CFSE'1
Rennes, 8 juin - 11 juin 1999
1ère Conférence Française sur les Systèmes d'Exploitation

Modèle et plate-forme pour le support d'applications multimédias réparties

I. Demeure †, L. Leboucher ‡, N. Rivierre ‡ et F. 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 [16]. 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.

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 on utilise, 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 [23], RTP/RTCP [17]). On s'appuie également sur des composants spécifiques tels que les

cartes audio et les décodeurs MPEG.

On rencontre alors plusieurs problèmes : 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 [13]. Ensuite, on ne peut pas se contenter de considérer chaque composant du système isolément : il faut prendre en compte le comportement du système dans sa globalité, ou de bout-en-bout [4].

Par ailleurs, une petite modification dans le comportement temporel de l'application peut nécessiter une grande modification dans la solution développée ; 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) ; enfin, les solutions développées sont peu portables.

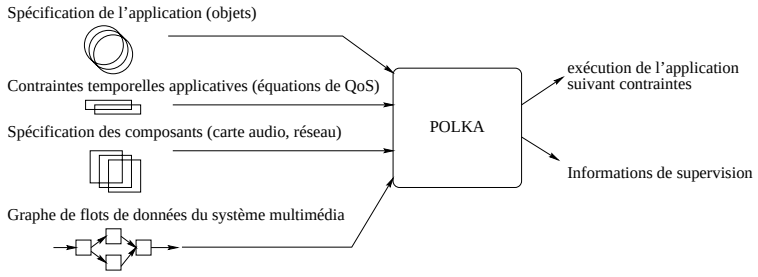

FIG. 1 – *Modèle et plate-forme POLKA*

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 importants de la plate-forme support et le graphe de flots de données qui représente l'ensemble du système multimédia - application et composants de la plate-forme (cf. figure 1).

La plate-forme d'exécution POLKA utilise ces spécifications pour faire **l'ordonnancement automatique** de l'application de façon à respecter les contraintes 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 une application répartie de démonstration qui manipule un flux audio et un flux vidéo qui doivent être synchronisés. L'application est utilisée pour illustrer le modèle POLKA. Dans le chapitre 3, nous décrivons les objectifs et l'architecture de la plate-forme d'exécution. La plate-forme comporte une couche "machine virtuelle" et un bus à objets au standard CORBA. La machine virtuelle a été développée pour augmenter la portabilité de POLKA

et de ses applications. Actuellement, POLKA fonctionne sur Linux et Solaris (un portage sur L4 est en cours). Le chapitre 4 est consacré à une évaluation de l'approche. Nous 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.

2 Modélisation d'un système multimédia

2.1 Modélisation d'une application

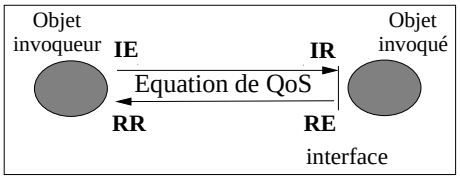

FIG. 2 – *Points d'observation durant une invocation de méthode*

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** [21]. 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.

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 [12]. 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).

En pratique, les objets applicatifs sont des objets CORBA. La figure 4 décrit l'interface IDL des objets *decodeurMpeg* et *sourceMpeg*.

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

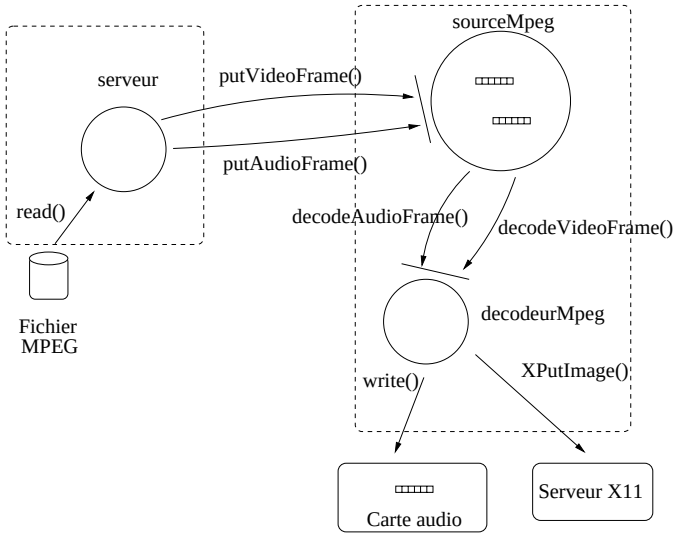

FIG. 3 – *Une application répartie sur POLKA*

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

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

FIG. 4 – *Interface* IDL *de l'application*

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[20] (**QL** pour **Q**oS **L**anguage). Dans cet article, nous nous limitons à des contraintes temporelles déterministes.

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

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

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 $n$ème 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 :

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

où H_p modélise une horloge de période p dont la date d'occurrence du $n$ème 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'équation (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 :

$$\forall\ n\ :\ \tau(e,n) = n * p$$

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.

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 des 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 donc un octet tous les 0.125 ms. Pour assurer une QoS satisfaisante, il faut donc qu'un octet soit restitué toutes les 0.125ms, ce qui est spécifié par l'équation :

$$\forall\ n\ :\ \tau(a,n) = n * 0.125\ ms \quad (S0_1)$$

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 (ie. une image tous 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 :

$$\forall\ n\ :80\ ms \leq \tau(i,n+1) - \tau(i,n) \leq 120\ ms \quad (S0_2)$$

où i désigne l'événement d'affichage d'une image vidéo. Enfin, sachant qu'une image doit être affichée tous 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 :

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

qui peut aussi s'écrire :

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

où $80\ ms = [-40ms, 40ms]$ constitue une gigue acceptable pour l'utilisateur[22].

2.3 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.

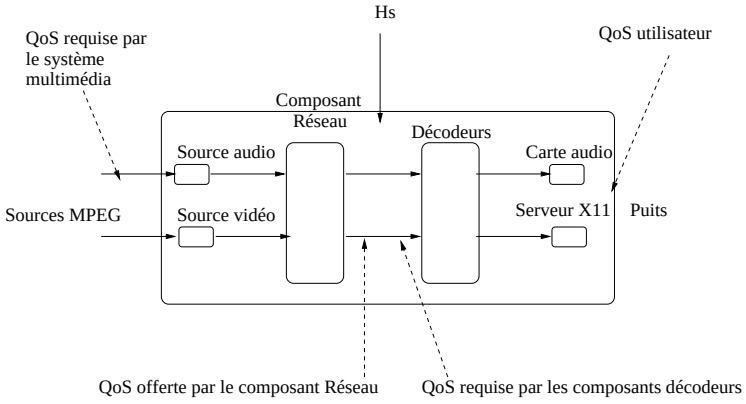

FIG. 5 – *Graphe d'un système multimédia*

Les traitements qui interviennent dans les applications multimédias sont souvent organisés en pipe-line ou en graphes de flots de données [1, 18]. 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 (voir 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'affichages 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, 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 [3]. 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 [14]. 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 : délais moyen de communication) sont spécifiées dans la définition du composant.

Dans le travail présenté ici, nous utilisons les contrats et la composition 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. La QoS requise correspond à une **condition suffisante** 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 spécifié. 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.

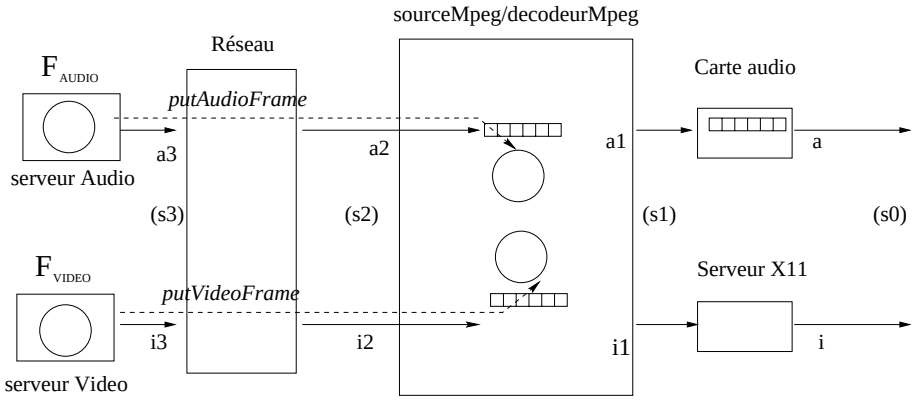

FIG. 6 – *Modélisation de l'application répartie*

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 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. Nous ne pouvons pas donner ici le détail des opérations qui permettent d'obtenir le jeu $(S3)$. Le lecteur intéressé les trouvera dans [5].

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 flot de données est associé à un événement de type IE, IR, RE ou RR. Ceci permet de lier la modélisation de l'application en terme 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

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[15] distribué par le laboratoire d'AT&T à Cambridge) Elle 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

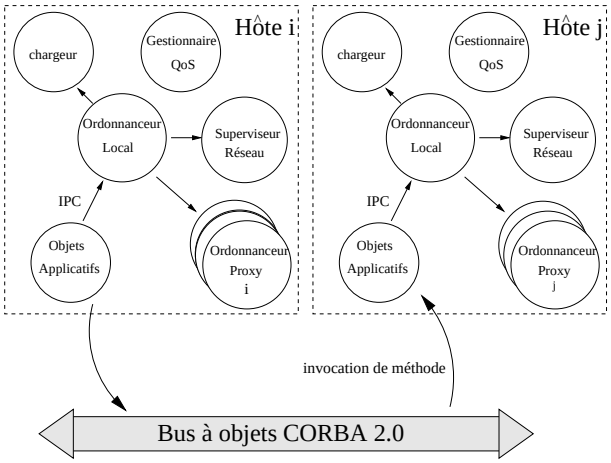

FIG. 7 – *Architecture du prototype*

invocations de méthodes. 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 (voir 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 $(S0)$ 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 utilisés 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 [7, 14]).

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

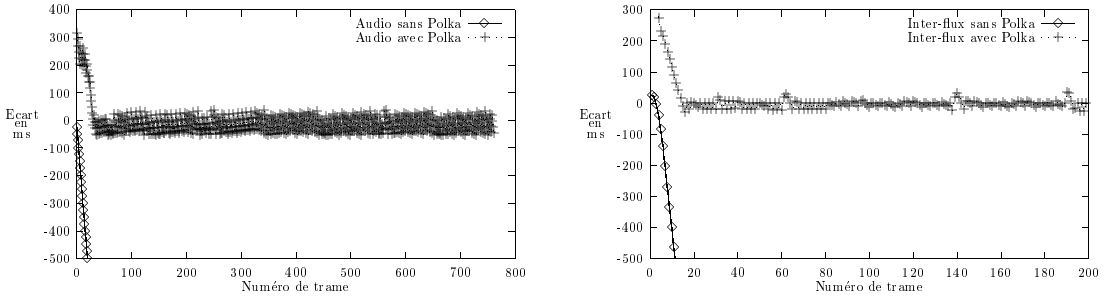

FIG. 8 – *Respect des synchronisations*

communication faite par le superviseur réseau [6].

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* normalisés 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[19]).

En pratique, la machine virtuelle est constituée d'un ensemble de classes C++ *inline* offrant principalement des abstractions de flots d'exécutions, de mécanismes de communication inter-processus, d'outils de synchronisations intra et inter-processus et de temporisateur. Nous en avons développé une version pour Solaris et une version pour Linux. Une implantation sur L4[10] est en cours de réalisation.

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, nous utilisons une application qui simule celle définie dans les paragraphes précédents. La taille des trames audio est de 627 octets, celle des trames vidéo est de 4096 octets. Nous la compilons puis nous l'exécutons 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 gauche 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 courbe de droite de la figure 8 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 d'une méthode CORBA 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 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.

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[8, 9] et d'un système d'exploitation[10, 11] mieux adaptés à notre problématique. Des expériences sur le micronoyau 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[2]. 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.

6 Remerciements

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

Bibliographie

1. D. P. Anderson. Meta-scheduling for distributed continuous media. Technical Report UCB/CSD 90/599, University of California, Berkeley, October 1990.
2. V. Baiceanu, C. Cowan, D. McNamee, C. Pu, and J. Walpole. Multimedia Applications Require Adaptive CPU Scheduling. Workshop on Resource Allocation Problems in Multimedia Systems, Washington DC, December 1996.
3. G. Blair and J. B. Stefani. *Open Distributed Processing and Multimedia.* Addison-Wesley, 1998.
4. A. Campbell C., Aurrecoechea, and L. Hauw. A Review of Quality of Service Archtectures. pages 173–194. Université de Columbia, in Proceedings of the 4th international IFIP Workshop on Quality of Service, Paris , March 1996.
5. I. Demeure and F. Singhoff. Modélisation de systèmes multimédias : application à l'ordonnanceur POLKA. à paraître.
6. I. Demeure and F. Singhoff. Environnement d'exécution pour les applications réparties sous contraintes temporelles : une solution CORBA-RTP. pages 53–57. 10 ème Rencontres Francophones du Parallélisme (RENPAR'10) - Strasbourg, juin 1998.
7. I. Demeure and F. Singhoff. Spécification et ordonnancement dynamique d'applications multimédias : l'environnement Polka . pages 101–117. Real time systems'98, Paris la Défense, janvier 1998.
8. B. Dumant, F. Horn, F. Dang-tran, and J. B. Stefani. Jonathan : an Open Distributed Processing Environment in Java. pages 173–190. MIDDLEWARE'98. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing, 1998.
9. R. M. Edmundo H. S. Flavio. ODP-based QoS Specification for the Multiware Platform. pages 45–53. in Proceedings of the 4th international IFIP Workshop on Quality of Service, Paris , March 1996.
10. H. Hartig, M. Hohmuth, J. Liedtke, S. Schonberg, and J. Wolter. The Performance of micro-Kernel-Based Systems. 16th ACM Symposium on operating Systems Principles in Saint Malo (SOSP'97) - France, October 1997.
11. H. Härtig, R. Baumgartl, M. Borriss, Cl.-J. Hamann, M. Hohmuth, F. Mehnert, L. Reuther, S. Schönberg, and J. Wolter. DROPS - OS Support for Distributed Multimedia Applications. In the proceedings of the Eigth ACM SIGOPS European Workshop, Sintra, Portugal, September 1998.
12. ISO. Press Release, 29th Meeting of JTC 1/SC 29/WG 11. number 1110, March 1995.
13. Nieh J., Northcutt J.N., and Hanko J.G. SVR4 UNIX Scheduler unacceptable for multimedia applications. In *Proceedings of the 4th International Workshop on NOSDAV*, pages 49–60, Lancaster, U.K., November 1993. Lecture Notes in Computer Science, Vol846, Springler-Verlag.
14. L. Leboucher. *Algorithmique et Modélisation pour la Qualité de Service des Systèmes Répartis Temps Réel.* Thèse de doctorat, Ecole Nationale Supérieure des Télécommunications, septembre 1998.
15. S. L. Lo and S. Pope. 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.
16. OMG TC Document 98-07-01. The Common Object Request Broker : Architecture and Specification. Revision 2.2. February 1998.
17. H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC1889 : RTP : A Transport Protocol for Real-Time Applications. Network Working Group, pages 1-75, January 1996.
18. S. Siewert, G. Nutt, and M. Humphrey. Real-Time Parametrically Controlled In-Kernel Pipelines. Third IEEE Real Time Technology and Application Symposium (RTAS'97), Work-In-Progress, Montreal - Canada, June 1997.
19. B. Srinivasan, S. Pather, R. Hill, F. Ansari, and D. Niehaus. A Firm Real Time System Implementation using Commercial Off-The-Self Hardware and Free Software. 4th IEEE Symposium on Real-time Technology and Applications, Denver Colorado, June 1998.
20. J. B. Stefani. Computational Aspects of QoS in an object-based, distributed systems architecture. 3rd Workshop on Responsive Computer systems, Lincoln, NH, USA, September 1993.
21. J.B. Stefani, G.S. Blair, G. Coulson, M. Papathomas, P. Robin, F. Horn, and L. Hazard. 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.
22. R. Steinmetz and K. Nahrstedt. *Multimedia : Computing, communicating and applications.* Prentice Hall, innovative technology series, 1995.
23. Ronald J. Vetter. ATM concepts, architectures, and protocols. *Communications of the ACM*, 38(2):30–38, feb 1995.