contextuelles disponibles qui permettent de cibler la recherche. Nous avons ainsi proposé une heuristique adaptée aux contraintes du multimédia. Sa complexité ne dépend plus alors que de la complexité intrinsèque de l'application et de l'étendue de l'offre de qualité de service que le concepteur souhaite proposer : elle est donc incompressible. La solution que nous proposons à ce problème NP-complet est centrée sur l'utilisateur ce que ne pourrait réaliser une heuristique définie uniquement d'un point de vue mathématique.

Ces modèles de plate-forme, d'application et de qualité de service, ainsi que les méthodes associées, graphes de flux et heuristique, ont été validés par un prototype développé avec LabVIEW. Il permet de souligner la dynamicité de l'adaptation aussi bien dans sa partie spécification — puisque les politiques d'adaptation sont définies en cours d'exécution — que dans sa réalisation. Cette dynamicité complète est intéressante car elle n'est pas courante dans les systèmes habituellement proposés lorsque la QdS tient compte de l'utilisateur en absence de réservation de ressource. Or elle est essentielle dans des environnements imprédictibles d'autant plus qu'elle autorise la personnalisation des services. L'adaptation obtenue alors se veut totale puisqu'elle concerne la structure, les fonctionnalités et l'ordonnancement des applications regroupant ainsi des aspects étudiés fréquemment séparément. L'offre de QdS est ainsi étendue au maximum en fonction des ressources matérielles et logicielles disponibles. Enfin, notre système est caractérisé par la possibilité d'obtenir une amélioration comme une dégradation de qualité du service en fonction du contexte là où une majorité de travaux proposent uniquement une dégradation. Nous avons ainsi essayé de proposer une adaptation et une dynamicité les plus abouties possibles dans le cadre de notre problématique.

L'objectif de la plate-forme demeure à tout instant de fournir le meilleur compromis entre les caractéristiques de qualité dépendant du contexte et celles qui en sont indépendantes. Sa manière d'y parvenir est fondée sur une reconfiguration dynamique de l'application aux différents niveaux structurels proposés par le modèle d'application. Nous pouvons noter que ce type de reconfiguration est actuellement utilisé à diverses granularités pour la réalisation de systèmes contraints temporellement ou temps réel. Au-delà des niveaux logiciel et matériel entre postes informatiques comme nous le proposons, cette reconfiguration peut consister à répartir différemment l'application entre des composants

électroniques non reconfigurables de type ASIC — *Application Specific Integrated Circuit* — ou D.S.P. — *Digital Signal Processor* — au sein d'un même poste informatique. Elle peut également être réalisée à une granularité plus fine au sein de composants aussi bien en réorganisant les portes logiques que les fonctionnalités de ces puces électroniques comme dans les A.R.D. — Architecture Reconfigurable Dynamiquement — et les F.P.G.A. — *Field Programmable Gate Array.* Dans un second temps, nous pourrions donc envisager d'étendre notre méthode d'adaptation aux domaines utilisant ces différentes reconfigurations dans la mesure où il nous sera possible de définir une orientation jouant un rôle semblable à la perception de l'utilisateur pour nos travaux. Il sera alors nécessaire de proposer un modèle architectural reflétant cette orientation et adapté à chaque cas de manière à identifier les différents niveaux reconfigurables.

Le succès récent des plates-formes associées aux composants — Jonas, Sun Java Application Server, Jboss, .NET etc. — montre qu'il devient nécessaire d'externaliser les propriétés non fonctionnelles ou liées à l'environnement dès lors que l'on s'intéresse à des applications réparties et complexes. La plate-forme présentée dans cette thèse n'a pas la prétention d'universalité de celles citées précédemment. Bien au contraire, elle vise la résolution d'un problème bien ciblé. Or, on a vu au cours du temps les applications en réseau — couche 7 du modèle ISO — intégrer de plus en plus les aspects de présentation — couche 6 du modèle ISO — à tel point qu'elles en sont indissociables. De la même façon, il semble acquis que les applications réparties se baseront de plus en plus souvent sur un intergiciel ou une plate-forme de déploiement intégrant la supervision ou la prise en compte des aspects de sécurité, de persistance etc. Pour toutes ces raisons, nous pensons que plate-forme et application doivent être intimement liées depuis la phase de la conception jusqu'à celle de l'exploitation afin de ne laisser à l'application que les aspects métier et de permettre ainsi la réutilisation de composants ou l'emploi de composants sur étagère — COTS. C'est pourquoi nous souhaitons réaliser non plus uniquement une plate-forme d'exécution mais également une partie d'aide à la conception. Pour cela, il sera vraisemblablement nécessaire d'étudier une taxonomie des rôles des composants de manière à automatiser la méthode de conception et l'intégration de composants en cours d'exécution. L'une des principales difficultés résidera alors dans la définition automatique des couples de flux à synchroniser à partir des fonctionnalités à synchroniser précisées par le concepteur.

D'autre part, lorsque les travaux de notre équipe concernant la modélisation des composants logiciels et des flux pour le multimédia auront abouti [BOX 05], nous pourrons les utiliser pour implanter notre modèle de plate-forme. Nous pourrons alors proposer une gestion de la synchronisation intégrée à la qualité de service alors que nos modèles actuels la posent en condition *sine qua non* à toute transmission. La plate-forme disposera ainsi d'un degré de liberté supplémentaire pour optimiser la qualité globale du service fourni. Les critères d'évaluation de la QdS devront donc tenir compte de la synchronisation et des mécanismes de décision la concernant devront est définis.

Nous envisageons également que la plate-forme puisse utiliser la prédictibilité de la défaillance de certains composants comme ceux alimentés par une batterie afin d'anticiper les dégradations de QdS en utilisant par exemple des outils probabilistes. Cela lui permettrait non plus de proposer la meilleure QdS possible mais un compromis entre la meilleure QdS et la stabilité de la qualité obtenue augmentant ainsi le confort de l'utilisateur. Pour cela, il est nécessaire de disposer de nouveaux outils — probablement des événements —générant les reconfigurations ainsi que de mécanismes gérant la disparition de composants sans que la plate-forme locale ne l'ait décidé.

Nous pourrons alors proposer que la plate-forme locale gère également l'apparition de nouveaux composants de manière à introduire dans l'application des composants matériels mobiles connectés par des réseaux sans fils sur lesquels on ne peut raisonnablement pas installer une plate-forme locale. Ils seraient rattachés par des règles à définir au site contenant une partie de plate-forme le plus accessible depuis leur localisation. Contrairement au modèle actuel où chaque site physique est accessible par la plate-forme puisqu'elle y est présente, ces composants seraient donc en dehors des sites directement accessibles par la plate-forme. S'ils sont mobiles, ils ne seront pas toujours visibles par la même portion de plate-forme et ne seront donc pas gérés par le même site. Ainsi les travaux de cette thèse s'inscriront naturellement dans l'un des nouveaux axes de recherche de notre laboratoire qui s'intéresse à la qualité de service dans le cadre des réseaux de capteurs mobiles.

Enfin, l'informatique ubiquitaire impose l'utilisation de réseaux hétérogènes. En effet, lors d'une même communication, les informations peuvent transiter pour partie sur des réseaux dénués de garantie de service et pour partie sur d'autres proposant des mécanismes de garantie de service ou de réservation de ressources. Il sera alors intéressant d'étudier la possibilité d'inclure notre plate-forme dans un système plus ambitieux permettant d'utiliser l'adaptation telle que nous la proposons dans tous les cas de figures mais également d'y adjoindre l'exploitation des garanties de service lorsque cela est possible. Nous pourrons alors utiliser une adaptation du réseau concernant les serveurs et utilisant des nœuds actifs. Ainsi le service sera optimisé du point de vue de l'utilisateur, puisque c'est là notre objectif, mais aussi du point de vue du fournisseur, préoccupation à l'origine de la notion de qualité de service.

Bibliographie

[AFN 04] 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.

[AMA 95] AMAMIYA M., KAWANO T., « Design Principle of Massively Parallel Distributed-Memory Multiprocessor Architecture », paru dans GAO G.R., Bic L., Gaudiot J.-L., *Advanced Topics in Dataflow Computing and Multithreading*, IEEE Computer Society Press, Los Alamitos, California, 1995.

[AND 98] ANDREW L. L. H., KUSUMA A. A. N. A., « Generalised Analysis of a QoS-Aware Routing Algorithm », IEEE GLOBECOM'98, 1998, p. 118-123.

[AUR 98] AURRECOECHEA C., CAMPBELL A.T., HAUW L., « A survey of QoS architectures » in *Multimedia Systems*, chapitre 6, pages 138-151. Springer-Verlag 1998.

[ARC 00] Architecture Répartie extensible pour Composants Adaptables. Programme de recherche, 2000, http://www.essi.fr/~riveil/recherche/00-index.html.

[ASP 03] Proposition d'Action Spécifique, « Systèmes répartis et réseaux adaptatifs au contexte », CNRS-STIC et GET, juin 2003, http://www.infres.enst.fr/%7Edemeure/AS150/TexteInitial.htm.

[ATA 03] BEN ATALLAH S., LAYAIDA O., DE PALMA N., HAGIMONT D., « Dynamic Configuration of Multimedia Applications », In *Proceedings of the IFIP/IEEE Conference of Multimedia Networks and Services, MMNS 2003*. Published in Lecture Notes for Computer Science (LNCS 2839), Belfast, 2003, pp 46-63.

[AYE 04] AYED D., TACONET C., BERNARD G., « Architecture à base de composants pour le déploiement adaptatif des applications multicomposants », Journées Composants 2004, Lille, France, Mars 2004.

[AYE 05] AYED D., TACONET C., SABRI N., BERNARD G., « CADeComp : Plate-forme de déploiement sensible au contexte des applications à base de composants », 4ème Conférence Française sur les Systèmes d'Exploitation CFSE'4 - Le Croisic France - 6-8 Avril 2005.

[BAS 93] BASTIEN, SCAPIN « Critères ergonomiques pour l'Évaluation d'Interfaces Utilisateurs », INRIA 1993.

[BER 80] BERGER P., COMTE D., HIDI N., PELOIS B., SYRE J.C., « Le système LAU : un multiprocesseur à assignation unique » ONERA-CERT, 1980.

[BOO 00] BOOCH G., RUMBAUGH J., JACOBSON I., *Le guide de l'utilisateur UML,* Éditions Eyrolles, Paris, 2000.

[BOU 01] N.M.N. BOURAQADI-SAÂDANI, LEDOUX T., Le point sur la programmation par aspects, Techniques et science informatiques. Volume 20-n°4/2001, pages 505 à 528.

[BOX 05] BOUIX E., DALMAU M., ROOSE P., LUTHON F., « A Component Model for transmission and processing of Synchronized Multimedia Data Flows », DFMA 05 - The First International Conference on Distributed Frameworks for Multimedia Applications - IEEE Computer Society Press - ISBN: 0-7695-2273-4, pp. 45-53, Besançon, France - February 6-9, 2005.

[CCI 89] 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.

[CHR 89] CHRETIENNE P. « Task scheduling over distributed memory machines ». In *Proc. Int'l Conf. Distributed Algorithms*, Amsterdam, Nov 1989, North Holland Publishers.

[COP 01] 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.

[COR 99] CORTES L.A., ELES P., PENG Z., *A survey on Hardware/Software Codesign Representation Models*, SAVE Project Report, Department of Computer and Information Science, Linköping University, Linköping, Sweden, June 1999.

[COU 01] COUTAZ J., NIGAY L., « Architecture logicielle conceptuelle des systèmes interactifs », *Analyse et conception de l'IHM,* Hermès 2001 Chapitre 7.

[CUI 01] CUI Y., NAHRSTEDT K., « QoS-Aware Dependency Management for Component-Based Systems », in *Proc. of International Symposium on High Performance Distributed Computing* (HPDC '01).

[CUI 01b] CUI Y., XU D., NAHRSTEDT K., « SMART: A Scalable Middleware Solution for Ubiquitous Multimedia Service Delivery », in *Proc. of IEEE International Conference on Multimedia and Expo 2001 (ICME2001)*, Tokyo, Japan, August, 2001.

[DAL 05] DALMAU M., ROOSE P., BOUIX E., LUTHON F., « A Multimedia Oriented Component Model », AINA 2005, The IEEE 19th International Conference on Advanced Information Networking and Applications, Tamkang University, Taiwan - March 2005.

[DAS 00] DASILVA L.A., « Pricing for QoS-enabled Networks: A Survey, » IEEE Communication Surveys and Tutorials, vol. 3, no. 2, Second Quarter 2000, pp. 2-8.

[DAY 88] DAYAL U., BLAUSTEIN B., BUCHMANN A., « The HiPAC Project : Combining Active Databases and Timing Constraints », SIGMOD RECORD, vol. 17, n°1, p. 51-70, March 1988.

[DEM 99] DEMEURE I., LEBOUCHER L, RIVIERRE N., SINGHOFF F. « Modèle et plate-forme pour le support d'applications multimédias réparties » CFSE'1 Première Conférence Française sur les Systèmes d'Exploitation, Rennes, 1999.

[DEM 02] 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.

[DEN 75] DENNIS J.B., MISUNAs D.P., « A Preliminary Architecture for a Basic Data-Flow Processor », *Proc. Of the 2nd Annual Symposium on Computer Architecture*, January 1975, Houston, Texas, pp126-132.

[DUM 03] DUMOULIN C., BOULET P., DEKEYSER J.-L., MARQUET P., *UML 2.0 Structure Diagram for Intensive Signal Processing Application Specification*, Rapport de recherche n°4766, 11 mars 2003, ISSN 0249-6399.

[ELE 98] 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.

[ELE 00] 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.

[FER 89] FERNANDEZ-BACE. « Allocating modules to processors in a distributed system ». *IEEE Transactions on Software Engineering*, Vol. 15, N°11 :1427-1436, nov 1989.

[FIR 03] 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.

[FRA 91] 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.

[GAC 01] GARCIA É., LAPAYRE J.-C., RENARD F., BA T., « CAliF multimédia : une plate-forme à objets pour le développement de télé-applications multimédia » in Réseaux et Systèmes Répartis Calculateurs Parallèles, numéro spécial Télé-Applications, Hermès Science, 2001, vol.13, n°2, mars 2001, p 295 –318.

[GAC 01b] GARCIA É., Une plate-forme de développement pour applications coopératives multimédia intégrant la gestion de la qualité de service, Thèse de l'Université de Franche-Comté novembre 2001.

[GAL 99] GALILEE F., Athapascan-1 : interprétation distribuée du flot de données d'un programme parallèle, Thèse de doctorat de l'Institut National Polytechnique de Grenoble, septembre 1999.

[GAR 79] GAREY M. R., JOHNSON, D. S., *Computers and Interactability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[GUX 00] GU X., XU D., WICHADAKUL D., NAHRSTEDT K., « QoSTalk: A Visual Quality of Service Programming Environment », Illinois Computer Affiliates Program (ICAP) workshop, September 2000.

[GUX 00b] GU X., NAHRSTEDT K., « Visual Quality of Service Programming for Distributed Heterogeneous Systems », Technical Report UIUCDCS-R-2000-2190 UILU-ENG-2000-1746, Department of Computer Science, University of Illinois at Urbana-Champaign, Nov.2000.

[GUX 01] GU X., WICHADAKUL D., NAHRSTEDT K., « Visual QoS Programming Environment for Ubiquitous Multimedia Services », In *Proceedings of IEEE International Conference on Multimedia and Expo2001* (ICME2001), Tokyo, Japan, Aug. 22 - Aug. 25, 2001.

[GUX 01b] GU X., NAHRSTEDT K., « An Event-Driven, User-Centric, QoS-aware Middleware Framework for Ubiquitous Multimedia Applications », *In Proceedings of ACM Multimedia 2001* (Multimedia Middleware Workshop), Ottawa, Canada, October 2001.

[GUX 02] GU X., NAHRSTEDT K., YUAN W., WICHADAKUL D., XU D., « An XML-based Quality of Service Enabling Language for the Web », Journal of Visual Language and Computing (JVLC), special issue on Multimedia Languages for the Web, vol. 13, num. 1, pp. 61-95, Academic Press, 2002.

[HAF 98] 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.

[HAG 02] HAGIMONT D., LAYAIDA N., « Adaptation d'une application multimédia par un code mobile », *Technique et Science Informatique* (TSI), numéro spécial "Agents et code mobile", Volume 21, numéro 6/2002.

[HUL 97] HULL D.,SHANKAR A., NAHRSTEDT K., LIU J. W.-S., « An End-to-End QoS Model and Management Architecture » in *Proceedings of IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, p.82-89, December 1997.

[ISO 91] ISO/IEC JTC1/SC29/WG12, MHEG Working Group S.5, Information Technology, Coded Representation of Multimedia and Hypermedia Information Objects, juillet 1999, http://www.itscj.ipsj.or.jp/sc29/open/29view/29n3190c.htm.

[ISO 95] ISO, QoS Framework, ISO/IEC/JTC1/SC21/WG1 N9680, 1995.

[ISO 97] ISO/IEC, Information Technology – Quality of Service – Framework, ISO/IEC JTC1/SC21 N13236, Geneva, 1997.

[KUI 02] KUIPERS F., VAN MIEGHEM P., 2002, « MAMCRA: a constrained-based multicast routing algorithm », Computer Communications, vol. 25, pp.802-811.

[LAG 90] LAGEWEG B.J., VELTMAN B., LENSTRA J.K., « Multiprocessor scheduling with communication delays ». *Parallel Computing*, 16 : 173-182, 1990.

[LAP 02] LAPLACE S., Architecture logicielle intégrant la qualité de service pour les applications multimédias réparties reconfigurables, Mémoire de D.E.A., Université du Maine, septembre 2002.

[LAR 96] Le petit Larousse Illustré 1996 Dictionnaire encyclopédique. Editions Larousse. Paris 1995.

[LAY 04] 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

[LED 01] LEDOUX T. et al., Etat de l'art sur l'adaptabilité. Délivrable D1.1, projet R.N.T.L. ARCAD, 12 décembre 2001.

[LEE 94] LEE B., HURSON A.R., « Dataflow Architectures and Multithreading » Vol. 27, pp 27-39, August 1994 Computer, IEEE.

[LIB 99] LI B., NAHRSTEDT K., « A Control-based Middleware Framework for Quality of Service Adaptations » in IEEE Journal on Selected Areas in Communications, Special Issue on Service Enabling Platforms, Vol. 17, No. 9, pp. 1632-1650, September 1999.

[LIB 01] 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.

[LIB 02] LI B., XU D., NAHRSTEDT K., «An integrated runtime QoS-aware middleware framework for distributed multimedia applications *» Multimedia Systems,* 8, p. 420-430, Springer-Verlag 2002.

[LOY 98] LOYALL J.P., SCHANTZ R.E., ZINKY J.A., BAKKEN D.E., « Specifying and measuring quality of service in distributed object system », *Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing* (ISORC '98), 20-22 April 1998, Kyoto, Japan.

[LUT 99] LUTHON F., CAPLIER A., LIÉVIN M., « Spatiotemporal MRF approach to video segmentation: Application to motion detection and lip segmentation. », Signal Processing, 76(1):61-80, July 1999.

[MAR 99] Marangozova V., Qualité de service et adaptabilité pour des applications réparties, Additional Master's report, 1999.

[MIC 05] MICHEL R., ROOSE P., BARBIER F., « 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 - August 2005.

[MOR 02] MORENO G.A., HISSAM S.A., WALLNAU K.C., « Statistical Models for Empirical Component Properties and Assembly-Level Property Predictions:

Toward Standard Labeling » *Proceedings of the 5th ICSE Workshop on Component-Based Software Engineering*, Orlando, Florida, May 2002.

[OFF 96] OFFICE QUÉBÉCOIS DE LA LANGUE FRANCAISE, Grand dictionnaire terminologique. Disponible sur : http://www.olf.gouv.qc.ca/ressources/gdt.html

[PAL 00] PAL P.P., LOYALL J.P., SCHANTZ R.E., ZINKY J.A., SHAPIRO R., MEGQUIER J., « Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration ». *Proceedings of ISORC 2000*, The Third IEEE International Symposium on Object-Oriented Real-time Distributed Computing, March 15-17, 2000, Newport Beach, CA.

[PAR 92] PARKIN M., FLUET C-D, BADE R*., Introduction à la microéconomie moderne*, Editions ERPI, 1992, ISBN 2-7613-0673-2.

[PLA 76] PLAS, A., COMTE D., GELLY O., SYRE J.C., « LAU System Architecture: A Parallel Data-Driven Processor Based on Single Assignment », in *Proc. Of the Int. Conf. Parallel Processing*, IEEE, New York, 1976, pp. 293-302.

[RAK 01] RAKOTONIRAINY A., INDULSKA J., LOKE S., ZASLAVSKY A., « Middleware for Reactive Components : An Integrated Use of Context, Roles, and Event Based Coordination », IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg, Allemagne, Novembre 2001.

[RAY 87] RAYWARD-SMITH V., « UET scheduling with unit interprocessor communication delays », *Discrete Applied Mathematics*, vol. 18, 1987, p. 55-71.

[SAL 00] SALBER D., « Context-awareness and multimodality », Colloque sur la multimodalité, mai 2000, I.M.A.G., Grenoble.

[SCA 02] SCHANTZ R., LOYALL J., ATIGHETCHI M., PAL P., « Packaging Quality of Service Control Behaviors for Reuse », in *Proceedings of the 5th IEEE International Symposium on Object-oriented Real-Time Distributed Computing* (ISORC), IEEE/IFIP, Crystal City, VA, 2002.

[SCA 03] SCHANTZ R.E., LOYALL J.P., RODRIGUES C., SCHMIDT D.C., KRISHNAMURTHY Y., « Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware ». ACM/IFIP/USENIX International Middleware Conference, Rio de Janeiro, Brazil, June 2003.

[SIN 98] SINGHOFF F., DEMEURE I. « Spécification et ordonnancement dynamique d'applications multimédias: l'environnement POLKA » Real Times Systems RTS'98 Paris La Défense.

[SPY 02] SZYPERSKI C., *Component software : beyond object-oriented programming*, second edition. Addison-Weslye/Acm Press, 2002.

[STE 02] STEPHEN S., YU W., FARIAZ K., « Development of Situation-Aware Application Software for Ubiquitous Computing Environments », COMPSAC'02, Angleterre, 2002.

[STE 93] STEFANI J.-B., « Computational Aspects of QoS in an object-based, distributed systems architecture ».3rd Workshop on Responsive Computer Systels, Lincoln, NH, U.S.A., September 1993.

[SUN 99] SUN, Java Media Framework API Guide Version 2.0», Sun Microsystems novembre 1999.

[SYR 77] SYRE, J. C., COMTE D., HIFDI N., « Pipelining, parallelism and asynchronism in the LAU system », in *Proceedings of the 1977 International Conference on Parallel Processing* (Detroit, Mich., Aug. 23-26), J. L. Baer, Ed. IEEE, New York, pp. 87-92.

[VER 00] VERIANS X., Extraction du parallélisme entre flots d'instructions à l'aide de techniques superscalaires pour des architectures multiprocesseurs ou multiflots, Thèse de doctorat en Sciences Appliquées de l'Université Catholique de Louvain, mai 2000.

[VOG 95] VOGEL A., KERHERVE B., BOCHMANN G., GECEI., « Distribued multimedia applications and Quality of Service :- A Survey-» IEEE Mutimedia Journal, august 1995.

[WAG 04] WANG N., GILL C., SCHMIDT D., GOKHALE A., NATARAJAN B., LOYALL J., SCHANTZ R., RODRIGUES C., QoS-enabled Middleware. Chapter in *Middleware for Communications*, Qusay H. Mahmoud (Editor), Wiley, July 2004.

[WAN 96] 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.

[WIC 01] WICHADAKUL D., NAHRSTEDT K., GU X., XU D., « 2KQ+ : An integrated approach of QoS compilation and component-based, run-time middleware for the unified QoS management framework », in *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*. Heidelberg, Germany, November 2001.

[WOL 91] WOLPER P., *Introduction à la calculabilité*, 1991, InterEditions, Paris, p 237, ISBN 2-7296-0372-7.

[ZHU 01] ZHU W., GEORGANAS N. « JQoS : Design and Implementation of a QoS-based Internet Videoconferencing System using Java Media Framework (JMF) » CCECE'2001, Canadian Conference on Electrical and Computer Engineering, Toronto, Canada.

[ZIN 97] ZINKY J.A., BAKKEN D.E., SCHANTZ R.D., « Architectural Support for Quality of Service for CORBA Objects », *Theory and practice of Object Systems,* vol. 3 (1), John Wiley & Sons, Inc.,1997.

Publications

1. Philippe Roose, Marc Dalmau, Franck Luthon, Sophie Laplace – *Gestion de la qualité de service par reconfiguration dynamique dans les applications interactives multimédia* - Journées Systèmes à Composants Adaptables et Extensibles (JC2002) - Grenoble, 17-18 Oct. 2002.
2. Sophie Laplace, Marc Dalmau, Philippe Roose - *New formal approach for QoS management in Distributed Multimedia Applications* - Poster - 6th IFIP/IEEE International Conference on Management of Multimedia Networks and Services - 7th - 10th September 2003, Queen's University of Belfast, Northern Ireland.
3. Sophie Laplace, Marc Dalmau, Philippe Roose - *A formal method for assessing quality of service in distributed multimedia applications* - ICSSEA 2003, 16th International Conference on "Software & Systems Engineering and their Applications - December 2-4 2003 - CNAM — Paris, France.
4. Marc Dalmau, Sophie Laplace, Philippe Roose - *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.
5. Marc Dalmau, Sophie Laplace, Philippe Roose - *Gestion de la QdS des applications réparties par adaptation dynamique au contexte* - Journée de l'action spécifique CNRS-GET - Systèmes répartis et réseaux adaptatifs au contexte ("Context-Aware") - AS 150 (RTP 1 Réseaux et RTP 5 Systèmes Répartis)- 1er avril 2004 – Paris.
6. Sophie Laplace, Marc Dalmau, Philippe Roose - *QoS Oriented Design and Management for Multimedia Distributed Applications* - 6ème colloque francophone GRES 2005 - Luchon, France - 28 Février-3 mars 2005.
7. Sophie Laplace, Philippe Roose, Marc Dalmau - *Gestion de la qualité de service de la conception à l'exécution dans les applications distribuées multimédias* - Journées Composants 2005 - 4ème Conférence Francophone autour des Composants Logiciels - Le Croisic, France - 6-8 avril 2005.
8. Sophie Laplace, Marc Dalmau, Philippe Roose – *Prise en compte de la qualité de service dans la conception et l'exploitation d'applications multimédia réparties* – Congrès INFORSID 2006 (à paraître) – Hammamet (Tunisie) – 1-3 juin 2006.

« Calme-toi. C'est bien. Tout va bien, dit la guérisseuse. Tu te rappelles chaque pas que tu as fait pour escalader la montagne. Tu te rappelles chaque pas que tu as fait pour gagner la confiance de l'ours. Tu te rappelles tout ce que tu as vu, tout ce que tu as entendu, tout ce que tu as ressenti ?

– Oui, dit la jeune femme, je me le rappelle parfaitement. »

La vieille guérisseuse lui adressa un doux sourire.

« Eh bien maintenant, ma fille, dit-elle, rentre chez toi avec cette compréhension nouvelle… »

L'ours au croissant de lune, conte extrait de « Femmes qui courent avec les loups » sous-titré « Histoires et mythes de l'archétype de la femme sauvage » par Clarissa Pinkola Estés (Grasset 1996 pour la version française)