Modélisation et support d'applications multimédias réparties

Isabelle Demeure * — **Laurent Leboucher** **
Nicolas Rivierre ** — **Frank Singhoff** *

* *Ecole Nationale Supérieure des Télécommunications*
CNRS URA 820
46, rue Barrault - 75634 Paris Cedex 13, France
** *Centre National d'Etude des Télécommunications*
38,40 rue du Général Leclerc - 92131 Issy-Les-Moulineaux, France

RÉSUMÉ. *Cet article décrit une technique de spécification d'applications multimédias réparties et une plate-forme d'exécution pour ces applications. Les contraintes temporelles de l'application sont spécifiées indépendamment de l'application et du graphe de flots de données qui décrit le système multimédia. La plate-forme d'exécution est basée sur CORBA. Elle ordonnance automatiquement les applications de manière à respecter les contraintes temporelles spécifiées, dans la limite des ressources disponibles. Les concepts introduits sont illustrés par un exemple. Une évaluation de performances est fournie.*

ABSTRACT. *In this paper we describe a specification technique and a middleware to support distributed multimedia applications. Temporal constraints are specified independently from the application itself and from the dataflow graph that describes the multimedia system. The middleware is based on CORBA. It automatically schedules the applications in order to meet the QoS contraints, provided enough resource is available. The notions introduced are illustrated by an example. We also provide performance evaluation.*

MOTS-CLÉS : *Applications multimédias réparties, objets, ordonnancement, contraintes temporelles.*
KEY WORDS : *Distributed multimedias applications, objects, scheduling, temporal constraints.*

1. Introduction

Dans ce travail, nous nous intéressons au support d'applications *multimédias réparties* qui mettent en œuvre des *flux continus* tels que l'audio et la vidéo. Par « flux continu » on entend des flux composés d'éléments de données qui doivent être présentés en respectant des contraintes de *qualité de service (QoS)* temporelles (ex : délai entre l'affichage de deux images successives, synchronisation voix-lèvres, synchronisation de l'affichage d'objets animés).

Pour prendre en compte ces contraintes temporelles les solutions développées à ce jour utilisent, en général, les propriétés temps réel de l'ordonnanceur du système sous-jacent et le support de contraintes de QoS temporelle qu'offrent les nouvelles générations de protocoles de communication (ex : ATM [VET 95], RTP/RTCP [SCH 96]). Elles s'appuient également sur des composants spécifiques tels que les cartes audio et les décodeurs MPEG.

Ces solutions présentent plusieurs inconvénients : tout d'abord, les ordonnanceurs traditionnels tels que celui du système Unix SVR4, par exemple, n'ont pas été conçus pour les processus qui sont chargés de traiter des flux continus [NIE 93]. Un ordonnanceur pour des applications multimédias doit offrir des abstractions proches de celles qui sont utilisées dans les applications multimédias.

Ensuite, on ne peut pas se contenter de considérer chaque composant du système isolément : dans certains cas, il faut prendre en compte le comportement du système dans sa globalité, ou de bout-en-bout [CAM 96].

Par ailleurs, une petite modification dans le comportement temporel de l'application peut nécessiter une grande modification dans la solution développée si les contraintes temporelles sont incluses dans le code de l'application ; idéalement, on devrait pouvoir modifier les contraintes de QoS indépendamment du code de l'application. Ceci est nécessaire si l'on souhaite atteindre un niveau élevé de portabilité de l'application. Notons également que si la prise en compte de contraintes statiques est envisageable, celle de contraintes dynamiques (évoluant au gré de l'exécution de l'application) est souvent complexe, voire impossible.

De plus, le développement de telles applications demande une connaissance très approfondie des systèmes d'exploitation sous-jacents, des protocoles de communication et des composants spécifiques utilisés (ex : carte audio, décodeur MPEG).

Le modèle de spécification et la plate-forme d'exécution POLKA apportent des solutions à ces problèmes. Le modèle de spécification permet, d'une part, de décrire l'application multimédia et le *comportement temporel* qu'elle doit observer ; il permet, d'autre part, de décrire les composants de la plate-forme support et le graphe de flots de données qui représente l'ensemble du système multimédia (applications et composants de la plate-forme, cf. figure 1).

La plate-forme d'exécution POLKA utilise les spécifications pour faire *l'ordonnancement automatique* de l'application de façon à respecter les contraintes

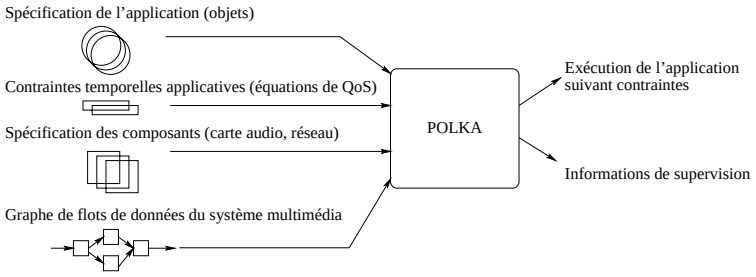

Figure 1. *Modèle et plate-forme POLKA*

temporelles spécifiées, dans la limite des ressources disponibles (ex : processeur, mémoire et bande passante réseau). Dans POLKA, on peut également modifier le comportement temporel de l'application en modifiant uniquement le comportement temporel spécifié. Enfin, POLKA permet de faire varier les contraintes temporelles au cours de l'exécution.

Dans la suite de cet article, nous présentons le modèle de spécification de l'application, des composants et du graphe de flots de données du système multimédia. Nous introduisons, à titre d'illustration, une application répartie qui manipule un flux audio et un flux vidéo qui doivent être synchronisés. La spécification de QoS de cette application est donnée dans les annexes B et C. Dans le chapitre 3, nous décrivons les objectifs et l'architecture de la plate-forme d'exécution. Bien que les techniques de modélisation et d'ordonnancement proposées dans POLKA puissent être utilisées dans tous systèmes à objets, la plate-forme actuelle est basée sur un bus à objets au standard CORBA [OMG 98]. Elle comporte une couche « machine virtuelle » qui a été développée pour augmenter la portabilité de POLKA et de ses applications. Aujourd'hui, POLKA fonctionne sur Linux et Solaris (un portage sur L4 est en cours). La plate-forme actuelle n'offre pas de services de réservation de ressources, néanmoins, nous expliquons dans l'annexe A, comment ajouter un modèle de ressources à POLKA, de façon à fournir des garanties de QoS. Le chapitre 4 est consacré à une évaluation de l'approche. Nous y présentons des mesures de performance réalisées sur la plate-forme POLKA. Ces mesures montrent que l'approche permet effectivement la prise en compte automatique des contraintes temporelles de façon significativement plus efficace qu'une plate-forme CORBA standard utilisant l'ordonnancement par défaut du système d'exploitation sous-jacent. Le surcoût engendré par la plate-forme est par ailleurs raisonnable. Enfin, nous discutons des travaux similaires à POLKA et nous concluons.

2. Modélisation d'un système multimédia

Dans ce chapitre, nous regardons successivement comment modéliser une application multimédia, puis, comment exprimer ses contraintes temporelles, et enfin, comment ces spécifications sont intégrées dans un modèle global du système multimédia.

2.1. *Modélisation d'une application*

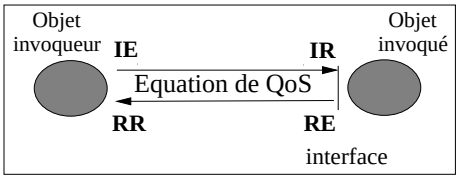

Figure 2. *Points d'observation durant une invocation de méthode*

Dans l'approche POLKA, une application est modélisée par un ensemble d'objets qui coopèrent pour traiter et présenter des flux continus ou *flux multimédias* [STE 96]. Un objet est une unité d'encapsulation de code et de données. Les objets interagissent uniquement par appel de méthodes exportées aux interfaces d'autres objets : ils ne communiquent pas par mémoire partagée. Ces objets sont animés par des *threads*. Comme dans Clouds [DAS 90], une *thread* POLKA est un flot d'exécution traversant éventuellement plusieurs objets et plusieurs machines au gré des invocations réalisées.

Un flux multimédia correspond à une suite d'invocations de méthodes. Une invocation de méthode se décompose en événements (cf. figure 2). Ces événements sont l'émission d'une invocation par l'invoqueur (événement IE), la réception de cette invocation par l'objet invoqué (événement IR), l'émission de la réponse qui fait suite (événement RE), et la réception de la réponse par l'invoqueur (événement RR). Les invocations asynchrones donnent lieu aux seuls événements IE et IR. *Un flux multimédia correspond donc à une suite d'événements.*

Considérons, par exemple, une application qui lit les éléments audio et vidéo d'un film stocké dans un ensemble de fichiers et qui les transmet par un réseau à un site distant où le film est visionné. On utilise la norme de compression MPEG-2 [ISO 95]. Cette application est constituée de trois objets : un objet *serveur* qui lit des trames MPEG sur un disque et les envoie à un objet distant : *sourceMpeg* ; un objet *sourceMpeg* qui consomme les trames qu'il a stockées dans ses tampons et les transmet à l'objet *decodeurMpeg* ; un objet *decodeurMpeg* qui décode des trames MPEG et les présente sur les périphériques de sortie (carte audio et écran géré par un serveur X11).

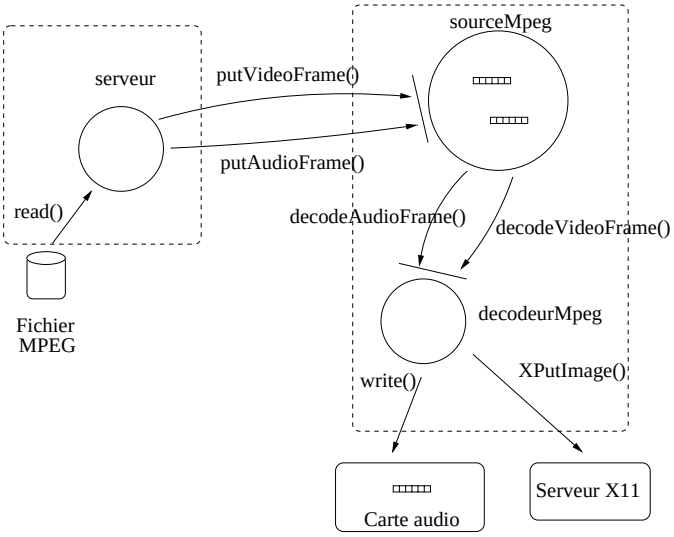

Figure 3. *Une application répartie sur POLKA*

En pratique, les objets applicatifs sont des objets CORBA. La figure 4 décrit l'interface IDL des objets *decodeurMpeg* et *sourceMpeg*. Cet exemple fait intervenir quatre *threads*. Ainsi pour la vidéo (resp. pour l'audio), une *thread* remplit le tampon en bouclant sur l'invocation de la méthode $putVideoFrame$ (resp. $putAudioFrame$) de l'objet *sourceMpeg*. Une deuxième *thread* consomme les trames du tampon et boucle sur l'invocation de la méthode $decodeVideoFrame$ (resp. $decodeAudioFrame$) de l'objet $decodeurMpeg$.

```
interface decodeurMpeg {
    long decodeAudioFrame(in frame f);
    long decodeVideoFrame(in frame f);
};
interface sourceMpeg {
    long putAudioFrame(in frame f);
    long putVideoFrame(in frame f);
};
```

Figure 4. *Interface* IDL *de l'application*

Une fois les objets applicatifs décrits, on peut spécifier les contraintes temporelles que doit respecter l'application. On utilise pour cela des équations de QoS qui expriment des contraintes temporelles entre les événements (IE, IR, RE, RR) observables durant les invocations de méthodes d'un objet. Dans le cas général, contrairement aux objets de l'application, les équations peuvent être

dissimulées à l'utilisateur final par une interface de haut niveau. Par exemple, une application de vidéo à la demande peut proposer à l'utilisateur un ascenseur permettant de choisir le rythme d'affichage des flux vidéo. En faisant glisser l'ascenseur, l'utilisateur déclenche une modification des équations de QoS.

2.2. *Equations de QoS temporelle*

Les équations de QoS sont exprimées dans la logique temporelle QL [STE 93] (*QoS Language*). Dans cet article, nous nous limitons à des contraintes temporelles déterministes.

Considérons, tout d'abord, l'inéquation suivante :

$$\forall\ n\ :\ \epsilon_1 \leq \tau(e, n+k) - \tau(e', n) \leq \epsilon_2 \qquad [1]$$

où e et e' sont deux événements et $\tau(x, n)$, l'opérateur qui fournit la date de l'occurrence n de l'événement x. Cette inéquation stipule que la n ième occurrence de e' doit être séparée d'au moins ϵ_1 et d'au plus ϵ_2 unités de temps de l'occurrence $n+k$ de e.

Cette inéquation permet, par exemple, de décrire un trafic périodique de période p avec une gigue donnée (ainsi, si l'on pose ϵ tel que $\epsilon_1 = p - \epsilon$ et $\epsilon_2 = p + \epsilon$, la gigue maximale entre les occurrences de e et de e' sera de $2 * \epsilon$ unités de temps). Cette inéquation peut aussi modéliser les contraintes inter-flux (ex : synchronisation voix-lèvres). Elle s'utilise également pour borner des temps de réponse : si e' désigne un événement de début d'invocation et e la fin de cette invocation le temps de réponse est borné supérieurement par ϵ_2 unités de temps et inférieurement par ϵ_1 unités de temps.

Considérons maintenant une autre inéquation :

$$\forall\ n\ :\ \epsilon_1 \leq \tau(e, n) - \tau(H_p, n) \leq \epsilon_2 \qquad [2]$$

où H_p modélise une horloge de période p dont la date d'occurrence du n ième top est donnée par $\tau(H_p, n) = n * p$ et e désigne un événement. Cette inéquation permet de modéliser une synchronisation intra-flux pour un flux périodique avec gigue. Les éléments successifs du flux doivent être présentés avec une gigue maximale de $\epsilon_2 - \epsilon_1$ unités de temps par rapport aux tops de l'horloge H_p. Contrairement à l'inéquation [1], le flux ne peut dériver car il est asservi aux tops de l'horloge H_p. Une telle inéquation implique que deux occurrences de e soit séparées par au moins $p - (\epsilon_2 - \epsilon_1)$ et au plus $p + (\epsilon_2 - \epsilon_1)$ unités de temps.

Enfin, une version plus contrainte de l'inéquation [2] peut prendre la forme suivante :

$$\forall\ n\ :\ \tau(e, n) = n * p \qquad [3]$$

Cette équation modélise une contrainte intra-flux sur un flux isochrone dans lequel on aurait compensé la gigue pour que les occurrences de e se produisent à un rythme régulier.

2.3. *Illustration par un exemple*

Prenons l'exemple de l'application introduite ci-dessus, et déterminons un système d'équations de QoS, que nous désignerons par $(S0)$, correspondant à la QoS que peut souhaiter un utilisateur de l'application. L'application restitue le flux audio sur une carte audio ; elle utilise un serveur X11 pour afficher la vidéo.

Une carte audio compense la gigue sur le flux de données qui lui est fourni : elle possède un tampon que l'utilisateur alimente et elle restitue de façon autonome les données sur la sortie audio. La carte utilisée a une fréquence d'échantillonnage de 8000 Hz : elle consomme un octet tous les $0,125$ ms. Pour assurer une QoS satisfaisante, il faut donc qu'un octet soit restitué toutes les $0,125$ ms, ce qui est spécifié par l'équation :

$$\forall\, n \;:\; \tau(a,n) = n * 0,125\ ms \qquad [S0_1]$$

où a désigne l'événement de restitution d'un échantillon audio. Par la suite, nous noterons Ha_{pa} l'horloge logique associée à l'équation précédente et dont la période est $pa = 0,125$ ms.

Si l'utilisateur souhaite afficher 10 images par seconde (c'est-à-dire une image toutes les 100 ms), en autorisant une gigue maximale de 40 ms entre deux images successives, la QoS qui doit être offerte par le composant « serveur X11 » est de la forme :

$$\forall\, n \;: 80\ ms \leq \tau(i,n+1) - \tau(i,n) \leq 120\ ms \qquad [S0_2]$$

où i désigne l'événement d'affichage d'une image vidéo. Enfin, sachant qu'une image doit être affichée toutes les 100 ms et qu'en 100 ms, $100/0,125 = 800$ échantillons audio sont délivrés par la carte audio, on en déduit qu'une solution pour spécifier la synchronisation inter-flux est :

$$\forall\, n \;: -40\ ms \leq \tau(a, n*800) - \tau(i,n) \leq 40\ ms \qquad [S0_3]$$

qui peut aussi s'écrire :

$$\forall\, n \;: -40\ ms \leq \tau(Ha_{pa}, n*800) - \tau(i,n) \;\leq 40\ ms$$

où 80 ms= $[-40$ ms, 40 ms$]$ constitue une gigue acceptable pour l'utilisateur [STE 95].

2.4. *Modèle de système multimédia*

Nous avons expliqué comment modéliser une application et les contraintes de QoS temporelle auxquelles elle est soumise. Toutefois, afin de pouvoir prendre en compte ces contraintes, la plate-forme d'exécution POLKA a également besoin des spécifications du comportement de certains *composants* du système multimédia tels que le réseau et la carte audio.

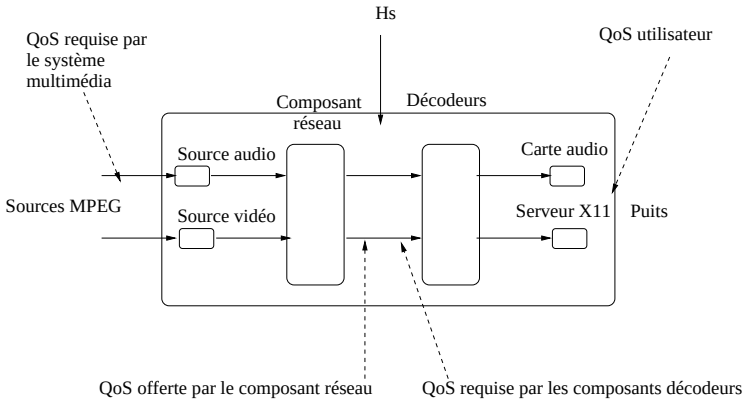

Figure 5. *Graphe d'un système multimédia*

Les traitements qui interviennent dans les applications multimédias sont souvent organisés en pipe-line ou en graphes de flots de données [AND 90, SIE 97, JEF 95, MIC 97]. Nous modélisons donc un système multimédia par un graphe de flots de données dont les nœuds correspondent aux composants du système et les flèches décrivent les flux de données multimédias (cf. figure 5). Ces flux sont vus comme une suite *d'événements* discrets, chacun étant identifié par un nom d'événement (ex : e) et un numéro d'occurrence (ex : n). L'horloge Hs est un flux entrant qui modélise le temps physique.

L'utilisateur spécifie les contraintes de QoS que le système doit respecter en sortie (ex : variations tolérées sur les délais d'affichage entre deux images consécutives, synchronisation audio-vidéo). On parle alors de *QoS utilisateur*.

Les équations de QoS spécifiées sur les flux d'entrée du système multimédia correspondent aux contraintes de QoS que le système multimédia requiert pour satisfaire la QoS utilisateur. On parle, dans ce cas, de *QoS requise*.

Examinons maintenant comment spécifier les composants d'un système multimédia. Le composant est la brique de base d'un système multimédia. Les composants sont assemblés en parallèle ou en séquence. Un composant est défini par les mêmes éléments qu'un système multimédia complet : une horloge ainsi que des flux multimédias entrants et sortants. Un composant peut éventuellement contenir un tampon lui permettant de stocker les informations véhiculées par les événements entrants.

Le comportement temporel d'un composant est défini par un *contrat de QoS*. Ce contrat est composé de deux jeux d'équations de QoS : un pour la QoS offerte et un pour la QoS requise [BLA 98]. Les termes du contrat sont les suivants : *sous réserve du respect des contraintes de QoS requises par le composant en entrée, le composant s'engage à respecter une QoS offerte donnée*. La notion de contrat offre un cadre pour déduire la QoS associée à une composition d'objets de la QoS de chaque objet [LEB 98]. Des définitions sont proposées pour ces notions en annexe A. Il est possible de définir deux types de composants : ceux dont le comportement est connu *a priori* (ex : composant réseau isochrone) et ceux dont le comportement est déduit durant l'exécution de l'application grâce aux services de supervision de POLKA (ex : composant réseau asynchrone) ; dans ce cas, les informations à obtenir (ex : délai moyen de communication) sont spécifiées dans la définition du composant.

Dans le travail présenté ici, nous utilisons les contrats pour déduire la QoS requise à la source à partir de la QoS attendue en sortie par l'utilisateur. Cette déduction se fait en remontant des puits aux sources, ce qui est une forme de composition. La QoS requise correspond à une *condition* pour que le système respecte la QoS attendue, dans la limite des ressources disponibles dans le système. En effet, la plate-forme actuelle n'offre pas de services de réservation de ressources, et ne peut donc pas garantir à l'utilisateur le respect de la QoS qu'il a spécifiée. En revanche, la plate-forme n'exige pas la fourniture d'informations temporelles telles que les temps d'exécution des invocations de méthodes. Dans le cas où une équation de QoS est violée par manque de ressources, des mécanismes d'alerte de l'application sont prévus. C'est alors de la responsabilité de l'application d'adapter son comportement de façon à restreindre ses exigences de QoS, *ce qui peut se faire pendant l'exécution en changeant les équations*. Néanmoins, le modèle présenté ci-dessus n'exclut pas l'utilisation de services de réservation. L'annexe A explique comment ajouter un modèle de ressources à POLKA, de façon à donner des garanties contractuelles et à généraliser la notion de composition.

Montrons maintenant comment modéliser l'application de démonstration introduite précédemment. La figure 6 donne le graphe de flots de données de l'application. Un composant modélise la carte audio qui restitue le flux audio en sortie, un autre le serveur X11 chargé d'afficher le flux vidéo. Les 2 composants centraux représentent les décodeurs MPEG audio et vidéo d'une part, et le réseau d'autre part. Comme proposé au paragraphe 2.2, on désigne par $(S0)$ le système d'équations correspondant à la QoS utilisateur attendue en sortie (respectée par la carte audio et le serveur X11).

On rappelle que le jeu d'équations $(S0)$ est le suivant :

$$\forall\ n\ \ : \begin{cases} \tau(a,n) \ = \ n * 0,125\ ms \\ 80\ ms \leq \tau(i,n+1) - \tau(i,n) \leq 120\ ms \\ -40\ ms \leq \tau(Ha_{pa}, n*800) - \tau(i,n) \ \leq 40\ ms \end{cases}$$

On désigne par $(S1)$ (cf. figure 6) le système d'équations qu'il est *suffisant* de respecter en entrée de la carte audio et du serveur X11, pour obtenir la QoS

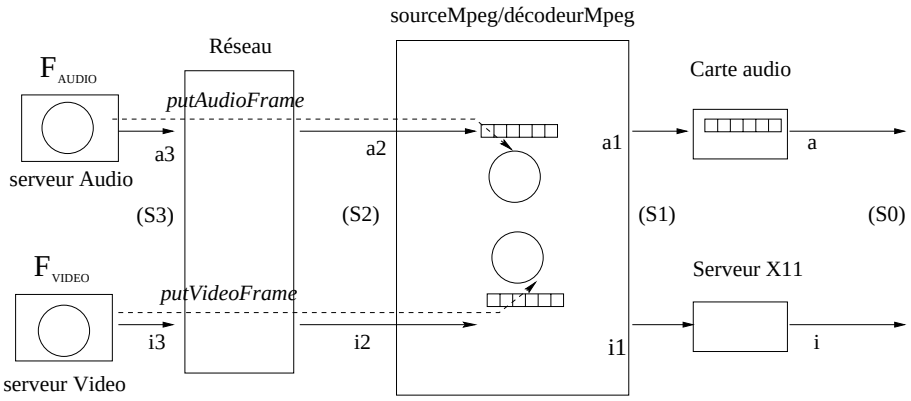

Figure 6. *Modélisation de l'application répartie*

utilisateur spécifiée par $(S0)$. $(S1)$ correspond à la QoS offerte par les composants décodeurs. On déduit alors le jeu $(S2)$ requis en entrée des composants décodeurs. Du comportement du composant réseau et du jeu $(S2)$, on déduit enfin le jeu $(S3)$ qu'il faut satisfaire en entrée du système pour que le jeu $(S0)$ soit respecté en sortie.

De proche en proche, on peut ainsi obtenir le jeu $(S3)$ à partir du jeu $(S0)$ et des contrats des composants. Le lecteur intéressé par le détail des opérations qui permettent d'obtenir le jeu $(S3)$ les trouvera dans l'annexe B.

C'est la plate-forme POLKA qui construit automatiquement les jeux $(S1)$, $(S2)$ et $(S3)$, à partir de $(S0)$ et de la spécification des composants. Chaque événement spécifié dans le graphe de flots de données est associé à un événement de type IE, IR, RE ou RR. Ceci permet de lier la modélisation de l'application en termes de composants avec sa conception objet (ex : l'événement $a3$ de la figure 6 correspond à l'événement $putAudioFrame.IE$ et l'événement $a2$ à $putAudioFrame.IR$). La construction des jeux d'équations $(S1)$ et $(S3)$ détermine finalement les équations de QoS qui seront utilisées pour ordonnancer les invocations de méthodes. Nous détaillons maintenant les objectifs et l'architecture de cette plate-forme.

3. Plate-forme d'exécution

Dans ce chapitre, nous donnons quelques éléments d'information sur l'implantation actuelle de POLKA. Puis, nous décrivons succinctement comment une application est développée avec notre plate-forme.

3.1. *Architecture et fonctionnement*

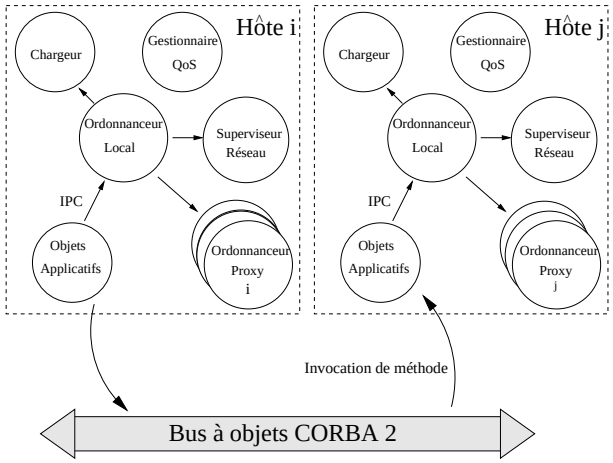

Figure 7. *Architecture du prototype*

Comme représenté en figure 1, la plate-forme d'exécution prend en entrée une spécification de l'application (en pratique des objets CORBA), les équations de QoS décrivant le comportement temporel souhaité, la spécification des composants importants du système et une description du graphe de flots de données de l'application. Elle se charge alors d'ordonnancer automatiquement l'application de façon à respecter les contraintes de QoS, si suffisamment de ressources sont disponibles dans le système. Elle produit, par ailleurs, des informations de supervision qui permettent de suivre le comportement temporel de l'application et de déterminer le comportement temporel des composants lorsqu'il n'est pas connu *a priori*.

La plate-forme POLKA est construite autour d'un bus à objets CORBA (le bus à objets omniORB2 [LO 98]). Bien que l'implantation actuelle soit réalisée à l'aide d'un bus à objets conforme au standard CORBA, il est important de préciser que le modèle de QoS ainsi que les techniques d'ordonnancement proposés dans le projet POLKA ne sont pas dédiés à CORBA. Ainsi, les systèmes à objets utilisant des protocoles de communication mieux adaptés aux applications multimédias que ne l'est IIOP (*Internet Inter-ORB Protocol*), ou qui supportent la notion de flux dans leur IDL, peuvent tout à fait bénéficier des techniques présentées dans cet article [DUM 98, DON 98]. Ces techniques peuvent être appliquées à tous les systèmes objets où des événements peuvent être captés durant les interactions entre objets.

La plate-forme est principalement constituée d'un compilateur IDL et d'un démon. Le compilateur IDL génère les souches et les squelettes CORBA et insère les traitements nécessaires à l'ordonnancement des invocations de mé-

thodes. Le démon est une application CORBA composée des objets suivants : le gestionnaire de QoS, le chargeur, l'ordonnanceur local, les ordonnanceurs *proxies* et le superviseur réseau (cf. figure 7).

Le *gestionnaire de QoS* analyse les descriptions d'application et fournit à l'ordonnanceur local les jeux d'équations de QoS.

Le *chargeur* initialise en mémoire les informations nécessaires à l'ordonnancement.

L'ordonnancement global de l'application est réalisé en faisant coopérer les *ordonnanceurs locaux* des différents sites. La plate-forme ordonnance les flots d'exécution sur chaque site du système, conformément aux équations de QoS. Ainsi, dans le cas de notre application de démonstration, le site « puits » réalise l'ordonnancement de façon à respecter le jeu $(S1)$ et le site « source » de façon à respecter le jeu $(S3)$.

Les événements IE, IR, RE et RR correspondant aux invocations de méthode de l'application, délimitent les tâches à ordonnancer. Ces événements sont également ceux manipulés dans les équations de QoS. L'ordonnanceur local les utilise donc pour associer des échéances aux tâches applicatives. Nous ne décrivons pas ici les algorithmes d'ordonnancement qui ont été présentés dans d'autres publications (voir par exemple [DEM 98b, LEB 98]).

Sur chaque site i, il existe un objet *ordonnanceur proxy* j pour chaque site j du système (avec $j \neq i$). Ces *proxies* offrent à l'ordonnanceur local une vue des informations d'ordonnancement manipulées par les objets ordonnanceurs distants.

Enfin, le *superviseur réseau* collecte des informations concernant l'état du réseau telles que le taux de perte des paquets, les délais de communication de bout en bout et la gigue maximale. Ces informations sont utilisées par les ordonnanceurs locaux.

Le superviseur réseau et les ordonnanceurs *proxies* masquent le comportement temporel du réseau, ainsi que les problèmes dus à la répartition, aux autres objets de l'architecture POLKA. Les traitements effectués par ceux-ci et le type des informations de supervision obtenues dépendent des caractéristiques temporelles du réseau sous-jacent (ex : on n'utilisera pas de supervision en présence d'un réseau offrant des services isochrones).

Notons qu'afin d'éviter l'utilisation d'une horloge globale au système, nous utilisons une technique de conversion des dates d'événements qui se base sur l'estimation des temps de communication faite par le superviseur réseau [DEM 98a].

L'utilisation de CORBA constitue un premier pas vers la portabilité des applications multimédias réparties. Il est toutefois insuffisant. En effet, hormis les faiblesses de CORBA à cet égard, l'ordonnancement automatique des applications ciblées peut demander l'utilisation de services qui sont susceptibles de différer d'un système à un autre.

Ainsi, pour permettre l'ordonnancement temps réel, certains systèmes fournissent une implantation des *threads* normalisées par POSIX. Force est de constater qu'il existe des différences parfois importantes entre les différentes implantations (que ce soit sur l'interface ou sur le comportement des services offerts).

Pour palier ces inconvénients, nous introduisons une couche « machine virtuelle » dans la plate-forme. Elle a pour rôle d'offrir une vision homogène des services du système sous-jacent utilisés par POLKA et d'harmoniser leur comportement (ex : elle configure le temporisateur de Linux avec une précision suffisante [SRI 98]).

En pratique, la machine virtuelle est constituée d'un ensemble de classes C++ *inline* offrant principalement des abstractions de *threads* [1], de mécanismes de communication inter-processus, d'outils de synchronisation intra et inter-processus et de temporisateurs. Nous en avons développé une version pour Solaris et une version pour Linux. Une implantation sur L4 [HAR 97] est en cours de réalisation.

3.2. *Chaîne de production d'une application avec POLKA*

Dans cette partie, nous détaillons les différentes étapes qui interviennent lorsqu'un utilisateur souhaite construire une application avec POLKA.

La première étape consiste à spécifier l'application en termes d'objets et de *threads*, ce qui se traduit, en particulier, par la description IDL de l'application. La spécification IDL est utilisée pour générer les souches et squelettes CORBA qui sont instrumentées par POLKA pour que l'ordonnanceur soit invoqué à chaque occurrence d'événement IE, IR, RE et RR. Pour ce faire, le développeur utilise le compilateur IDL *i2p* de POLKA.

Par la suite, l'application est développée de la même façon que n'importe quelle autre application CORBA dans un environnement *multi-thread*. Toutefois, plutôt que d'utiliser la bibliothèque de *thread* offerte par le système, le développeur exploite la bibliothèque de *thread* de POLKA. Aucune information d'ordonnancement (ex : priorité, classe d'ordonnancement, quantum, etc.) n'est précisée lors de la création des *threads* dans les programmes ; ces informations sont déduites pendant l'exécution de l'application par l'ordonnanceur POLKA grâce à la QoS spécifiée. Dans POLKA, une séparation claire existe entre la description fonctionnelle de l'application et de sa qualité de service, *puisqu'aucune information de QoS n'est donnée dans le code de l'application.*

[1] L'abstraction de *thread* définie dans la machine virtuelle n'est pas directement manipulée par le développeur. Elle constitue une abstraction de flot d'exécution centralisée sur un seul processeur. En fait, les *threads* de la machine virtuelle sont utilisées par la plate-forme POLKA pour construire la notion de *threads* « à la Clouds » qui constitue l'abstraction finalement manipulée par l'utilisateur.

Une fois l'application écrite, il est nécessaire de spécifier le modèle de QoS qui permettra d'ordonnancer les *threads* de l'application. La phase de modélisation commence par l'identification des composants du système et de l'application. Modéliser un composant consiste à déterminer le jeu d'équations de QoS requise pour chaque jeu d'équations de QoS offerte. Ce travail nécessite une connaissance précise du comportement temporel du composant. Néanmoins, POLKA offre une bibliothèque de composants génériques qui modélisent des comportements temporels usuels et qui peuvent être instanciés pour modéliser un composant spécifique (ex : un composant qui applique à chaque événement d'entrée un retard compris entre une borne maximale et une borne minimale peut être instancié pour modéliser un réseaux offrant des services de communication isochrone [STE 95]).

Un utilisateur peut construire un composant donné par composition parallèle ou séquentielle de plusieurs composants de la bibliothèque, puis, l'instancier avec des valeurs spécifiques (ex : délais minimal et maximal de transit). La bibliothèque peut être enrichie par l'utilisateur.

Une fois les composants modélisés, le développeur doit construire le graphe flots de données qui représente le système dans sa globalité. Cette étape consiste à connecter les différents composants du système. Les événements IE, IR, RE et RR du modèle objet constituent les éléments permettant d'effectuer cette connection. Ainsi, dans la figure 6, la connection entre le composant réseau et le composant F_{audio} s'effectue en associant l'événement $putAudioFrame.IE$ à l'événement $a3$ et l'événement $putAudioFrame.IR$ à l'événement $a2$.

Pour aider le développeur, un compilateur de QoS (le compilateur *qc*) analyse et effectue la composition des descriptions de composant. Le compilateur calcule aussi, à partir du graphe de flots de données, les différents jeux d'équations de QoS qui doivent être fournis sur chacun des processeurs du système. Ce calcul est effectué en remontant les contraintes de QoS des puits jusqu'aux sources du graphe de flots de données. Les algorithmes utilisés par le compilateur de QoS sont réutilisés par le gestionnaire de QoS de la plate-forme POLKA pour analyser les informations de QoS durant l'exécution de l'application.

L'annexe C donne un extrait de la description de QoS de notre application. Cette dernière est constituée de deux parties. La première partie spécifie certains des composants de l'application. Un exemple de composition parallèle est donné (composant *decodeur*). La deuxième partie décrit le graphe de flots de données qui instancie les composants en précisant sur quelle machine ils sont situés. C'est à ce niveau que le lien avec le modèle objet est réalisé et que sont précisées les éventuelles informations à obtenir par le service de supervision. Il n'est pas possible, dans cet article, de décrire de façon plus détaillée cette description.

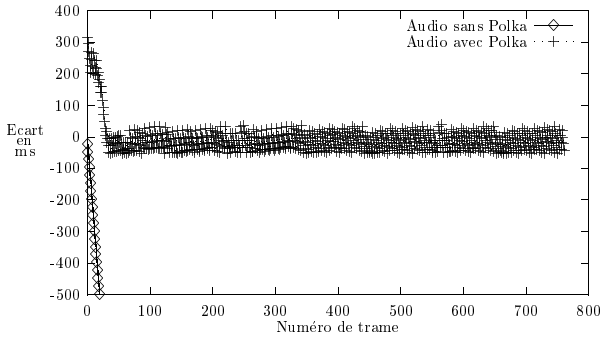

Figure 8. *Respect des synchronisations intra-flux*

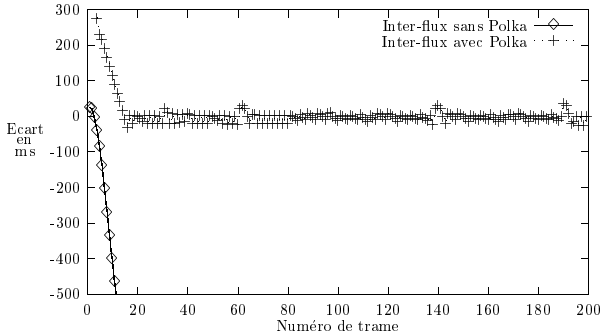

Figure 9. *Respect des synchronisations inter-flux*

4. Evaluation de l'approche

Evaluons maintenant le surcoût induit par l'utilisation de la plate-forme POLKA, et son impact sur les synchronisations intra et inter-flux d'une application. Pour effectuer les mesures présentées ici, nous avons utilisé l'application définie dans les paragraphes précédents mais sans la présentation et le décodage des trames MPEG-2 (la présentation et le décodage sont simulés avec les durées d'exécution observées sur la version Solaris de l'application). La taille des trames audio est de 627 octets, celle des trames vidéo est de 4 096 octets. L'application est exécutée sans POLKA, puis avec POLKA. L'application est répartie sur deux machines Linux (Pentium 200 MMX avec 64 Mo de mémoire vive) connectées par un bus Ethernet à 10Mbits/s.

La courbe de la figure 8 positionne les dates de livraison des trames audio à la carte audio par rapport aux tops de l'horloge Ha_{pa}. Lorsque la présentation est synchronisée avec un top d'horloge, la courbe croise la droite d'ordonnée zéro. Une variation de 64 ms autour de cette droite correspond à la tolérance

sur la synchronisation intra-flux (système d'équations ($S1$)). On peut observer que sans POLKA les trames audio sont délivrées de façon complètement asynchrone aux tops d'horloge. Avec POLKA, comme le montre la courbe, la présentation des trames est asservie à l'horloge. Les contraintes de QoS intra-flux sont respectées.

La deuxième courbe (cf. figure 9) montre le délai entre l'affichage d'une image et la présentation des trames audio associées. Lorsque la courbe croise la droite d'ordonnée zéro, la synchronisation inter-flux est optimale. POLKA prouve, là encore, son efficacité.

Examinons maintenant le surcoût engendré par POLKA. Ce dernier comprend principalement le temps nécessaire pour calculer les échéances des tâches et pour transmettre les données locales d'ordonnancement vers d'autres ordonnanceurs. Ce surcoût s'évalue à 2,8 ms par invocation de méthode en réparti et 1,5 ms en centralisé (ce surcoût a été évalué à 840 μs sur une UltraSparc 1 à 167 MHz avec 128 Mo de mémoire vive sous Solaris). Si l'on compare ces chiffres au temps nécessaire pour décoder et afficher une image (55,24 ms), le surcoût généré par POLKA reste raisonnable (de l'ordre de 5 pourcent).

5. Modélisation et ordonnancement d'applications multimédias : travaux similaires

Nous donnons ici quelques éléments de comparaison de POLKA à des travaux qui portent, soit sur la spécification et l'ordonnancement des tâches d'une application multimédia, soit sur la modélisation d'un système multimédia, soit sur d'autres plates-formes multimédias.

Pour spécifier les contraintes temporelles et ordonnancer une application multimédia, on peut utiliser des techniques qui s'adressent aux applications temps réel en général. La littérature sur l'ordonnancement temps réel distingue les tâches répétitives des tâches non répétitives [LEB 98]. Les tâches répétitives font l'objet de plusieurs activations successives : le modèle de tâches répétitives le plus répandu est celui des tâches périodiques [LIU 73]. Ce modèle a fait l'objet de nombreux travaux [STA 95, GEO 96]. Une variante de la tâche périodique est la tâche sporadique, avec un délai minimal entre les arrivées de deux activations successives. Pour ordonnancer ces tâches, deux grandes familles d'algorithmes ont été proposées : RM (*Rate Monitonic*) qui associe une priorité aux tâches de façon statique et EDF (*Earliest Deadline First*) où les priorités sont attribuées dynamiquement. Ces techniques d'ordonnancement sont utilisées dans TAO [GIL 99], un bus à objets offrant des services d'ordonnancement.

Notons que d'autres travaux sur les bus à objets et les contraintes temps réel ont vu le jour : ils portent essentiellement sur des aspects architecturaux ; c'est notamment le cas de Jonathan [DUM 98], DIMMA [DON 98] ou QuO [VAN 98].

Mercer et al. proposent un modèle où les tâches sont décrites par une période de réservation et un pourcentage de temps processeur [MER 94]. Ce pourcentage est aisément convertible en temps d'exécution d'une activation. Les tâches apériodiques sont dotées d'une période « artificielle » et les algorithmes utilisés sont soit EDF, soit RM. Un mécanisme de rétroaction est proposé pour ajuster la réservation de ressource.

Buddhikot et al. utilisent aussi le modèle de Liu et Layland au travers du concept de RTU (*Real Time Upcall*). Un RTU est une fonction qui peut être exécutée périodiquement par l'ordonnanceur dans l'espace d'adressage de l'application [BUD 96].

Le concept de RTU est particulièrement adapté à la mise en œuvre de protocole de communication applicatif : en effet, il peut aider à limiter les multiples recopies de données (l'exécution se faisant dans l'espace applicatif). L'ordonnanceur utilise une technique proche de RM. Un contrôle d'admission et une réservation du processeur sont effectués pour chaque RTU nouvellement créé.

Les contraintes de QoS décrites ci-dessus restent très simples et parfois peu pratiques pour des applications multimédias. En effet, dans une application multimédia, on s'intéresse souvent à des contraintes autres qu'une échéance relative à une période. Il est courant de spécifier des contraintes de délais minimaux et/ou maximaux entre deux fins de tâches. A titre d'exemple, un utilisateur peut vouloir spécifier un délai minimal et maximal entre l'affichage de deux images successives. On parle alors de contraintes de « distance » [BUC 93, STE 95, SAK 95, HAN 96]. Malheureusement, les plates-formes qui utilisent ces modèles offrent une flexibilité limitée : les possibilités d'adaptation dynamique du comportement temporel de l'application sont pauvres.

Jones [JON 97] propose une solution où les contraintes de QoS sont directement spécifiées dans le code de l'application. Lors de son démarrage, l'application réserve des unités de temps sur une période donnée. Pendant son exécution, elle délimite, grâce à une interface de programmation, les portions de code où doivent s'appliquer les contraintes de QoS. Les contraintes de QoS sont des contraintes d'échéance, de date de début d'exécution et de criticité. Elles ne sont pas spécifiées lors de la réservation du processeur mais au moment où le bloc de code doit être exécuté. L'ordonnanceur prédit alors si la contrainte de QoS pourra être respectée (c'est-à-dire si la contrainte de QoS spécifiée reste compatible avec la réservation préalablement effectuée). C'est de la responsabilité de l'application d'exploiter convenablement cette prédiction. L'ordonnanceur est basé sur EDF.

SMART [NIE 97] autorise également la spécification de contraintes de QoS (sous forme d'échéances) sur des portions de code. L'application doit fournir une estimation du temps d'exécution pour chaque bloc. Les informations temporelles sont également intégrées dans le code de l'application. L'ordonnanceur offre un mécanisme d'*upcall* permettant à l'application d'effectuer des traitements lorsqu'une contrainte temporelle ne peut être respectée. L'ordonnanceur de SMART est conçu pour faire coexister des tâches avec et sans contraintes

temporelles. Pour ce faire, il associe deux informations par tâche pour effectuer son allocation de ressource : une notion d'importance et une notion d'échéance. Schématiquement, l'élection d'une tâche par l'ordonnanceur est effectuée en deux étapes : d'abord en construisant l'ensemble des tâches de plus grande importance, puis en choisissant la plus urgente de cet ensemble. Les tâches de même importance partagent de façon équitable la ressource processeur. A cet effet, la notion d'importance est constituée d'une partie statique spécifiée par l'utilisateur et d'une partie dynamique, calculée par l'ordonnanceur.

La solution de Brandt et al. [BRA 98] est plus radicale : ils ne font pas de réservation mais spécifient plusieurs niveaux de QoS. Ils proposent d'associer non plus un pourcentage de processeur à chaque tâche mais plusieurs niveaux de pourcentages avec un degré de satisfaction pour chaque niveau. Dans leur implantation, un gestionnaire de QoS affecte un niveau de QoS à chaque tâche en fonction des ressources disponibles. La plate-forme n'utilise pas d'ordonnanceur particulier : la modification des besoins en ressource est effectuée par les applications qui s'adaptent au niveau de QoS dans lequel le gestionnaire les a placées.

Toutes les solutions présentées ci-dessus demandent au développeur un travail important : il doit déterminer les tâches de son système puis définir les contraintes temporelles de celles-ci. Or, ces tâches sont rarement indépendantes. Par exemple, la contrainte de QoS offerte par une tâche qui affiche une image dépendra de la QoS offerte par la tâche qui aura lu l'image sur le disque. Les modèles décrits ci-dessus ne s'attachent pas à spécifier ces dépendances entre tâches et leur prise en compte doit être faite à la main.

Certaines équipes ont néanmoins proposé des solutions qui traitent ce problème ; Jeffay [JEF 95] utilise un modèle de flots de données où chaque nœud constitue une tâche ou une ressource. Une tâche de l'application multimédia est contrainte par une échéance et est définie par une loi d'arrivée plus générale que celle initialement décrite par Liu et Layland : ici, une tâche peut être activée x fois pendant une période de temps y (on parle de cadence d'activation). Aucune hypothèse particulière n'est faite sur la distribution des x activations sur la période de temps y. Seuls les nœuds sources du graphe sont définis par la cadence d'activation : les contraintes des nœuds suivants sont calculées grâce à leur temps d'exécution et les contraintes de leurs nœuds successeurs. Le modèle est proche (bien que plus général) du modèle LBAP (*Linear Bounded Arrival Process*) proposé par Anderson [AND 90] dans Dash, et pour lequel, là aussi, l'application est décrite sous la forme d'un flot de données représentant les ressources utilisées (ex: tampon mémoire). Les cadences d'activation ont été implantées sur YARTOS [JEF 92]. Dans cette plate-forme, un service de réservation de ressources est disponible. A chaque arrivée d'une tâche, un contrôle d'admission est réalisé. Le cas échéant, il est possible de renégocier les ressources allouées par une tâche (ex : dans le cas où la source n'est plus conforme à la spécification de la cadence d'activation).

Diaz et Owezarski [OWE 97, OWE 98] proposent aussi de modéliser les dépendances entres les tâches mais cette fois au moyen d'un réseau de Petri temporel : le TSPN (*Time Stream Petri Net*). Le réseau de Petri modélise les synchronisations appliquées sur les flux multimédias. De plus, il permet une validation formelle des contraintes. Dans un TSPN, chaque flèche sortant d'une place est complétée d'un intervalle de validité sur la présentation d'un élément de flux multimédia. L'intervalle est constitué d'un triplet (x, y, z) où x correspond à une contrainte temporelle souhaitée sur la présentation, y et z représentent les dérives minimales et maximales souhaitées sur cette contrainte. Les synchronisations sont alors spécifiées grâce au choix de la sémantique de tir de la transition. Le modèle est exploité par leur plate-forme pour réaliser les synchronisations ainsi spécifiées. POLKA et les travaux de Diaz et Owezarski ont en commun la spécification des contraintes temporelles dans un modèle de haut niveau et l'exploitation de ce modèle pour la réalisation des synchronisations nécessaires. Toutefois, à notre connaissance, Diaz et Owezarski ne proposent pas de stratégies d'adaptation des contraintes temporelles alors qu'il existe beaucoup d'applications multimédias où cette adaptation est nécessaire.

Enfin, DirectShow [MIC 97], de Microsoft, utilise aussi un graphe de flots de données pour spécifier une application multimédia. Ces flots de données traversent des composants qui effectuent soit des traitements, soit de la présentation, soit produisent les flots de données (sources). DirectShow offre à l'utilisateur la possibilité de composer en ligne ou hors ligne les graphes de flots de données au travers d'un éditeur de graphes. Construire le graphe déclenche le lancement des différents composants dans le système. Une interface de contrôle de flux permet de suspendre et de démarrer les flux de données. Enfin, les composants de présentation peuvent renvoyer des informations de rétroaction aux composants en amont. Ces informations de QoS sont simples *et peu flexibles* ; elles sont essentiellement constituées d'une information précisant si le composant est en surcharge ou en sous-charge, d'un rythme de traitement des données auquel les prédécesseurs doivent se conformer, ainsi que d'une estimation du retard ou de l'avance que le composant de présentation a pris. La QoS est gérée par chaque composant ; le code de l'application est donc fortement lié à sa QoS. Enfin, il n'est pas possible de spécifier des contraintes de QoS entre des flots traités par des composants différents.

Nous avons décrit plusieurs travaux qui portent, soit sur la spécification et l'ordonnancement des tâches d'une application multimédia, soit sur la modélisation d'un système multimédia dans sa globalité. POLKA propose un modèle de QoS plus général, qui, une fois traité par la plate-forme POLKA fournit une expression de QoS exploitable par un ordonnanceur avec un surcoût raisonnable. Notons qu'il est aisé de spécifier les modèles classiques d'ordonnancement (tâches périodiques, sporadiques et apériodiques), les contraintes de distances et bon nombre des contraintes temporelles présentées ci-dessus par des équations QL.

De plus, POLKA sépare la spécification de QoS du code de l'application, ce qui rend plus portable et plus flexible les applications ainsi développées. Enfin, cette séparation de la QoS et du code autorise POLKA à implanter plusieurs stratégies permettant de modifier la QoS de l'application durant son exécution.

6. Conclusion

Dans cet article, nous avons décrit comment modéliser et exécuter une application multimédia avec POLKA. POLKA permet au concepteur, à partir d'une description des composants du système, des objets de son application et d'équations de QoS (éventuellement dissimulées à l'utilisateur par une interface de haut niveau), d'ordonnancer automatiquement une application présentant des contraintes temporelles. POLKA répond bien aux besoins de dynamicité des applications multimédias dont le comportement est partiellement prédictible et où les besoins en ressources sont difficiles à évaluer.

Bien que les techniques présentées dans cet article puissent être utilisées dans tout système à objets où les interactions entre objets sont observables, notre implantation actuelle est basée sur un bus à objets CORBA 2 sur Solaris et Linux. Ce choix a simplifié sa mise en œuvre. Les mesures montrent l'efficacité de POLKA. Toutefois, l'application de démonstration qui manipule une importante quantité de données multimédias, nous a rappelé que le surcoût impliqué par CORBA et notre ordonnanceur ne sont pas négligeables. Comme d'autres équipes, nous pensons que la résolution de ces problèmes de performances passe par l'utilisation d'un bus à objets [DUM 98, DON 98] et d'un système d'exploitation [HAR 97, HäR 98] mieux adaptés à notre problématique. Des expériences sur le micro-noyau L4 sont en cours.

Enfin, dans cet article, le modèle de QoS proposé reste limité à des contraintes *déterministes* pour lesquelles nous avons étudié le pire cas. De ce fait, nos spécifications peuvent surcontraindre le comportement temporel des applications [BAI 96]. De plus, les contraintes considérées ne peuvent modéliser efficacement des trafics dont le débit est variable. A court terme, une première extension du travail présenté ici consiste donc à étendre le modèle pour prendre en compte des contraintes *probabilistes*.

Enfin à plus long terme, nous envisageons le support par POLKA de la diffusion de groupes et de la garantie de service.

7. Remerciements

Ce travail est financé par un contrat de l'ENST avec l'équipe DTL/ASR du CNET.

8. Bibliographie

[AND 90] Anderson D. P. , « Meta-scheduling for distributed continuous media ». Technical Report UCB/CSD 90/599, University of California, Berkeley, October 1990.

[BAI 96] Baiceanu V. , Cowan C. , McNamee D. , Pu C. , et Walpole J. , « Multimedia Applications Require Adaptive CPU Scheduling ». Workshop on Resource Allocation Problems in Multimedia Systems, Washington DC, December 1996.

[BLA 98] Blair G. et Stefani J. B. , *Open Distributed Processing and Multimedia*. Addison-Wesley, 1998.

[BRA 98] Brandt S. , Nutt G. , Berk T. , et Humphrey M. , « Soft Real-Time Application Execution with Dynamic Quality of Service Assurance ». International Workshop on Quality of Service (IWQOS'98), Napa - CA, May 1998.

[BUC 93] Buchanan M.C. et Zellweger P.T. , « Automatically Generating Consistent Schedule For Multimedia Applications ». *Multimedia Systems*, 1(2):55–67, 1993.

[BUD 96] Buddhikot M. M. , Parulkar G. M. , et Gopalakrishnan R. , « Scalable Multimedia-On-Demand via World-Wide-Web (WWW) with QOS Guarantees ». pages 23–26. Sixth International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV'96, Zushi, Japon, April 1996.

[CAM 96] Campbell A. , Aurrecoechea C. , et Hauw L. , « A Review of Quality of Service Architectures ». pages 173–194. Université de Columbia, in Proceedings of the 4th international IFIP Workshop on Quality of Service, Paris, March 1996.

[DAS 90] Dasgupta P. , Chen R.C. , Menon S. , Pearson M. , Ananthanarayanan R. , Ramachandran U. , Ahamad M. , Jr. R. LeBlanc , Applebe W. , Bernabeu-Auban J. M. , Hutto P.W. , Khalidi M.Y.A. , et Wileknloh C. J. , « The Design and Implementation of the Clouds Distributed Operating System ». *Computing Systems Journal*, 3(1):11–46, Winter 1990.

[DEM 98a] Demeure I. et Singhoff F. , « Environnement d'exécution pour les applications réparties sous contraintes temporelles : une solution CORBA-RTP ». pages 53–57. 10 ième Rencontres Francophones du Parallélisme (RENPAR'10) - Strasbourg, juin 1998.

[DEM 98b] Demeure I. et Singhoff F. , « Spécification et ordonnancement dynamique d'applications multimédias : l'environnement Polka ». pages 101–117. Real time systems'98, Paris, janvier 1998.

[DON 98] Donaldson D. I. , Faupel M. C. , Hayton R. J. , Herbert A. J. , Howarth N. J. , Kramer A. , MacMillan I. A. , Orway D. D. J. , et Waterhouse S. W. , « DIMMA - A Multi-Media ORB ». pages 141–156. MIDDLEWARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.

[DUM 98] Dumant B. , Horn F. , Dang-tran F. , et Stefani J. B. , « Jonathan : an Open Distributed Processing Environment in Java ». pages 173–190. MIDDLE-

WARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.

[GEO 96] George L. , Rivierre N. , et Spuri M. , « Preemptive and Non-Preemptive Real-time Uni-processor Scheduling ». Rapport de recherche de l'INRIA numéro 2966, 1996.

[GIL 99] Gill C. D. , Levine D. L. , et C.Schmidt D. , « The Design and Performance of a Real-Time CORBA Scheduling Service ». *To appear in the International Journal of Time-critical Computing Systems*, April 1999.

[HAN 96] Han C. C. , Lin K. J. , et Hou C. J. , « Distance-constrained Scheduling and Its Applications to Real-time Systems ». *IEEE Transactions on computers*, 45(7):814–826, Jully 1996.

[HAR 97] Hartig H. , Hohmuth M. , Liedtke J. , Schonberg S. , et Wolter J. , « The Performance of micro-Kernel-Based Systems ». 16th ACM Symposium on operating Systems Principles in Saint-Malo (SOSP'97) - France, October 1997.

[HäR 98] Härtig H. , Baumgartl R. , Borriss M. , Hamann Cl.-J. , Hohmuth M. , Mehnert F. , Reuther L. , Schönberg S. , et Wolter J. , « DROPS - OS Support for Distributed Multimedia Applications ». In the proceedings of the Eigth ACM SIGOPS European Workshop, Sintra, Portugal, September 1998.

[ISO 95] ISO , « Press Release, 29th Meeting of JTC 1/SC 29/WG 11 ». number 1110, March 1995.

[JEF 92] Jeffay K. , Stone D. L. , et Smith F. D. , « Kernel Support for Live Digital Audio and Video ». *Computer Communications*, 15(6):388–395, August 1992.

[JEF 95] Jeffay K. et Bennett D. , « A Rate-Based Execution Abstraction For Multimedia Computing ». *In Lectures Notes in Computing Science, T. D. C. Little and R. Gusella, Springer-Verlag, Heidelberg*, 1018:64–75, April 1995.

[JON 97] Jones M. , Rosu D. , et Rosu M. , « CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities ». 16th ACM Symposium on operating Systems Principles in Saint-Malo (SOSP'97) - France, October 1997.

[KOF 96] Kofman D. et Gagnaire M. , *Réseaux Haut Débit: réseaux ATM, réseaux locaux, réseaux tout-optiques.* Masson-Inter Editions, Collection IIA, 1996.

[LAM 94] Lamport L. , « The Temporal Logic of Actions ». *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[LEB 97] Leboucher L. et Najm E. , « A framework for real-time QoS in distributed systems ». IEEE Workshop on Middleware for Real-Time and services conference, San Francisco, 1997.

[LEB 98] Leboucher L. , « *Algorithmique et Modélisation pour la Qualité de Service des Systèmes Répartis Temps Réel* ». Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications, septembre 1998.

[LIU 73] Liu C. L. et Layland J. W. , « Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environnment ». *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[LO 98] Lo S. L. et Pope S. , « The implementation of a High Performance ORB over Multiple Network Transports ». pages 157–172. MIDDLEWARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.

[MER 94] Mercer C. W. , Savage S. , et Tokuda H. , « Processor Capacity Reserves: Operating System Support for Multimedia Applications ». In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, May 1994.

[MIC 97] Microsoft , *PC 98 System Design Guide*. Microsoft Press, September 1997.

[NIE 93] Nieh J. , Northcutt J. N. , et Hanko J. G. , « SVR4 UNIX Scheduler unacceptable for multimedia applications ». Dans *Lecture Notes in Computer Science, Vol846, Springler-Verlag, Proceedings of the 4th International Workshop on NOSDAV*, pages 49–60, Lancaster, U.K., November 1993.

[NIE 97] Nieh J. et Lam M. , « The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications ». 16th ACM Symposium on operating Systems Principles in Saint-Malo (SOSP'97) - France, October 1997.

[OMG 98] OMG , « The Common Object Request Broker : Architecture and Specification. Revision 2.2 ». TC Document 98-07-01, February 1998.

[OWE 97] Owezarski P. et Diaz M. , « Hierarchy of time streams Petri nets models in generic videoconferences ». pages 59–72. Proc. international Conference on Application and Theory of Petri Nets Workshop on Multimedia and Concurrency, Toulouse (France), June 1997.

[OWE 98] Owezarski P. et Diaz M. , « Conception et implémentation d'applications multimédias en Solaris 2 ». *Calculateurs Parallèles, Réseaux et Systèmes répartis*, 10(3):311–331, juillet 1998.

[SAK 95] Saksena M. , Gerber R. , et Pugh W. , « Parametric Dispatching of Hard Real-time Tasks ». *IEEE Trans. on Computers*, 44(3):471–479, 1995.

[SCH 96] Schulzrinne H. , Casner S. , Frederick R. , et Jacobson V. . « RFC1889 : RTP : A Transport Protocol for Real-Time Applications ». Network Working Group, pages 1-75, January 1996.

[SIE 97] Siewert S. , Nutt G. , et Humphrey M. , « Real-Time Parametrically Controlled In-Kernel Pipelines ». Third IEEE Real Time Technology and Application Symposium (RTAS'97), Work-In-Progress, Montreal - Canada, June 1997.

[SRI 98] Srinivasan B. , Pather S. , Hill R. , Ansari F. , et Niehaus D. , « A Firm Real Time System Implementation using Commercial Off-The-Self Hardware and Free Software ». pages 112–119. 4th IEEE Symposium on Real-time Technology and Applications, Denver Colorado, June 1998.

[STA 95] Stankovic J. , Spuri M. , Natale M. Di , et Buttazzo G. , « Implications of Classical Scheduling Results For Real-Time Systems ». *IEEE Computer*, 28(6):16–25, June 1995.

[STE 93] Stefani J. B. , « Computational Aspects of QoS in an object-based, distributed systems architecture ». 3rd Workshop on Responsive Computer systems, Lincoln, NH, USA, September 1993.

[STE 95] Steinmetz R. et Nahrstedt K. , *Multimedia : Computing, communicating and applications*. Prentice Hall, innovative technology series, 1995.

[STE 96] Stefani J.B. , Blair G.S. , Coulson G. , Papathomas M. , Robin P. , Horn F. , et Hazard L. , « A programming Model and System infrastructure for real-time synchronization in distributed multimedia systems ». *IEEE Journal on selected areas in communications*, 14(1):249–263, January 1996.

[VAN 98] Vanegas R. , Zinky J. A. , Loyall J. P. , Karr D. , Schantz R. E. , et Bakken D. E. , « QuO's Runtime Support for Quality of Service in Distributed Objects ». pages 207–222. MIDDLEWARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.

[VET 95] Vetter Ronald J. , « ATM Concepts, Architectures, and Protocols ». *Communications of the ACM*, 38(2):30–38, February 1995.

A. Annexe: garanties contractuelles

Cette annexe explique brièvement comment introduire des garanties sur le respect des contraintes de QoS posées sur une application dans POLKA. Nous expliquons sous quelles hypothèses ces contraintes sont respectées.

1.1. *Notion de relation de QoS*

Les contrats de QoS utilisés par POLKA s'inspirent de la notion de relation temporelle (ou de QoS) formalisée dans [LAM 94] et adaptée en contexte objet dans [LEB 97]. Cette notion sépare clairement (et temporellement) la responsabilité d'un composant de celle de son environnement.

Définition **1** (Relation de QoS) *: Une relation de QoS attachée à un objet, ou un groupe d'objets, O peut être écrite sous la forme* $E \rightarrow M$ *où E est une propriété de sûreté*[2] *qui s'applique sur l'environnement*[3] *de O, M est une*

[2] Intuitivement, une propriété de sûreté est une propriété qui peut être réfutée en un temps fini car elle énonce que quelque chose de « mauvais » ne va pas se produire (par exemple, les expressions QL utilisées pour exprimer les contraintes temporelles de notre application Audio/Vidéo peuvent être violées ce qui suffit à les réfuter). A l'inverse, une propriété de vivacité ne peut être réfutée en un temps fini; elle annonce typiquement que quelque chose de « bien » va se produire.

[3] Intuitivement, l'environnement de O inclut tout les objets autres que O dans l'univers considéré.

propriété de sûreté qui s'applique sur O et $E \rightarrow M$ est une implication temporelle qui énonce que M ne peut être violée avant E.

Sur l'exemple Audio/Vidéo utilisé comme support applicatif dans cet article (cf. figure 6), ce modèle permet d'écrire $S0 \rightarrow S1$ comme la relation de QoS associée au composant formé par l'association de la carte audio et du serveur X11. On vérifie aisément que les jeux d'équations $(S1)$ et $(S0)$ (cf. annexe B) sont des propriétés de sûreté. D'après la définition 1, on a donc bien une relation de QoS qui énonce que, tant que l'environnement de ce composant ne viole pas l'hypothèse $(S1)$, alors ce composant lui offre sa capacité $(S0)$.

Notons que ce type d'énoncé peut être produit lors de la conception d'un composant générique (préalablement à son utilisation), lors de la spécification d'une application ou encore modifié dynamiquement durant une phase de négociation. De même, une relation de QoS peut être attachée à un composant, un groupe de composants ou même un artefact de composant. Par exemple, $E \rightarrow TRUE$ représente la relation de QoS d'un pseudo-composant « utilisateur » qui serait satisfait[4] tant que l'hypothèse E se vérifie sur son environnement.

1.2. *Notion de composition*

Prise isolément, une relation de QoS peut ne pas offrir de garantie contractuelle (trivialement, la relation de QoS : $E \rightarrow TRUE$ n'est ainsi garantie que si la propriété de sûreté E vaut $TRUE$). Généralement, on cherche plutôt à maîtriser les capacités et les hypothèses respectives d'un ensemble de composants en associant leurs relations de QoS. On joue en quelque sorte sur l'offre et la demande de ces composants en termes de QoS. Pour cela, [LAM 94] établit un résultat de composition qui, adapté à 2 objets, donne :

Théorème **2** (Composition) *: Soient $O1$ et $O2$, 2 objets ayant pour contraintes de QoS respectives $E1 \rightarrow M1$ et $E2 \rightarrow M2$. Soit $E = TRUE$ l'hypothèse sur l'environnement*[5] *de $O1$ et $O2$. On a la règle d'inférence suivante :*

$$\frac{(M1 \cap M2) \subseteq (E1 \cap E2)}{(E1 \rightarrow M1) \cap (E2 \rightarrow M2) \subseteq (TRUE \rightarrow M1 \cap M2)}$$

Cette règle (qui peut aisément être généralisée à n objets) se lit comme suit : si la conjonction des offres de $O1$ et $O2$ implique logiquement la conjonction des hypothèses de $O1$ et $O2$, alors les relations de QoS de $O1$ et $O2$ impliquent logiquement la relation de QoS suivante : $TRUE \rightarrow M1 \cap M2$. En d'autres termes, $O1$ et $O2$ sont composables puisque mis ensemble ils conduisent à un

[4] Ce que l'on modélise ici par la propriété de sûreté élémentaire $TRUE$.

[5] Signifiant tout simplement qu'il n'y a pas d'hypothèse sur l'environnement dans lequel sont plongés $O1$ et $O2$.

contrat plus global qui garantit l'ensemble des offres de QoS de $O1$ et $O2$ sans être contraints par leurs hypothèses respectives.

Ce résultat est de peu d'intérêt pour notre exemple Audio/Vidéo puisque les relations de QoS ont été spécifiées pour obtenir à la main une composition valide. On impose en fait l'hypothèse ($S3$) (cf. annexe B) sur les trafics en entrée du composant réseau pour aboutir à un contrat global $TRUE \rightarrow S0 \cap S1 \cap S2$ en remontant en cascade des composants puits aux sources. Ceci se vérifie aisément par le théorème de composition.

Ce résultat devient plus intéressant, si l'on souhaite prendre en compte des modifications fréquentes sur cette application en automatisant la preuve des garanties offertes. On peut, par exemple, envisager de modifier les exigences sur les débits, ajouter de nouveaux flux Audio/Vidéo à la demande ou même réutiliser les composants pour de nouvelles applications (par exemple, une application full-duplex). L'introduire dans POLKA permettrait d'utiliser des composants génériques (par exemple en adaptant la taille des tampons), de les composer plus simplement, de remonter des exceptions lorsqu'un contrat global est violé, de renégocier les contrats élémentaires...

1.3. *Notion de modèle de ressource*

Spécifier des relations de QoS et en vérifier la composabilité n'est pas tout. Si les relations de QoS utilisées ne font pas d'hypothèse sur la disponibilité des ressources (ex : processeur, mémoire, bande passante), celles-ci ne seront pas intégrées dans les garanties obtenues. On prouve donc uniquement que les relations de QoS sont composables en présence de ressources infinies. Pour obtenir des garanties plus complètes, [LEB 98] propose de généraliser l'approche en associant des relations de QoS aux ressources partagées. Une ressource partagée R est ainsi encapsulée dans un composant auquel on associe le test d'admission suivant :

$$AdmissionTest_R \rightarrow \cap_{i=1}^{n}(AdR(Oi)) \qquad [4]$$

où $AdmissionTest_R$ est un prédicat signifiant que R n'est pas saturé et où $AdR(Oi)$ est un prédicat signifiant que l'objet Oi a été admis sur R. Une condition de faisabilité d'ordonnancement temps réel est un exemple typique d'une telle relation associée (par exemple) à un processeur. Ce type de condition étant généralement basé sur des propriétés de sûreté (énonçant que quelque chose de mauvais ne va pas se produire), on peut donc l'intégrer au théorème de composition en ajoutant bien-sûr (par conjonction[6]) le prédicat $AdR(Oi)$ aux hypothèses de chaque objet Oi devant être admis. Sur notre exemple Audio/Vidéo, on pourrait ainsi spécifier un nouveau composant processeur sur lequel s'exécuterait en concurrence le composant décodeur et le composant formé

[6] On utilise la conjonction car la relation de QoS de Oi demande maintenant de vérifier ses hypothèses initiales et d'y ajouter l'admission de Oi.

de l'association carte Audio et serveur X11. Le composant processeur donnerait alors lieu à un test d'admission (que nous ne chercherons pas à établir dans cette annexe) et les deux composants déjà existants devraient ajouter à leurs relations de QoS respectives, le prédicat d'admission $AdR(Oi)$ dans leurs hypothèses respectives.

Nous en resterons là pour cette introduction qui n'a pour autre objectif que de présenter un cadre pour étendre les services de POLKA à des garanties temporelles. Bien que de nombreux points aient été passés sous silence (qui masquent la complexité à offrir des garanties dans l'environnement distribué et non déterministe envisagé), l'utilisation de la logique temporelle proposée par Lamport et la prise en compte des ressources dans les résultats de composition nous semblent cependant être des pistes intéressantes.

B. Annexe : de la QoS offerte à la QoS requise

Nous avons expliqué comment modéliser un système multimédia par un graphe de flots de données dont les nœuds correspondent aux composants du système et les flèches décrivent les flux de données multimédias (cf. figure 5).

Rappelons que l'utilisateur spécifie les contraintes de QoS que le système doit respecter en sortie ou *QoS utilisateur*.

Les équations de QoS spécifiées sur les flux d'entrée du système multimédia correspondent aux contraintes de QoS que le système multimédia requiert pour satisfaire la QoS utilisateur ou *QoS requise*. La QoS requise par le système multimédia peut être déduite de la QoS utilisateur et des contrats de QoS des composants en remontant des puits à la source.

Prenons l'exemple d'un *composant de « retard variable »*. Il ne possède pas de tampon et applique un temps de retard variable de l tops d'horloge, où $\alpha \leq l \leq \beta$, à chaque flux entrant.

Le temps de transit d'une information dans le composant est donné par :

$$\forall\, n \;:\; \alpha \leq \tau(O,n) - \tau(I,n) \leq \beta$$

où I est l'événement correspondant à l'entrée du flux dans le composant et O sa sortie.

Pour une QoS offerte de la forme :

$$\forall\, n, r \;:\; \epsilon_1 \leq \tau(O,n+r) - \tau(O,n) \leq \epsilon_2$$

la QoS requise est :

$$\forall\, n, r \;:\; \epsilon_1 - \beta + \alpha \leq \tau(I,n+r) - \tau(I,n) \leq \epsilon_2 + \beta - \alpha$$

D'autre part, pour une QoS offerte de la forme :

$$\forall\ n\ :\ \epsilon_1 \leq \tau(H_p, n) - \tau(O, n) \leq \epsilon_2$$

où H constitue une horloge logique de période p, la QoS requise est :

$$\forall\ n\ :\ \epsilon_1 + \beta \leq \tau(H_p, n) - \tau(I, n) \leq \epsilon_2 + \alpha$$

Notons que lorsque $l = \alpha = \beta$ on a un *composant de retard fixe*.

Montrons maintenant comment modéliser l'application de transmission d'un film par le réseau. La figure 6 montre le graphe de flots de données de l'application. Un composant modélise la carte audio qui restitue le flux audio en sortie, un autre le serveur X11 chargé d'afficher le flux vidéo. Les 2 composants centraux représentent les objets $sourceMpeg$ et $decodeurMpeg$ d'une part, et le réseau entre la source et le couple de décodeurs d'autre part.

Comme introduit au paragraphe 2.2, les contraintes de QoS de l'utilisateur sont exprimées par le jeu d'équations $(S0)$ suivant :

$$\forall\ n\ :\ \begin{cases} \tau(a, n) \ = \ n * 0,125\ ms \\ 80\ ms \leq \tau(i, n+1) - \tau(i, n) \leq 120\ ms \\ -40\ ms \leq \tau(Ha_{pa}, n * 800) - \tau(i, n) \ \leq 40\ ms \end{cases}$$

Il s'agit maintenant de déduire de $(S0)$ et du comportement des composants « carte audio » et « serveur X11 » le jeu d'équations $(S1)$ à fournir en entrée de ces composants.

Avec le film utilisé pour les tests, les trames en sortie du décodeur audio ont toutes une taille de 210 octets. Supposons que l'on souhaite consommer au plus 1 024 octets de mémoire dans la carte audio. Il a été montré que [KOF 96] :

$$tailleTampon \ = \ 2 \ * \ gigue \ * \ debitSource$$

où *gigue* constitue la gigue maximale sur une livraison périodique des trames audio dans le tampon de la carte. Donc, la gigue maximale qui devra être autorisée est de $\frac{1\ 024 * 0,125}{2} = 64$ ms. Ainsi, l'équation :

$$\forall\ n\ :\ -32\ ms \leq \tau(Hb_{pb}, n) - \tau(a1, n) \leq 32\ ms \qquad [S1_1]$$

est une solution permettant le respect de la QoS demandée ; Hb_{pb} est une horloge logique de période $pb = 26,25$ ms (avec $26,25 = 0,125 * 210$).

Les équations $[S0_2]$ et $[S0_3]$ trouvent leur correspondance directe dans les équations :

$$\forall\ n\ :\ 80\ ms \leq \tau(i1, n+1) - \tau(i1, n) \leq 120\ ms \qquad [S1_2]$$

$$\forall\ n\ :\ -40\ ms \leq \tau(Ha_{pa}, n * 800) - \tau(i1, n) \ \leq 40\ ms \qquad [S1_3]$$

En effet, la carte audio et le serveur X11 sont ici modélisés, pour simplifier les calculs, par des composants de retard fixe. On sait que $\forall\, n\ : \tau(Ha_{pa}) = \tau(a,n)$; étant donné le jeu $(S0)$ et le comportement temporel du serveur X11 et de la carte audio, $(S1)$ constitue la QoS que devra fournir l'application pour que la QoS utilisateur soit respectée.

Le jeu $(S1)$ est donc :

$$\forall\, n\ : \begin{cases} -32\ ms \leq \tau(Hb_{pb},n) - \tau(a1,n) \leq 32\ ms \\ 80\ ms \leq \tau(i1,n+1) - \tau(i1,n) \leq 120\ ms \\ -40\ ms \leq \tau(Ha_{pa}, n*800) - \tau(i1,n)\ \leq 40\ ms \end{cases}$$

Il nous faut maintenant examiner quelle QoS (jeu $(S2)$) il faut fournir à l'entrée des objets « sourceMpeg/décodeurMpeg » pour que la QoS offerte corresponde au jeu $(S1)$. Nous modélisons les décodeurs comme des composants de retard fixe. Les temps moyens de décompression pour le film de test sont $2,11$ ms pour une trame audio et $55,24$ ms pour une image vidéo ce qui nous donne les retards fixes : $2,11$ ms et $55,24$ ms respectivement. Donc $[S1_1]$ et $[S1_3]$ deviennent :

$$\forall\, n\ :\ -29,89\ ms \leq \tau(Hb_{pb},n) - \tau(a2,n) \leq 34,11\ ms \qquad [S2_1]$$

$$\forall\, n\ :\ 15,24\ ms \leq \tau(Ha_{pa}, n*800) - \tau(i2,n)\ \leq 95,24\ ms \qquad [S2_3]$$

L'équation $[S1_2]$ trouve sa correspondance directe dans l'équation :

$$\forall\, n\ :\ 80\ ms \leq \tau(i2,n+1) - \tau(i2,n) \leq 120\ ms \qquad [S2_2]$$

(car le retard fixe ne joue pas sur le délai entre deux images consécutives).

Le jeu $(S2)$ est donc :

$$\forall\, n\ : \begin{cases} -29,89\ ms \leq \tau(Hb_{pb},n) - \tau(a2,n) \leq 34,11\ ms \\ 80\ ms \leq \tau(i2,n+1) - \tau(i2,n) \leq 120\ ms \\ 15,24\ ms \leq \tau(Ha_{pa}, n*800) - \tau(i2,n)\ \leq 95,24\ ms \end{cases}$$

Enfin, le réseau est modélisé par un composant de retard variable ; $mina$ et $maxa$ désignent les temps minimaux et maximaux que met un échantillon audio pour traverser le réseau et $minv$ et $maxv$ désignent les temps minimaux et maximaux que met une image pour traverser le réseau. L'application est exécutée sur un réseau Ethernet, il n'est donc pas possible de donner a priori des valeurs pour $maxa$, $maxv$, $mina$ et $minv$. Ces quatre informations constituent des variables dans les équations de QoS, dont la valeur sera renseignée par les services de supervision de POLKA. Le jeu $(S3)$ est donc :

$$\forall\, n\ : \begin{cases} -29,89\ ms + maxa \leq \tau(Hb_{pb},n) - \tau(a3,n) \leq 34,11\ ms + mina \\ 80\ ms - maxv + minv \leq \tau(i3,n+1) - \tau(i3,n) \leq 120\ ms + maxv - minv \\ 15,24\ ms + maxv \leq \tau(Ha_{pa}, n*800) - \tau(i3,n) \leq 95,24\ ms + minv \end{cases}$$

C. Annexe : exemple d'une spécification de QoS

```
component retardfixe:
        signals in I, out O;

        qoscontract avechorloge:
                common
                        clock rfc = (,p);
                provided
                        eqos rf1 T(rfc,n) - T(O,n) < epsilon2;
                        eqos rf2 T(rfc,n) - T(O,n) > epsilon1;
                required
                        eqos rf3 T(rfc,n) - T(I,n) < epsilon2 + l;
                        eqos rf4 T(rfc,n) - T(I,n) > epsilon1 + l;
        end
        qoscontract sanshorloge:
                provided
                        eqos rf1 T(O,n+1) - T(O,n) < epsilon2;
                        eqos rf2 T(O,n+1) - T(O,n) > epsilon1;
                required
                        eqos rf3 T(I,n+1) - T(I,n) < epsilon2;
                        eqos rf4 T(I,n+1) - T(I,n) > epsilon1;
        end
end

component decodeur: faudio/retardfixe,fvideo/retardfixe
        signals in a2=faudio.I, in i2=fvideo.I,
                out a1=faudio.O, out i1=fvideo.O;

        qoscontract interflux: faudio.avechorloge,fvideo.sanshorloge
                common
                        clock ha=(,0.125);
                provided
                        eqos inter1 T(ha,n*800) - T(i1,n) < 40 ms;
                        eqos inter2 T(ha,n*800) - T(i1,n) > - 40 ms;
                required
                        eqos inter3 T(ha,n*800) - T(i1,n) < 40 ms + l1;
                        eqos inter4 T(ha,n*800) - T(i1,n) > - 40 ms + l2;
        end
end

component ...
```

```
system demo:

        instance decodeur:
                type applicationComponent twist;
                component decodeur;
                qoscontract interflux hb/faudio.rfc, ha/ha;
                parameters faudio.l=2.11, faudio.p=26.25,
                           l1=2.11, l2=55.24,
                           faudio.epsilon1=-32, faudio.epsilon2=32,
                           fvideo.epsilon1=120, fvideo.epsilon2=80;
                signals a1=md.decodeAudioFrame.RR,
                        a2=ms.putAudioFrame.RR,
                        i1=md.decodeVideoFrame.RR,
                        i2=ms.putVideoFrame.RR;
        end

        instance reseauAudio:
                type networkDevice rock twist;
                component retardvariable;
                qoscontract avechorloge hb/rvc;
                parameters p=26.25,
                           alpha=monitoring@network@delay@min,
                           beta=monitoring@network@delay@max;
                signals O=ms.putAudioFrame.RR,
                        I=ms.putAudioFrame.IE;
        end

        instance ...

end
```