Extensions multimédia de la messagerie MMS

Max Roger Pokam

Extensions Multimédia de la Messagerie MMS

Max R. Pokam

13 septembre 1995

THESE

présentée par

Max Roger POKAM

pour obtenir le titre de DOCTEUR

de L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(arrêté ministériel du 30 mars 1992)

(Spécialité : **Informatique**)

EXTENSIONS MULTIMÉDIA DE LA MESSAGERIE MMS

Date de soutenance : 12 septembre 1995

Composition du jury :

Messieurs Jacques MOSSIÈRE Président

Jean Pierre ELLOY Rapporteur

Francis LEPAGE Rapporteur

Pascal SICARD Examinateur

Gérard MICHEL Directeur

Thèse préparée au sein du Laboratoire de Génie Informatique.

TABLE DES MATIÈRES

Table des figures

Chapitre 1

Introduction : Automatisation et Multimédia

1.1 Des progrès technologiques favorables

Les progrès technologiques confirmés au cours de cette décennie favorisent une importante révolution dans la façon de concevoir les systèmes informatiques. Les nouveaux processeurs sont en effet de plus en plus rapides. Et la transmission optique permet d'atteindre des débits de plus en plus élevés. Ces progrès favorisent la miniaturisation des équipements et l'intégration de l'information sous forme de média continus réputés traditionnellement comme étant gourmands en bande passante de traitement et de transmission.

Les fréquences annoncées des nouveaux processeurs sont de plus en plus importantes. En effet, les performances de microprocesseurs à faible coût sont en passe de défier celles des super-calculateurs d'il y a quelques années. Le temps de cycle des processeurs pourra bientôt passer en dessous du seuil de la nanoseconde, offrant ainsi des puissances que l'on exprimera en milliards d'instructions par seconde. Avec de telles performances, de nombreux services, jusqu'alors considérés gros consommateurs de calcul et réalisables seulement sur des super calculateurs ou des stations de travail graphiques super puissantes (reconnaissance vocale, graphique temps-réel de haute qualité, reconnaissance de l'écriture manuelle, animation et vidéo temps-réel, etc.) seront accessibles sur un ordinateur personnel moyen.

Simultanément à cette évolution des processeurs, on assiste à une amélioration spectaculaire de la capacité de transmission des réseaux. Il y a peu de temps de cela, un réseau local à 10 Mbit/s ou une ligne MIC à 2 Mbit/s étaient considérés comme étant rapides. Les réseaux locaux et les réseaux grandes distances d'aujoud'hui atteignent des débits de l'ordre de quelques milliards de bits par seconde. Cette avancée est l'aboutissement de deux décennies de travaux sur la transmission optique.

Des réseaux offrant des débits de milliards de bits par seconde permettront d'interconnecter des ordinateurs personnels, moyens ultra-performants exécutant de nouvelles applications pouvant traiter, émettre ou recevoir des dizaines de mégabits de données par seconde. Ces considérations ont conduit à la définition du projet IMAG **RACINES** dont l'objectif général est de déterminer l'impact des communications rapides, et en particulier l'ATM, sur les architectures de systèmes informatiques, qu'ils soient distribués,

parallèles ou monolithiques. C'est dans le cadre de ce projet mené au Laboratoire de Génie Informatique (**LGI**) de l'**IMAG** que notre étude s'est déroulée.

1.2 La richesse du multimédia

Les systèmes automatisés sont des exemples de systèmes informatiques qui connaîtront l'influence des progrès technologiques. La richesse de l'information multimédia constituée de média continus et discrets facilitera et changera le travail des opérateurs et techniciens d'atelier du système automatisé du futur.

Grâce à ses possibilités de communication audiovisuelle, l'information multimédia permettra une communication interpersonnelle plus efficace entre l'opérateur et les techniciens d'atelier du site du procédé automatisé. L'opérateur pourra ainsi communiquer aux techniciens les étapes d'une opération de maintenance en cours, et vérifier le bon respect des mesures de sécurité par ces derniers.

Le multimédia permet également de véhiculer des informations plus réalistes sur le procédé physique automatisé. De ces informations réalistes découleront une prise de décision plus informée, des opérations plus sauves, plus économiques et plus promptes, et une meilleure qualité des produits. Ces dernières années ont vu l'intégration de la vision artificielle dans certaines applications de productique et d'inspection de qualité. Une utilisation plus répandue de capteurs visuels miniaturisés dans les systèmes automatisés est à prévoir. L'addition de l'information visuelle, aussi bien en flots temps-réel qu'en données stockées, permettra une meilleure compréhension et un meilleur suivi du procédé, une analyse plus informée, une prise de décision plus rapide et plus avisée. Une séquence de vidéo stockée pourrait être utilisée par exemple pour :

1. une analyse en ligne de procédés afin de mieux planifier les opérations,
2. permettre une étude de l'évolution de la qualité des produits dans le temps,
3. réviser les situations d'alarme, en déterminer les causes et prendre des mesures préventives,
4. superviser les activités et les méthodes de travail des techniciens,
5. vérifier le respect des mesures de sécurité,

6. et inspecter les équipements.

Actuellement, la majorité des systèmes automatisés traitent principalement des données numériques provenant de capteurs, de contrôleurs logiques programmables et d'équipements de supervision du site. Ces valeurs sont souvent combinées avec des données alphanumériques et graphiques, et sont représentées sur un écran d'affichage en guise de moyen de supervision, de compréhension et de contrôle. Cependant, le texte et les graphiques ne représentent qu'une infime partie de la réalité du procédé. Ces capteurs sont incapables de saisir toute la réalité du procédé. Il est encore très fréquent que les opérateurs quittent la salle de contrôle pour aller voir ce qui se passe dans les ateliers, ou qu'ils se servent de la communication vocale avec les techniciens d'ateliers pour recevoir des descriptions verbales et subjectives. L'information vidéo temps-réel ou stockée nous semble être d'un grand intérêt pour les systèmes automatisés.

Qui plus est, l'information "visuelle" ne sera pas limitée uniquement au spectre visible. Les ondes infrarouges, les rayons X, et autres techniques d'imagerie seront utilisés pour capter des informations inexploitables autrement. Les infrarouges seront utilisés par exemple pour détecter les phases préliminaires de corrosion du matériau d'un outil qui serait indécelable par l'oeil humain. L'imagerie ultrasonore ou aux rayons X pourrait, dans un même ordre d'idée, être utilisée comme technique d'inspection non-invasive pour certains types de produits. Ces nouveaux types d'information sont présentés sous forme de média continus et peuvent être intégrés, eux aussi, à la faveur des progrès technologiques.

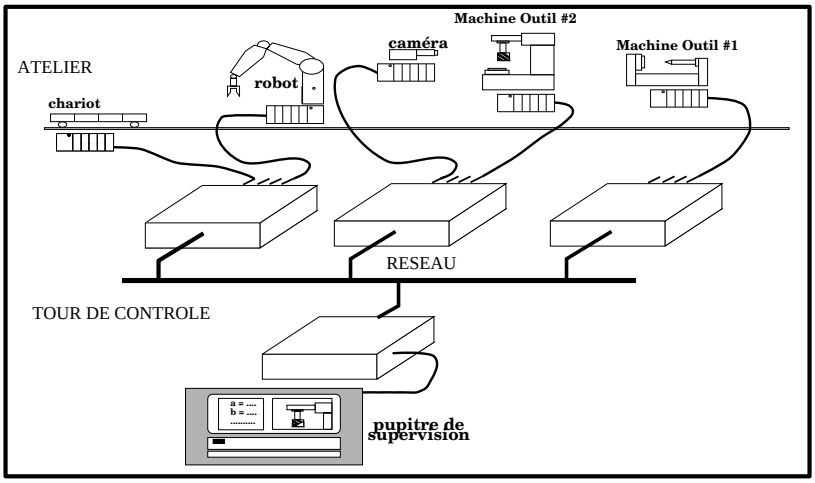


Figure 1.1 : Une configuration de système automatisé intégrant un médium continu d'information vidéo

La figure 1.1 illustre un exemple de système automatisé dans lequel la saisie de la scène

de la machine outil d'une chaîne d'usinage d'un atelier flexible est réalisée au moyen d'une caméra et d'une carte d'acquisition. Un enregistrement d'une durée de six minutes est remis à jour périodiquement toutes les dix minutes.

À l'occurrence d'un événement correspondant au bourrage de la chaîne, la source vidéo de la carte d'acquisition est directement reliée à une fenêtre du pupitre de supervision de la tour de contrôle.

L'opérateur constate une détérioration de l'outil et peut alors consulter les six dernières minutes de la scène de la machine outil qui on été enregistées. L'opérateur n'est pas obligé de se déplacer ou de faire appel à des techniciens de terrain pour analyser la situation et prendre des décisions ; ceci grâce à l'information vidéo véhiculée jusqu'à son pupitre de supervision.

Le rôle du multimédia dans les systèmes automatisés ayant ainsi été illustré, il est impératif de définir les étapes de l'évolution des systèmes automatisés actuels à ceux du futur intégrant des média continus en plus des média discrets conventionnels. C'est en réponse à cette préoccupation que nous proposons des extensions multimédia à la norme internationale de communication des systèmes automatisés, MMS.

1.3 Extensions de la norme MMS

Quand MMS (Manufacturing Message Specification) a été retenue comme norme de communication de systèmes automatisés ouverts, sa priorité était d'offrir une plate-forme indépendante et ouverte pour le développement d'applications distribuées temps-réel. Un environnement de communication basé sur la pile de protocoles OSI, des objets et des services de manipulation de ces derniers, et un protocole client-serveur basé sur des messages ont ainsi été définis. Il en résulte une norme dans laquelle la communication est basée sur les messages et qui est mal adaptée aux flots soutenus de média continus.

Compte tenu des avantages offerts par une intégration de média continus aux systèmes automatisés, une étude des EXTENSIONS MULTIMÉDIA À LA NORME MMS est d'un intérêt certain.

1.4 Articulations de la réflexion

Le chapitre 2 est une présentation des particularités des systèmes multimédia traitant et véhiculant l'information sous forme de média discrets et continus. Compte tenu de la prolifération des termes utilisés dans la littérature du multimédia, nous précisons la terminologie que nous utiliserons. Ensuite nous présentons la notion de la qualité de service des média continus et décrivons les défis que le respect de cette qualité de service impose aux systèmes d'exploitation d'une part, et aux systèmes de communication d'autre part. Nous abordons à la fin de ce chapitre l'état de l'art sur les modèles de programmation de plate-formes multimédia distribuées.

Le chapitre 3 est une présentation de la norme de messagerie industrielle MMS, avec une mise en évidence des aspects qui la rendent inapte au traitement et au transfert de flots de média continus. Les points que nous traitons sont : la propriété de MMS comme solution ouverte de communication pour systèmes automatisés, son environnement de communication, son modèle objet, et son modèle client-serveur.

Dans le chapitre 4 nous proposons de remédier aux limites multimédia de MMS en définissant un nouvel environnement de communication, de nouvelles classes d'objets et de nouveaux services. Ces extensions répondent aux besoins des flots continus.

Le chapitre 5 est ensuite consacré à la présentation d'un prototype d'implémentation des nouveaux objets et services, en vue de disposer d'une plate-forme de mesure de performances.

Une conclusion présente les principaux résultats et les prolongements possibles de cette étude.

Chapitre 2

Les Particularités des Média Continus

Ce chapitre est consacré à la présentation des particularités des systèmes multimédia traitant et véhiculant l'information sous forme de média discrets et continus. Compte tenu de la prolifération des termes utilisés dans la littérature du multimédia, nous précisons d'abord la terminologie employée. Ensuite, nous présentons la notion de la qualité de service des média continus et décrivons les défis que le respect de cette dernière impose aux systèmes d'exploitation d'une part, et les systèmes de communication d'autre part. Nous terminons le chapitre par l'examen de quelques modèles de programmation de plate-formes multimédia distribuées.

2.1 La terminologie

Un médium : Un médium est une forme de support de l'information. C'est la représentation informatique de l'information à traiter ou à transmettre.

Un médium discret : Un médium discret est un support constitué d'une seule instance d'une forme de représentation informatique de l'information. Une variable, un fichier, un message, un paquet, un graphique, une unité de données protocolaire, et un texte sont des exemples de média discrets.

Un médium continu : Un médium continu est un support constitué d'une suite **finie** ou **infinie** de plusieurs instances d'une forme de représentation informatique de l'information ayant entre elles une relation de causalité et de synchronisme. Un médium continu peut aussi être défini comme une séquence de média discrets de même type ayant entre eux une relation de causalité et de synchronisme. La séquence des paquets de l'information sonore et la séquence d'images de l'information vidéo sont des exemples de média continus.

Le multimédia : Le multimédia est une combinaison des types de média discret et continu.

Un système distribué multimédia : Un système distribué multimédia est tout système distribué de traitement d'informations sous forme de média continus (voix, audio, images animées, et vidéo) et de média discrets (messages, variables, textes, fichiers, ...).

2.2 La qualité de service

Comme nous l'avons indiqué au chapitre d'introduction, les acquis technologiques en rapidité de traitement et de transfert des média continus favorisent le développement de systèmes distribués multimédia. L'information contenue dans les média continus a la particularité d'être synchrone (8000 échantillons du signal vocal par seconde, 25 images par seconde pour la vidéo). Il en résulte que tout élément du médium continu retardé ou erroné au cours du traitement ou du transfert causera un désagrément chez l'utilisateur (son ou vidéo saccadés, désynchronisation des composantes audio et vidéo). Les systèmes de traitement et de transfert étant distribués et asynchrones, il faut définir des artifices pour répondre aux besoins de l'information sous forme de média continus. Ce problème concerne toutes les composantes du système distribué, allant des systèmes d'exploitation, aux interfaces et systèmes de communication en passant par les interfaces de programmation offertes aux applications.

2.3 Les systèmes d'exploitation

Les contraintes de qualité de service de l'information sous forme de média continus sur les systèmes d'exploitation sont liées au comportement temps-réel de ces derniers. Les points critiques du comportement temps-réel d'un système d'exploitation comprennent son temps de changement de contexte, sa politique d'ordonnancement, et sa politique de réservation des ressources[1].

Les systèmes UNIX standard ne sont pas, en général, capables de satisfaire ces exigences. Nieh et al. ont examiné à titre d'exemple l'ordonnanceur du système SVR4UNIX [31]. Ils le trouvent inacceptable pour les informations sous forme de média continus. Des investigations similaires ont été effectuées dans le cadre du système d'exploitation Mach [44]. Elles elles aboutissent à la conclusion suivante : "pour les systèmes temps-réel et multimédia limités par des performances au pire cas, le système Mach impose une trop grande surcharge".

Les approches de solutions proposées pour les défauts d'UNIX consistent en l'extension des systèmes d'exploitation existants ou à une ré-implémentation des systèmes UNIX. Dans cet ordre d'idée, Fisher[14] a présenté des extensions au noyau de l'Ultrix-4.2 in-

[1] Herrtwig [45] a donné un aperçu des exigences temps-réel imposées aux systèmes d'exploitation pour satisfaire les exigences de l'information sous forme de média continus.

cluant des **processus temps-réel**, des **points de préemption**, des **sémaphores** sur des structures de données, un **mécanisme d'héritage de priorité**, et un **traitement interne des événements**. Des mesures de performance effectuées sur ce noyau montrent son adéquation aux applications multimédia. Un deuxième exemple est l'extension temps-réel du système AIX de IBM. Son adéquation aux applications multimédia a été étudiée par Nahrstedt et Smith [35]. Ils trouvent que les extensions sont seulement partiellement adéquates aux exigences de l'information sous forme de média continus.

En ce qui concerne le système Mach, des améliorations sont obtenues par le Real-Time Mach [20] qui "étend Mach 3.0 par des processus légers temps-réel et par un ordonnanceur qui devraient améliorer considérablement les performances d'applications à faible latence". Une stratégie de réservation du processeur pour l'architecture à micro-noyau de Mach est présentée par Nakajima et al. [54]. Il s'agit d'un modèle de serveur temps-réel pour des services déterministes. Le serveur est implémenté et évalué dans l'environnement RT Mach. Une autre étude basée sur le micro-noyau CHORUS a été présentée par Coulson et Blair [15]. Les caractéristiques temps-réel de CHORUS permettraient la garantie des qualités de service de l'information sous forme de média continus.

Un exemple de systèmes d'exploitation hors du monde UNIX est OS/2. Parsons [32] rapporte sur une architecture multimédia basée sur le système OS/2. Il trouve que les propriétés temps-réel du système OS/2 sont suffisantes pour les applications multimédia.

Notons également qu'il existe un cadre de normalisation de systèmes d'exploitation POSIX (*Portable Operating System Interface, et X comme dans UNIX*) ou IEEE 1003. Ces travaux ont commencé en 1986 sous forme de symposia animés par General Motors et ont progressivement donné naissance à des normes ANSI et OSI développées par l'IEEE. POSIX fait le pont entre les applications et les systèmes d'exploitation. La section POSIX 1003.1b de la norme définit les extensions temps-réel d'un système d'exploitation portable. Il s'agit : 1) de sémaphores binaires, 2) de clés sur mémoires de processus, 3) de fichiers étendus en mémoire, 4) de mémoires partagées, 5) d'un ordonnancement par priorité, 6) de signaux temps-réel, 7) d'horloges et de temporisateurs, 8) du passage de message, 9) d'entrées-sorties synchrones, 10) et d'entrées/sorties asynchrones. Une autre section, la section POSIX 1003.1c définit l'extension au parallélisme supporté par des processus légers. Les extensions proposées dans le cadre de travail de POSIX pour les systèmes d'exploitation temps-réel sont au départ destinées aux applications critiques de l'avionique, de l'armement ou de contrôle de procédé. Les exigences de l'information sous forme de média continus n'étant pas d'égale criticité, les extensions POSIX 1003.1b et 1003.1c peuvent jouer un rôle intéressant dans le traitement et le transfert de média

continus.

2.4 Les systèmes de communication

Les systèmes distribués temps-réel nécessitent des sous-systèmes de communication capables de garantir les exigences de qualité de service du type d'information échangée. Dans ce paragraphe nous considérons les aspects de la qualité de service aux trois niveaux de la hiérarchie des protocoles de communication (figure 2.1).

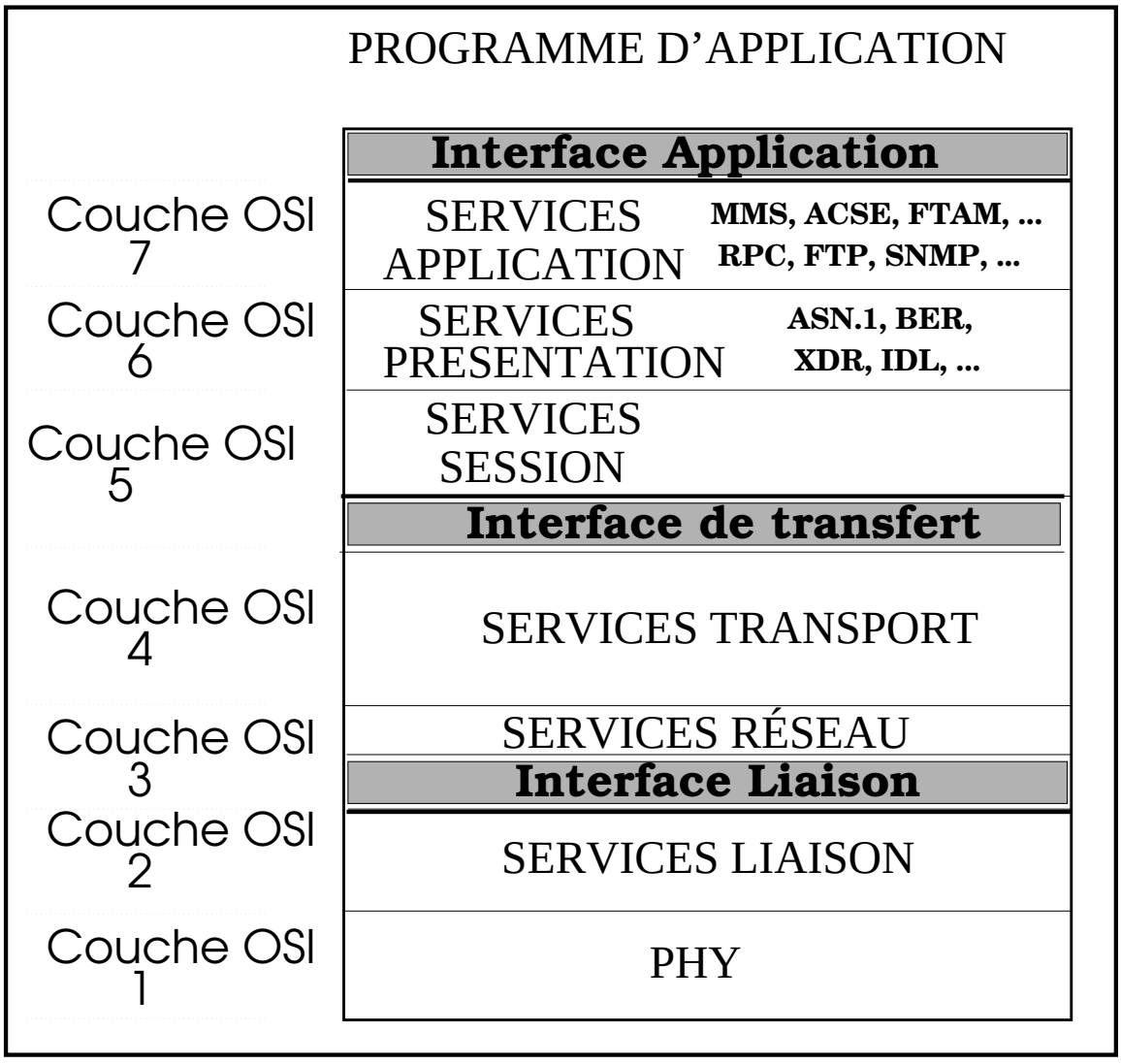


Figure 2.1 : Les trois niveaux de service du modèle de référence OSI

Les protocoles de **couches basses** sont ceux situés en dessous de la couche réseau

du modèle de référence OSI. Il s'agit de la couche de liaison de données et de la couche physique. En particulier, FDDI, ATM, et des versions de *Ethernet* haut-débit et synchrones présentent des caractéristiques attrayantes. Les protocoles de **transfert** correspondent aux couches réseau et de transport du modèle OSI. Les questions intéressantes que l'on se pose à ce niveau sont quels paramètres de qualité de service définir, et comment les garantir, c'est-à-dire, comment les ressources peuvent-elles être réservées et comment les paramètres peuvent-ils être mis en correspondance avec ceux des niveaux inférieurs. Les protocoles des **couches supérieures** concernent les protocoles des couches application, présentation et session du modèle OSI. Les protocoles à ce niveau doivent offrir des possibilités évoluées de négociation de la qualité de service entre l'application et les composantes de communication.

2.4.1 Les couches basses

Des techniques de compression des média continus normalisées permettent de réduire considérablement le débit requis pour leur transfert. On peut citer : a)- JPEG (ITU T.80 - ISO JTC1/SC2/WG10) pour les images fixes, b)- MPEG (ISO JTC1/SC2/WG11) pour l'audio et la vidéo, c)- H.221 (ITU séries H) pour des structures de trame pour un canal de 64 à 1920 kbit/s dans les télé-services audiovisuels, d)- H.242 (ITU séries H) pour la communication audiovisuelle sur des canaux pouvant atteindre 2Mbit/s, e)- H.261 (ITU séries H) pour codec de services audiovisuels à p*64 kbit/s.

Les progrès accomplis en électronique des transmissions électromagnétiques et optiques, permettent d'obtenir des débits de 100 Mbit/s (100Base-T) sur du cuivre, et des débits de dizaines de Gbit/s sur de la fibre optique. La bande passante disponible étant de plus en plus importante, cela n'empêche pas pour autant de définir des algorithmes efficaces de résolution de conflits d'accès concurrents.

Le protocole MAC CSMA/CD (IEEE 802.3 10Base-T et IEEE 802.3 100Base-T) ne permet pas de garantie de délai d'accès au médium à cause du non-déterminisme inhérent à son algorithme de résolution de conflits. Des travaux effectués dans le cadre du projet REFLECS à l'INRIA ont permis la mise au point du protocole CSMA/DCR [18] qui permet de garantir un délai d'accès aux stations connectées sur un médium Ethernet.

Dans le cadre du projet MAP[55], la norme de bus à jeton IEEE 802.4 permettant de gérer le médium physique comme un bus informatique a été définie comme norme MAC de la pile de protocole allégée MiniMAP, afin de répondre aux contraintes de temps de la

cellule de production industrielle du modèle CIM (Computer Integrated Manufacturing).

Compte tenu des contraintes de communication imposées par les nouveaux types d'application, certains constructeurs de systèmes contournent le problème de conflit d'Ethernet en construisant un commutateur central (*hub*) auquel toutes les stations du réseau local sont connectées par une infrastructure Ethernet point-à-point. Dans ce cas les 10Mbit/s du 10Base-T, ou les 100 Mbit/s du 100Base-T sont entièrement disponibles pour la communication entre le commutateur et la station. Cette solution peut garantir le temps d'accès aux stations mais elle est beaucoup plus vulnérable (cas de panne du commutateur).

Les technologies à *anneau à jeton* telles que Token Ring et FDDI[33] peuvent, en accord avec leur politique de contrôle de jeton particulière, borner le délai maximal et réserver des ressources pour le débit garanti. Token Ring utilise une connectique Ethernet (IEEE 802.5 10Base-T et 802.5 100Base-T). Il définit 8 niveaux de priorité et permet de garantir des débits et des délais d'accès bornés. FDDI utilise pour sa part un médium optique à 100 Mbit/s. Il supporte deux types de trafic : le trafic synchrone et le trafic asynchrone. Seul le trafic asynchrone est largement utilisé aujourd'hui. Le trafic asynchrone utilise la bande passante excédante de l'utilisation du trafic synchrone et permet 8 niveaux de priorité. Le trafic synchrone de FDDI permet des débits garantis et des délais bornés.

D'autres normes de réseaux locaux destinées à supporter les trafics temps-réel ont été définies ; il s'agit de ISO ETHERNET (IEEE 802.9) et de 100VG-AnyLAN (IEEE 802.12). L'idée de l'Iso Ethernet est de multiplexer sur un même médium physique un canal Ethernet à 10 Mbit/s et un canal isochrone à 6.144 Mbit/s. Les caractéristiques principales de l'Iso Ethernet sont : une compatibilité avec la connectique du 10Base-T, et la compatibilité à la signalisation ISDN Q.93X pour le canal synchrone. La norme 100VG-AnyLAN utilise pour sa part, un répéteur qui scrute les stations pour les requêtes de transmission. Il y a deux niveaux de priorité, et les transmissions sont autorisées selon une politique *round robbin* dans chaque classe de priorité. Les requêtes de priorité supérieure suspendent le service des requêtes de priorité inférieure. Les répéteurs peuvent être mis en cascade sous forme d'arborescence ; l'ensemble constitué de ces répéteurs se comporte alors comme un plus grand répéteur.

Le mode de transfert asynchrone ATM [10] offre des facilités explicites pour le traitement de la qualité de service dans le cadre de son protocole de signalisation. À cette fin, les messages SETUP et CONNECT contiennent des éléments d'information sur le délai de bout-en-bout, la gigue de délai et le débit de transfert de cellules de l'utilisateur.

Damaskos et Gavras [51] ont étudié la mise en correspondance des paramètres de qualité de service de la couche de transport à l'ATM. La technologie ATM offre des services de niveau réseau d'après le modèle de référence OSI. Nous la traitons dans le contexte des protocoles de couches basses parce que la tendance actuelle des constructeurs est de la situer au niveau 2 du modèle de référence OSI. La connectique utilisée par ATM peut être aussi bien à base du cuivre que de la fibre optique. Les interfaces optiques SDH et SONET qui domineront sur les interfaces de cuivre permettent des plages de débits ayant une croissance incrémentale de 51,84 à 2488,32 Mbit/s pour l'interface SONET et de 155,52 à 2488,32 Mbit/s pour l'interface SDH. Un bon aperçu sur les protocoles de bas niveau peut se trouver dans le chapitre deux du livre de Partridge[41].

2.4.2 Les couches de transfert

Les exigences imposées aux services de transfert pendant l'ère "pré-média continus" ont été surtout axées sur le contrôle de flux et la livraison sûre et non corrompue des données. Ces exigences ont été largement satisfaites par TCP/IP et les protocoles de transport de l'OSI. Cependant, les média continus synchrones ont des exigences de communication totalement différentes. Les données doivent être transmises et livrées comme des flots continus et soutenus, les irrégularités dans le flot d'éléments discrets constitutifs du médium continu pourraient causer une dégradation de la qualité de l'audio ou de la vidéo. Les applications interactives (entre utilisateurs) sont hautement sensibles aux délais. D'un autre côté, une certaine proportion de pertes et d'erreurs est acceptable dans la plupart des cas (les utilisateurs peuvent tolérer une dégradation passagère de la qualité de l'information reçue). Sebuktekin [52] a mené une étude sur 9 protocoles de transport [2] candidats au support d'applications haut-débit et temps-réel comme celles traitant des média continus de support d'information. Il compare lesdits protocoles selon six critères : 1) la parallélisation et l'implantation matérielle des opérations de protocole, 2) l'utilisation de stratégies de contrôle de flux appropriées, 3) l'utilisation de stratégies de récupération d'erreurs rapides et efficaces, 4) l'indépendance du débit par rapport aux variations de délai, 5) l'offre d'un chemin de traitement simple, minimal et vérifié, et 6) l'extraction aisée des informations de protocole (en-tête et queue) des paquets. Il ressort de cette étude, que les quatre meilleurs protocoles de transport, à quelques différences près, sont dans

[2] 1) Delta-t, 2) Datakit Universal Receiver Protocol (Datakit URP), 3) Transmission Control Protocol (TCP), 4) Versatile Message Transaction Protocol (VTMP), 5) Open Systems Interconnection Transport Protocol Class 4 (OSI/TP4), 6) Network Block transfer Protocol (NETBLT), 7) Express Transfer Protocol (XTP), 8) Sabnani, Netravali, and Roome's Protocol (SNR, et 9) ATM Adaptation Layer Convergence Protocol 3/4 (AAL 3/4).

l'ordre : XTP, NETBLT, SNR et AAL 3/4. Ces résultats sont à titre indicatif, étant donnés les critères subjectifs d'affectation de points utilisés par l'auteur.

Les paramètres de qualité de service pour les protocoles de transport multimédia sont : 1) la taille maximale de l'unité de données de service de transport (TSDU), 2) l'intervalle minimal d'inter-arrivée de ces unités de données, 3) le délai de bout-en-bout, et 4) la gigue ou la variation de ce délai. Afin de satisfaire certaines valeurs prédéterminées de ces paramètres de qualité de service, une réservation des ressources est indispensable. Le protocole de réseau utilise les services du protocole de liaison de données pour accéder au médium de transmission physique. Pour pouvoir garantir les exigences de délai de bout-en-bout, de gigue de délai, et de débit des flots synchrones de média continus, il ne suffit pas de garantir les débits et les délais d'accès au niveau liaison de données. Des ressources de bande passante de traitement, d'espace d'emmagasinage et de bande passante de transmission doivent être réservées au niveau de chaque noeud du chemin allant de la source à la destination du flot [3]. Une solution pour ce maintien d'informations de routage des paquets appartenant à un flot est le mode connecté dans les systèmes à commutation de paquets. Il faut créer une connexion virtuelle entre le noeud source et chaque noeud destination du flot. La connexion virtuelle porte des attributs de qualité de service liés au flot qu'elle véhicule. La technologie ATM, qui est un routage de paquets de taille fixe et réduite, est essentiellement basée sur ce concept de connexion virtuelle.

Les caractéristiques de la connexion virtuelle sont liées au profil du trafic de l'application. On considère aussi bien les trafics aléatoires que les trafics déterministes. Un trafic aléatoire est un trafic pour lequel la taille des paquets et l'intervalle d'inter-arrivée des paquets peuvent avoir une variation aléatoire. Ce type de trafic a été introduit dans l'espoir de tirer profit du multiplexage d'un grand nombre de flots pour une utilisation optimale des ressources des noeuds. Le trafic déterministe, quant à lui, ne possède pas de paramètres statistiques ; il décrit l'état le plus contraignant dans lequel pourrait se trouver le trafic. Le trafic déterministe est caractérisé par une taille maximale de paquet et un intervalle d'inter-arrivée minimal.

Diverses approches ont été développées afin de répondre aux besoins de garantie de qualités de service au niveau de transfert. Anderson, Herrwich, et Schaefer ont présenté un protocole de réservation de ressources pour la communication à performances garanties dans un système distribué basé sur IP. Le protocole s'appelle Session Reservation Protocol (SRP) [6]. Il permet de réserver des ressources telles que les bandes passantes CPU et

[3] Notons néanmoins que Shenker [53] et Le Lann [18] proposent l'utilisation des hypothèses de la théorie des jeux pour résoudre le problème d'allocation de ressources de communication temps-réel.

réseau, afin d'atteindre des délais et des débits déterminés.

L'approche du groupe Tenet [7] offre un ensemble de schémas et de protocoles pour la communication temps-réel. Les paramètres de qualité de service supportés sont les bornes sur le délai, la gigue sur la probabilité de violation de délai et sur la saturation de tampons d'emmagasinage. La pile de protocoles du groupe Tenet comprend : le *Realtime Channel Administration Protocol (RCAP)* qui établit le canal et réserve les ressources requises, le *Realtime Internet Protocol (RIP)* qui ordonnance les paquets en accord avec les ressources réservées, le *Realtime Message Transport Protocol (RMTP)* qui supporte un protocole de transport à base de messages entre points terminaux, et le *Continuous Media Transport Protocol (CMTP)* qui offre une interface à base de flots pour les applications isochrones. Une extension de l'approche du groupe Tenet est présentée par Parris, Ventre. et Zhang. Ils introduisent des approches de dégradation progressive (*Graceful Degradation Schemes (GDS)*) [2]. Ces nouveaux schémas permettent l'adaptation de nouveaux paramètres de qualité de service pendant la durée de vie d'une connexion établie. L'adaptation peut être initiée par le client ou par le réseau.

Le protocole expérimental *Stream Protocol Version 2 (ST-II)* [3] de Internet est un protocole de réseau offrant des services de communication point-à-multipoint. Il offre des facilités de négociation et de réservation des ressources pour des paquets de taille et des débits déterminés. ST-II est constitué de deux protocoles ; le protocole de transmission de données appelé ST, et le protocole *ST Control Message (SCMP)*. SCMP contrôle les arbres de diffusion, c'est-à-dire l'ajout ou l'élimination d'adresses cibles, et la négociation et le choix des paramètres de qualité de service.

Dans la cadre du système *Heidelberg Transport System (HeiTS)*, un protocole des transport, le *Heidelberg Transport Protocol (HeiTP)*, a été développé au dessus d'une implémentation de ST-II [36]. Des propriétés additionnelles à ST-II ont été ensuite proposées et implémentées par Herrtwich et Delgrossi [22]. Elles incluent une dégradation progressive de service, une approche similaire au travail de Pariss, Ventre, et Zhang[2] présenté précédemment, et une boucle de contrôle de synchronisation automatique entre émetteur et récepteur. Cette boucle permet d'optimiser le débit et d'éviter la saturation des tampons du côté récepteur. Quatre classes de fiabilité ont été suggérées pour le HeiTP : *ignorer les données erronées*, *éliminer les données erronées*, *indiquer les données erronées*, *et corriger les données erronées*. La définition de ces quatre classes de service a été motivée par le développement des techniques de compression de média continus comme MPEG[29]. En effet, des données sous forme compressée peuvent avoir des conséquences plus sévères en comparaison à la transmission d'un médium non compressé, lorsqu'elles sont erronées.

Dans le cadre du projet BERKOM [50], il a été développé un système de transport similaire à celui de HeiTS. Le service de transport est appelé *MMTS (Multimedia transport service)*. Il supporte les paramètres de qualité de service suivants : 1) le délai de bout-en-bout, 2) la taille maximale de l'unité de données de service, 3) la fréquence d'arrivée, 4) et quatre classes de fiabilité. Le protocole de transport MMTS est lui aussi implémenté au dessus de ST-II.

2.4.3 Couches supérieures et modèles de programmation

L'hypothèse de média ayant plusieurs degrés de qualité a été faite dans divers protocoles de niveau application. Dans le RSVP (*A New Resource Reservation Protocol*) [37], plusieurs versions de degrés de qualité différents de données de média continus sont diffusées. Les clients peuvent établir des connexions véhiculant uniquement les flots nécessaires à la satisfaction de leurs exigences de qualité de service.

Delgrossi et al. [12] proposent également une séparation des données multimédia en flots différents, les flots ayant des caractéristiques de qualité de service différentes. Par exemple, une vidéo codée MPEG[29] peut être transmise sur trois flots différents : les images I sont transmises sur un flot avec des paramètres de qualité de service garantis, et les images P et B sur des flots ayant des paramètres de type *meilleur effort*. De cette manière, il est possible d'optimiser l'utilisation de la bande passante disponible.

Mines et al. [48] propose quant à eux une interface de programmation donnant accès à un jeu d'objets de l'environnement distribué multimédia. Il suffit de connecter et de faire fonctionner (*Plug and Play*) ces objets. Les auteurs rapportent que l'environnement DAVE (*Distributed Audio Video Environment*) ainsi obtenu faciliterait le développement d'applications aussi bien centralisées que distribuées. Il permettrait l'intégration de nouveaux types d'information et de nouveaux périphériques, la réutilisation des objets, et supporterait l'interopérabilité et l'indépendance vis à vis des systèmes de communication. Les champs d'application cibles de DAVE sont la vidéo-conférence, l'archivage des média continus, le **contrôle de procédé à distance** et le télé-enseignement. Les objets constitutifs de DAVE pour une application de vidéo-conférence sont : 1)- la source réseau, 2)- le puits réseau, 3)- le compresseur, 4)- le décompresseur, 5)- la caméra, 6)- le haut-parleur, 7)- le microphone, 8)- et la fenêtre vidéo. Ces objets sont encapsulés par un gestionnaire effectuant le contrôle d'accès et l'allocation de ressources, et un gestionnaire d'objets proprement dits (voir figure 2.2).

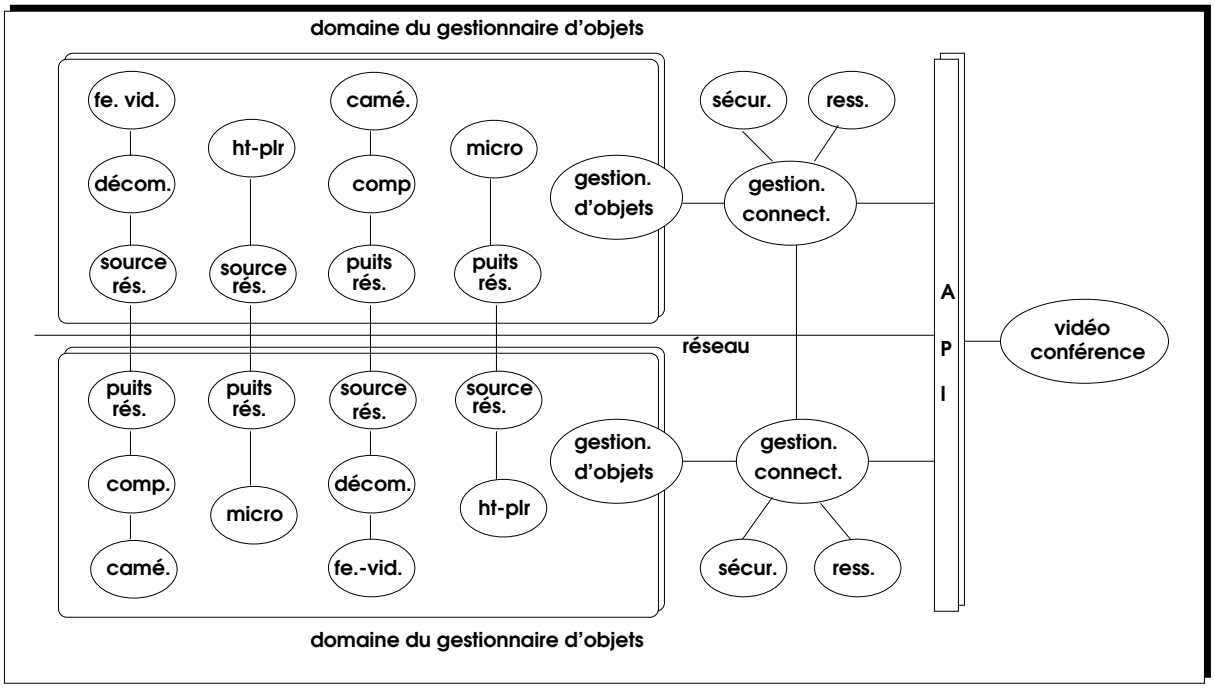


Figure 2.2 : Une application de vidéo-conférence basée sur le modèle de DAVE

Gopal et al. [17] proposent aussi Une approche objet développée dans le cadre du projet *TOURING MACHINE*. C'est une expérience qui, d'après les auteurs, devrait permettre de juger le degré de facilité que les réseaux de communication publics offrent pour le déploiement des applications multimédia du futur. Le principal travail est focalisé sur la conception et la réalisation d'une infrastructure logicielle de contrôle permettant le déploiement d'une large variété d'applications de communication. L'interface application définie dans le cadre de ce projet permet aux développeurs d'applications de manipuler des abstractions de base qui cachent la complexité des systèmes de communication. Ces abstractions sont : 1)- la session, 2)- le connecteur, 3)- le port, 4)- et le point terminal. Une session est une relation de contrôle entre programmes d'application offrant des services à des utilisateurs participant à une communication, ou agissant pour le compte de ces derniers. Un connecteur est une abstraction qui cache les détails de la réalisation physique du transport interconnectant les applications, de même que la complexité de contrôle du réseau sous-jacent. Les abstractions de port et de points terminaux permettent aux applications de participer simultanément à plusieurs sessions. Les abstractions visibles par les applications sont différentes des mécanismes sous-jacents de l'architecture interne Touring Machine (voir figure 2.3).

Elle est composée de 7 classes d'objets :

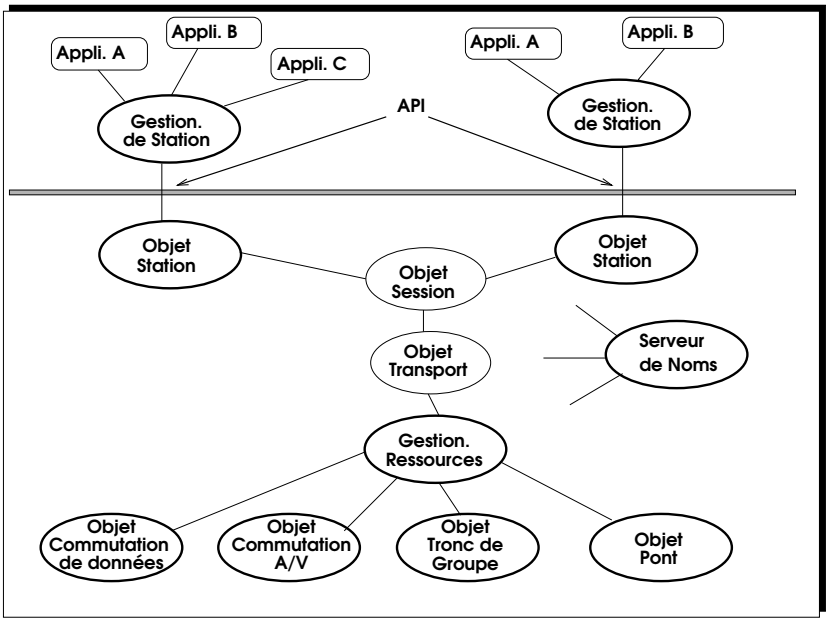


Figure 2.3 : L'architecture interne de Touring Machine

Le gestionnaire de station est un objet optionnel externe au système de Touring Machine. Il implémente les politiques définies par l'utilisateur pour l'arbitrage du partage des accès aux canaux de réseaux par plusieurs applications enregistrées au niveau d'une station.

L'objet station interne au système de Touring Machine, offre une interface application. Il propose par défaut une discipline FIFO pour la gestion des ressources d'accès au réseau de la station.

Le gestionnaire de ressources coordonne l'allocation et la désallocation des ressources système pour la réalisation des abstractions de type *connecteur*.

Les objets ressources contrôlent les vrais périphériques physiques, tout en cachant les détails et en offrant une abstraction logique aux niveaux supérieurs.

Le serveur de noms permet de stocker des ensembles d'attributs de diverses entités (utilisateurs, applications, points terminaux, sessions, ...). Il est accessible par la plupart des autres objets du système.

L'objet session est créé dynamiquement par l'objet station quand une application initialise une demande de communication.

L'objet transport est créé par l'objet session lorsque les ressources ont pu être allouées.

L'objet transport communique avec le gestionnaire de ressources lorsque les connecteurs sont modifiés par les applications durant une session pour une nouvelle allocation de ressources.

Tennenhouse et al. [9] proposent eux aussi une architecture logicielle de réalisation d'environnements de traitement de média continus. Ils décrivent le ViewStation du MIT, un environnement de traitement logiciel de média continus. Les auteurs sont d'avis que leur approche "vidéo à l'application" demande plus que les traditionnels numérisation et transport d'images animées. La chaîne de raisonnement qui a mené à leur conception est la suivante : i)- L'information vidéo doit être accessible et être manipulée par les applications. ii)- Une approche logicielle préservant les phénomènes d'échelle et la dégradation progressive est désirée. iii)- Le temps de perception, et non pas le temps réel, est le domaine d'intérêt d'applications interactives. iv)- Le logiciel résultant et le substrat de média doivent suivre l'évolution technologique. Ces 4 canevas de travail ont permis la conception de l'architecture de ViewStation (voir figure 2.4). L'environnement *VuSystem* qui incorpore cette approche de programmation pour systèmes à média continus, comporte un ensemble de conventions, un exécutif, un ensemble de modules de traitement de média continus, une bibliothèque de traitement d'images, et une interface utilisateur graphique/visuelle. Le substrat de média qui est séparé du système VuSystem par le système d'exploitation, offre des accès de bas niveau aux média (capture, affichage, etc.) et aux services de communication.

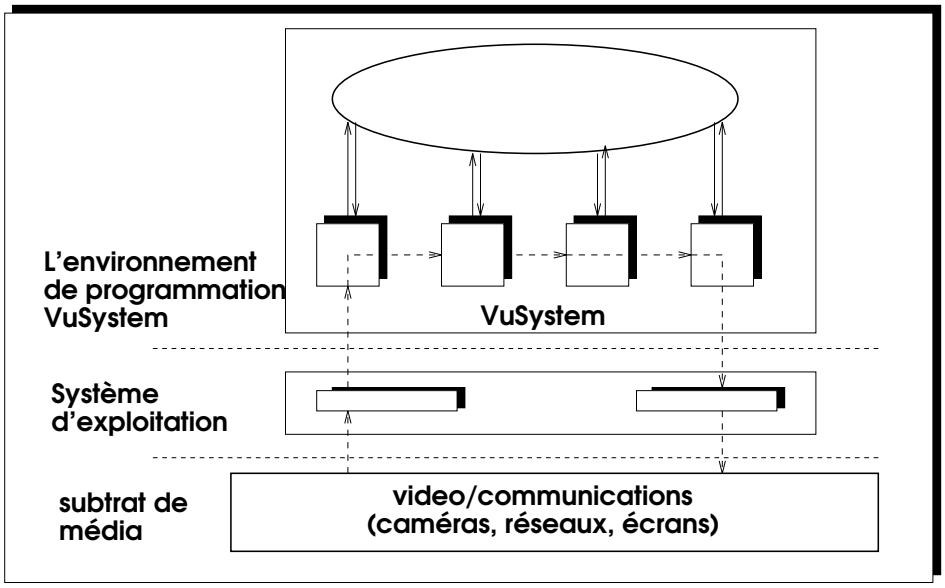


Figure 2.4 : L'Environnement de programmation de VuSystem

D'autres approches offrent des primitives pour la négociation entre différentes com-

posantes d'une application multimédia distribuée. Le MCAM *Application Layer Protocol for Movie Control Access, and Management* développé par Keller et Effelsberg [46] inclut entre autres paramètres de qualité de service la fiabilité, la vitesse, le mode et la direction. Le MCAM est basé sur un protocole X Windows [11] étendu.

Une autre direction de recherche est le développement d'approches intégrées incluant la mise en correspondance des paramètres de qualité de service entre les différents niveaux de service. Un exemple bien développé est le travail sur une architecture de qualité de service effectué dans le projet QoS-A de l'université de Lancaster [1]. Il met en correspondance, dans un environnement Chorus, la qualité de service d'une plate-forme d'application distribuée à l'ATM. Le QoS-A comprend un protocole de transport comparable à HeiTP et à MMTP, implémentés au dessus de ST-II. Nahrstedt et Smith ont présenté, pour leur part, le *QoS broker* [34] qui se charge de la négociation et de la traduction des paramètres de qualité de service entre les différentes couches. Florissi [40] a présenté le *Quality Assurance Language* (QuAL) qui améliore la gestion des paramètres de qualité de service pour les applications multimédia distribuées. Il offre un langage d'abstractions pour la spécification, la négociation, la renégociation et le contrôle de la qualité de service sur le réseau et les protocoles de niveau application, ceci en rapport avec les systèmes d'exploitation. Un premier prototype de QuAL est implémenté en Concert/C étendu et utilise ST-II comme protocole de réseau.

En ce qui concerne les standards de systèmes distribués ouverts, les travaux cités dans la littérature se situent en majorité autour d'une extension du modèle de l'ANSA (*Advanced Networked Systems Architecture*) [21,4]. Les modèles de calcul et d'ingénierie de l'ANSA initialement conçus pour les appels de procédures à distance ne se prêtent pas bien aux traitements répétitifs des média continus dans un système distribué multimédia. L'article de Coulson [16] décrit les résultats d'un travail de recherche dans l'extension de l'architecture ANSA pour le support de services multimédia. Il identifie les besoins imposés par le multimédia aux systèmes distribués, aux niveaux conceptuel et d'ingénierie, et décrit un ensemble de services qui peuvent être ajoutés à ANSA pour satisfaire ces besoins. Ces services comprennent une plate-forme de services de bas niveau qui pourraient être utilisés directement par les applications, ou utilisés pour la construction d'abstractions de haut niveau.

Dans un effort d'identification d'abstractions logicielles pour la programmation multimédia, Gibbs S. J. et Tsichristzis D. C. [19] proposent un modèle tout-objet de spécification des média continus ainsi que des composantes de connexion et de traitements de ces derniers dans une plate-forme multimédia distribuée. Une approche qui est similaire

à celle que nous adopterons au chapitre 4 lors de la spécification de nouveaux objets à ajouter au modèle VMD de MMS pour traiter et véhiculer les flots de média continus.

2.5 Conclusion

Après avoir précisé la terminologie, nous avons procédé à l'examen de l'état de l'art des technologies support de média continus.

Les réseaux hauts-débits actuels offrent suffisamment de bande passante et un délai acceptable pour le trafic multimédia. Ces ressources sont gérables par le biais de paramètres de qualité de service de bas niveau. Les protocoles de réseau, de transport et de session offrent des mécanismes pour le traitement des paramètres de qualité de service à travers des réseaux hétérogènes. La mise en correspondance des paramètres de qualité de service des couches hautes aux couches basses a été abordée dans plusieurs propositions de piles de protocoles comme HeiTS et Tenet. Elle exploite les caractéristiques des systèmes d'exploitation temps-réel préemptifs. Qui plus est, il existe des protocoles de haut niveau qui supporte une négociation globale de la qualité de service entre toutes les composantes impliquées dans une application multimédia distribuée. Nous pouvons conclure que les problèmes de communication pour les applications multimédia seront en principe résolus.

Forts de cette hypothèse, nous allons maintenant examiner les caractéristiques de MMS, la seule norme existante pour la conception et le développement d'applications temps-réel distribuées.

Chapitre 3

Les Limites de la Messagerie MMS

3.1 La norme de messagerie MMS

MMS (Manufacturing Message Specification) est une norme internationale de communication par messages pour données et informations de contrôle et de supervision entre équipements et ordinateurs d'applications. La communication est rendue indépendante : 1)- de la fonction application développée, 2)- du développeur d'équipement ou d'application. MMS est une norme internationale (ISO 9506) développée et maintenue par le Comité Technique Numéro 184 (TC184), de l'ISO.

Les services de messagerie offerts par MMS sont suffisamment génériques pour s'adapter à une large variété d'équipements, d'applications et d'industries. Par exemple, le service MMS **Read** permet à une application ou à un équipement de lire une variable d'une autre application ou d'un autre équipement. Que l'équipement soit un automate programmable (PLC), ou un robot, les services et messages MMS sont identiques. De même, des applications aussi diverses que la gestion de stock, l'annonce de pannes, la gestion d'énergie, le contrôle de distribution d'énergie électrique, le contrôle d'inventaire, et le positionnement d'antennes dans l'espace utilisent la messagerie MMS. Dans des industries aussi variées que l'automobile, l'aérospatiale, la pétrochimie, l'énergie, l'usine de production et l'exploration de l'espace, un intense usage pourrait être fait de la norme MMS.

3.2 L'histoire de MMS

Au début des années 80, un groupe de vendeurs de machines à commandes numériques (NC), de fabricants de machines et d'utilisateurs travaillant sous les auspices du comité IE31 de l'EIA (Electronic Industries Association) développe un "draft" de proposition de norme, le draft numéro 1393A intitulé "User Level Format and Protocol for Bidirectional Transfer of Digitally Encoded Information in Manufacturing Environment". Quand General Motors démarre son projet MAP (Manufacturing Automation Protocol) en 1980, il utilise le draft 1393A de l'EIA comme base pour un protocole de messagerie plus générique pouvant être utilisé pour les commandes numériques (NCs), les automates programmables (PLCs), les robots et autres équipements intelligents rencontrés communément dans les environnements de production automatisée. Le résultat en est la norme MMFS (Manufacturing Message Format Standard). MMFS est utilisée dans les spécifications de la version 2 de MAP publiées en 1984. Pendant cette utilisation initiale de

MMFS il devient apparent qu'une norme de messagerie plus rigoureuse était nécessaire. MMFS permet trop de choix pour les développeurs d'applications et d'équipements. Ce qui résultera en de nombreux dialectes de MMFS incompatibles. De plus MMFS n'offre pas suffisamment de fonctionnalités pour être utilisée dans les systèmes de contrôle de procédé (PCS) rencontrés dans les industries à traitement continu. Dans le but de développer un système de messagerie entre équipements intelligents d'automatisme générique et non spécifique à une industrie, le projet MMS va démarrer sous les auspices du Comité Technique numéro 184 de l'organisme international de normalisation ISO.

Le résultat est une norme basée sur le modèle réseau de l'OSI (Open Systems Interconnection). Une version DIS (Draft International Standard) de MMS est publiée en décembre 1986 comme ISO DIS 9506. La version DIS de MMS (Version 0) tranche les problèmes de MMFS mais n'est pas encore une norme internationale IS (International Standard). Confronté au problème de délai de publication de novembre 1988, les comités techniques de MAP feront références à la version DIS de MMS pour la spécification de MAP V3.0. En décembre 1988 la version IS de MMS (Version 1) est publiée comme ISO 9506, parties 1 et 2. Ce n'est qu'aprés le développement des accords de compatibilité antérieure par le NIST (National Institute of Standards and Technology) que la version IS de MMS sera référencée par les spécifications de MAP V3.0.

3.3 La norme MMS

La norme MMS (ISO 9605) est gérée par le comité technique numéro 184, Automatisation Industrielle, conjoint à l'ISO et l'IEC (International Electrotechnical Commission), et consiste en deux parties [1]. Les parties 1 et 2 définissent ce qui est considéré comme le "noyau" de MMS [25,26]. La partie 1 est la spécification de services. La spécification de services contient une définition : 1)- du VMD (Virtual Manufacturing Device), 2)- des services (ou messages) échangés entre les noeuds du réseau, et 3)- les attributs et paramètres associés au VMD et aux services. La partie 2 est la spécification de protocoles. La spécification de protocoles définit les règles de communication qui incluent 1)- le séquencement des messages à travers le réseau, 2)- le formatage (ou encodage) des messages, et 3)- l'interaction de la couche MMS et les autres couches du système de communication. Une norme appelée ASN.1 - ISO 8824 (Abstract Syntax Notation Number One) est intégrée à la couche présentation pour spécifier les formats de messages MMS.

[1] Hormis les Normes d'accompagnement [27,28,23,24]

MMS offre un ensemble riche de services pour la communication de bout-en-bout à travers le système de communication. MMS a été utilisée comme protocole de communication pour de nombreux équipements de contrôle industriels communs comme les commandes numériques (CNSs), les automates programmables (PLCs), et les robots. Il existe des applications MMS dans l'industrie d'énergie électrique telles que les unités de terminal distant (RTUs), les systèmes de gestion d'énergie (EMS) et autres équipements électroniques intelligents (IED) tels que les contacteurs. La majorité des plate-formes de calcul disposent d'une connectivité MMS, provenant soit du fabricant de l'ordinateur, soit d'une tierce partie. Certaines des applications disponibles incluent des interfaces d'application (API), des systèmes de supervision graphiques, des passerelles, des traitements de texte, et des systèmes de gestion de bases de données relationnelles (RDBMS). Les implémentations de MMS supportent une variété de liens de communication incluant Ethernet, Token Bus, RS-232, OSI, TCP/IP, MiniMAP, etc. et peuvent se connecter à plusieurs autres types de systèmes en utilisant des ponts de réseau, des routeurs, et des passerelles.

3.4 Les avantages de MMS

MMS offre des avantages par la réduction des coûts de construction et d'utilisation de systèmes automatisés. En particulier, MMS est appropriée pour toute application utilisant des mécanismes de communication communs pour la réalisation des diverses fonctions de communication relatives aux accès temps-réel des données distribuées de supervision et de contrôle de procédé. Quand on regarde la façon dont l'utilisation des services de communication communs comme MMS peut bénéficier à un système particulier, il est important d'évaluer les trois effets majeurs de l'utilisation de MMS qui contribuent en une réduction des coûts : 1)- l'Interopérabilité, 2)- l'Indépendance, 3)- l'Accès.

L'**interopérabilité** est la capacité de deux ou plusieurs applications réseau d'échanger des données utiles de supervision et de contrôle de procédé entre elles sans que l'utilisateur des applications ait à créer l'environnement de communication. Pendant que de nombreux protocoles de communication peuvent offrir un certain niveau d'interopérabilité, nombre d'entre eux sont soit très spécifiques (à la marque, au type de l'application ou de l'équipement, à la connectivité réseau, ou à la fonction réalisée – voir l'indépendance ci-après) ou pas assez spécifique (offre une trop grande possibilité de choix de la façon dont un développeur utilise le système de communication.)

L'**indépendance** permet à l'interopérabilité d'être assurée indépendamment :

- DU DÉVELOPPEUR DE L'APPLICATION. D'autres schémas de communication sont souvent spécifiques à une marque particulière (ou même un modèle dans certains cas) d'application ou d'équipement. MMS est définie par des organisations internationales de normalisation avec la participation d'experts de l'industrie et de vendeurs.

- DE LA CONNECTIVITÉ DU RÉSEAU. MMS devient l'interface réseau des applications, les isolant de ce fait de la plupart des aspects non-MMS du réseau et la façon dont le réseau transfère les messages d'un noeud à un autre.

- DE LA FONCTION RÉALISÉE. MMS offre un environnement de communication commun indépendant de la fonction réalisée. Une application de contrôle d'inventaire accède aux données de production contenues dans un équipement de contrôle de la même façon qu'un système de gestion d'énergie lirait les données de consommation d'énergie du même équipement.

L'**accès** aux données est la capacité des services d'application d'obtenir les informations requises par les applications. Même si pratiquement tous les schémas de communication de contrôle peuvent offrir l'accès aux données au moins de façon minimale, ils manquent les autres avantages de MMS, en particulier l'Indépendance.

3.5 La justification de MMS

Le vrai défi dans toute tentative de développement de la justification industrielle de MMS (ou tout investissement réseau) est d'affecter des valeurs d'importance aux bénéfices pour un objectif industriel donné. Dans le souci de le faire proprement, il est important de comprendre la relation entre les fonctions d'application, les fonctions de connectivité et les fonctions industrielles du réseau.

Afin d'assigner une valeur à l'utilisation de MMS, il est important de comprendre d'abord le rôle que joue MMS en rapport avec les applications. MMS, en tant que protocole de couche application, offre des services d'application aux fonctions industrielles, et non pas des services de connectivité. Il est commun de voir le réseau uniquement comme un mécanisme de transfert de messages (connectivité uniquement). Cette vision cache la valeur des fonctions d'application parce qu'elles deviennent indiscernables des applications industrielles qui doivent alors offrir les fonctions d'application réseau. Cependant

les coûts demeurent. Justifier MMS exige de la part de l'utilisateur qu'il reconnaisse la valeur apportée par les fonctions application de réseau pour faciliter l'interopérabilité, l'indépendance et l'accès aux données.

Dans certains cas le bénéfice d'infrastructures de communication communes offertes par MMS est uniquement réalisé quand le système est utilisé, maintenu, modifié, et étendu au fil du temps. Ainsi, la justification d'un tel système devrait considérer le coût dans un cycle de vie par rapport au coût d'acquisition. Il est également important de ne pas sous-estimer le coût associé au développement, à la maintenance, et à l'extension de fonctions d'application qui doivent être créées si MMS n'est pas utilisé. Un élément clé dans l'affectation des valeurs d'importance est la compréhension du fait que les fonctions industrielles sont celles qui apportent des valeurs ajoutées à l'entreprise. Le coût des fonctions d'application réseau appropriées réduit directement l'effort qui pourrait être investi dans le développement, la maintenance, et l'extension de fonctions industrielles.

Ce paragraphe de justification de MMS est inspiré de la présentation de MMS faite par Mackiewicz [47]. Avec MMS, les infrastructures de communication sont construites une fois et re-utilisées par toutes les fonctions industrielles[47].

3.6 Les services d'applications MMS

La propriété clé de MMS est le modèle d'Équipement Virtuel de Production (VMD). Le modèle VMD spécifie la manière dont les équipements MMS, aussi appelés serveurs, se comportent vus d'une application MMS cliente externe. MMS permet aussi à toute application ou tout équipement d'offrir simultanément des fonctions client et serveur. En général, le modèle VMD définit :

- des objets (ex. les variables) qui sont contenus dans le serveur,
- des services qu'un client peut utiliser pour accéder et manipuler ces objets (ex. lire ou écrire une variable), et
- le comportement du serveur à la réception des requêtes de service des clients.

La suite de cet aperçu sur MMS donnera un résumé des objets définis par le modèle VMD et des services MMS offerts pour accéder et manipuler ces objets. En dépit du fait que l'étendue des objets et services soit très large, une application donnée ou un

équipement n'a besoin d'implémenter qu'un sous-ensemble quelconque de ces objets et services qui sont utiles dans une situation particulière donnée.

3.7 Le modèle VMD

L'objectif premier de MMS est de spécifier une norme de communication pour les équipements et les ordinateurs d'application qui permet un grand degré d'interopérabilité. Afin d'atteindre ce but, il faut que MMS définisse plus que les formats des messages échangés – un format de message commun, ou protocole, est uniquement un aspect de l'interopérabilité.

En plus du protocole, la norme MMS donne aussi des définitions :

- Des objets : MMS définit un ensemble d'objets communs (ex. les variables, les programmes, les événements, etc.) et définit les attributs visibles réseau de ces objets (ex. les noms, les valeurs, les types, etc.).
- Des services : MMS définit un ensemble de services de communication (ex. Read, Write, Delete, etc.) pour l'accès et la gestion de ces objets dans un environnement réseau.
- Du comportement : MMS définit le comportement visible réseau qu'un équipement pourrait manifester lors du traitement des services.

Cette définition d'objets, de services (87 au total), et du comportement comprend une définition de la manière dont les équipements et les applications communiquent et que MMS appelle le modèle VMD (voir figure 3.1). Le modèle VMD spécifie uniquement les aspects visibles réseau de la communication. Les détails internes de la manière dont un équipement réel implémente le modèle VMD (c-à-d le langage de programmation, le système d'exploitation, le type de CPU, les systèmes d'I/O, etc.) ne sont pas spécifiés par MMS. En se focalisant sur les aspects visibles réseau d'un équipement, le modèle VMD est suffisamment spécifique pour offrir un haut niveau d'interopérabilité tout en demeurant suffisamment général pour permettre l'innovation dans l'implémentation d'applications et équipements, rendant ainsi MMS applicable à un large champ d'industries et d'équipements.

Un aspect clé du modèle VMD est la relation client-serveur entre les applications et les équipements.

Un serveur est une application ou un équipement qui contient un VMD et ses objets.

Un client est une application (ou équipement) qui demande l'accès aux données ou une action du serveur. Dans un sens très général, un client est une entité réseau qui émet des requêtes de services MMS à un serveur. Un serveur est une entité réseau qui répond aux requêtes de service MMS d'un client. Alors que MMS définit les services pour les clients et les serveurs, le modèle VMD définit le comportement visible réseau des serveurs uniquement.

Plusieurs applications MMS et équipements compatibles MMS offrent des fonctions MMS clients et serveurs. Le modèle VMD définira uniquement les fonctions serveurs de ces applications. Toute application MMS ou tout équipement MMS offrant des fonctions de service MMS doit suivre le modèle VMD pour les aspects visibles réseau de l'application ou de l'équipement. Les clients MMS sont uniquement contraints de se conformer aux règles régissant le format de message ou la construction et le séquencement de ces derniers (protocole).

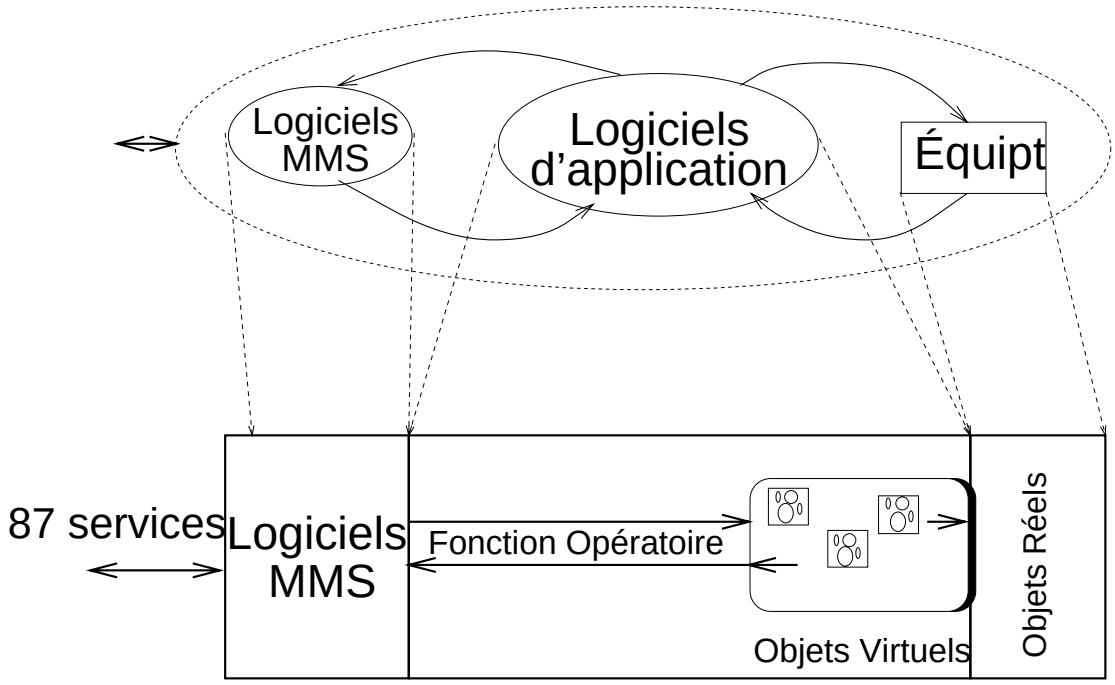


Figure 3.1 : Le modèle de l'équipement virtuel de production (VMD)

Le VMD en tant que tel peut être considéré comme un objet auquel tous les autres objets MMS sont subordonnés (variables, domaines, etc. sont contenus dans le VMD). MMS offre

des services tels que Status, UnsolicitedStatus, et Identity pour obtenir les informations et l'état relatifs au VMD. Il offre également des services tels que GetNameList et Rename pour la gestion et l'obtention d'informations relatives aux objets définis dans le VMD.

3.8 Les objets du modèle VMD

Nous donnons ici une présentation succinte des objets sur lesquels se base le modèle VMD.

3.8.1 Les variables et les objets types nommés

MMS offre un cadre compréhensible et flexible d'échange d'information sur les variables à travers un réseau. Le modèle d'accès aux variables MMS incluent les possibilités de variables nommées, de variables non-nommées (adressées), et de listes nommées de variables. MMS permet aussi la description de type de variables à être manipulées comme des objets MMS à part entière (objet type nommé). Les variables MMS peuvent être simples (ex. entier, booléen, virgule flottante, chaîne, etc.) ou complexes (ex. tableaux et structures). Les services disponibles pour l'accès et la manipulation des objets variables et type nommé sont très puissants et incluent :

- Les services Read et Write permettant aux applications clients MMS d'accéder au contenu des variables nommées, des variables non-nommées, et des listes nommées de variables.

- Le service InformationReport permettant au serveur de rapporter le contenu d'une variable à un client distant de manière non sollicitée.

- Les services Define, Delete, et GetAttribute disponibles pour les objets variable et type nommé. Ils permettent aux clients de manipuler l'environnement d'accès aux variables.

Les options de services permettent l'accès aux groupes de variables par de simples requêtes, l'accès partiel (ou alternatif) aux tableaux ou à des structures complexes.

3.8.2 Les objets de contrôle de programmes

Le modèle d'exécution du VMD définit deux objets pour le contrôle de l'exécution des programmes dans le VMD. Un domaine MMS est défini comme un objet qui représente une ressource dans le VMD (ex. la mémoire dans laquelle un programme est stocké). Une invocation programme est définie comme un groupe de domaines dont l'exécution peut être contrôlée et suivie. Quelques uns des services offerts aux clients par le modèle d'exécution du VMD sont :

- Les services pour commander au VMD de charger (décharger) les domaines à partir de (sur) un utilisateur MMS ou un système de fichier (sur le VMD ou externe au VMD).
- Les services permettant à un VMD de demander le chargement (déchargement) d'un domaine à partir de (sur) un client.
- Les services Start, Stop, Reset, Resume, et Kill pour le contrôle de l'exécution des invocations de programme.
- Les services pour effacer, créer, et obtenir les attributs des domaines et des invocations de programme.

Les changements sur les invocations de programmes peuvent être liés à des événements MMS.

3.8.3 Les objets événements

Le modèle de gestion d'événement définit plusieurs objets nommés :

- la condition événementielle : un objet qui représente l'état d'un événement (actif ou inactif).
- l'action événementielle : un objet qui consiste en une action qui doit être réalisée par le VMD à l'occurrence d'un changement d'état de la condition événementielle, et
- l'enveloppe événementielle : un objet qui représente lesquels des clients MMS doivent être alertés en cas de changement d'état d'une condition événementielle.

Le modèle de gestion d'événement offre un riche ensemble de services aux clients MMS :

- des services de notification aux clients des états d'événements et des services d'acquittement de ces notifications pour les clients.
- des services pour obtenir les résumés des conditions événementielles et des enveloppes événementielles (appelés des résumés d'alarmes).
- des services pour effacer, définir, obtenir l'état et les attributs, et contrôler les conditions événementielles, les actions événementielles, et les enveloppes événementielles.

3.8.4 Les objets sémaphores

Les sémaphores MMS sont des objets nommés qui peuvent être utilisés pour le contrôle d'accès à d'autres ressources et objets du VMD. Par exemple, un VMD qui contrôle l'accès à une consigne (une variable) pour une boucle de contrôle pourrait utiliser des sémaphores pour permettre à un seul client à la fois de changer la consigne (à l'aide du service MMS Write par exemple). Le modèle de sémaphore MMS définit deux types de sémaphores. Les sémaphores à jetons sont utilisés pour représenter une ressource spécifique sous le contrôle du VMD. Les sémaphores par groupe consistent en un ou plusieurs jetons nommés représentant chacun une ressource similaire, mais distincte, sous le contrôle du VMD. MMS offre des services de sémaphore permettant aux clients de :

- Acquérir (TakeControl) et libérer (Relinquish) les sémaphores
- Définir (Define), détruire (Delete), et obtenir les attributs ou l'état des sémaphores.

3.8.5 L'objet Journal

Un journal MMS est un objet qui représente un enregistrement daté des données. Chaque entrée dans un journal peut contenir l'état d'un événement, la valeur d'une variable, ou une chaîne de caractères (appelée annotation) que le VMD, ou le client MMS, introduit dans le journal. Les services offerts permettent aux clients de créer, de lire, de détruire, et d'effacer le journal (entièrement ou partiellement).

3.8.6 L'objet station opérateur

La station opérateur est un objet qui représente un moyen de communication avec l'opérateur du VMD par un clavier et un écran. Un service de sortie (output) est disponible pour afficher une chaîne alphanumérique sur un terminal texte. Un service entrée (input) est disponible pour obtenir une chaîne alpha-numérique du clavier avec ou sans "prompt" sur le terminal.

3.8.7 Les fichiers

MMS définit en annexe un ensemble de services simples pour le transfert, le renommage, et la destruction de fichiers dans le VMD.

3.8.8 La norme MMS utilise la syntaxe ASN.1

La norme MMS utilise la syntaxe (ASN.1 - ISO 8824) pour la présentation des primitives de services entre clients et serveurs. Nous présentons ci-après un extrait la spécification ASN.1 des types de données de base de MMS :

```
FileName ::= SEQUENCE OF GraphicString

TimeOfDay ::= OCTET STRING (SIZE (4 | 6))

Identifier ::= VisibleString

Integer8   ::= INTEGER
Integer16  ::= INTEGER
Integer32  ::= INTEGER

Unsigned8  ::= INTEGER
Unsigned16 ::= INTEGER
Unsigned32 ::= INTEGER

ObjectName ::= CHOICE
```

```
        {
        vmd-specific [0] IMPLICIT Identifier,
        domain-specific [1] IMPLICIT SEQUENCE
                {
                domainId Identifier,
                itemId          Identifier
                },
        aa-specific [2] IMPLICIT Identifier
        }

ApplicationReference ::= SEQUENCE
{
ap-title   [0] ISO-8650-ACSE-1.AP-title  OPTIONAL,
ap-invocation-id [1] ISO-8650-ACSE-1.AP-invocation-identifier
OPTIONAL,
ae-qualifier     [2] ISO-8650-ACSE-1.AE-qualifier
OPTIONAL,
ae-invocation-id [3] ISO-8650-ACSE-1.AE-invocation-identifier
OPTIONAL
}

Priority ::= Unsigned8

normalPriority Priority ::= 64
```

Integer8, *Integer16*, et *Integer32* sont des entiers de 8, 16 et 32 bits.

Unsigned8, *Unsigned16* et *Unsigned32* sont des entiers son signés de 8, 16 et 32 bits.

Le type *ObjectName* est celui utilisé pour le nommage des objets de MMS.

Il pêut être soit :

- de portée *VMD-Specific*, dans ce cas il est constitué uniquement d'un identificateur *Identifier* dans l'environnement OSI,
- de portée *Domain-Specific*, dans ce cas il est constitué d'un identificateur OSI du domaine et d'un identificateur OSI de l'instance,

- de portée *AA-Specifif*, dans ce cas il est constitué uniquement d'un identificateur OSI.

Les références d'application (*ApplicationReference*) permettent d'identifier les applications à partir des champs *ap-title*, *ap-invocation-id*, *ae-qualifier* et *ae-invocation-id* de l'élément de service d'application de contrôle d'association (*ACSE*). [2]

3.9 Les insuffisances de MMS pour les média continus

Les média continus, tels que nous les avons décrits dans le chapitre 2 ont des exigences de qualité de service différentes de celles des messages pour lesquels la norme MMS a été définie. D'où la nécessité d'effectuer des aménagements sur MMS pour respecter les exigences de qualité de service de l'information sous forme de média continus.

3.9.1 MMS n'offre pas de services de simple connectivité

Une simple connectivité permettrait en effet la transmission de données de types inconnus, et sous des protocoles inconnus de MMS, y compris des média continus. En dehors des services de gestion des domaines et fichiers, MMS ne permet pas le transfert de données brutes sans signification MMS. Tous les messages échangés dans le cadre de MMS ont une signification précise, et n'interviennent que dans des circonstances bien déterminées (ce sont des primitives de services MMS). Ces messages sont à priori définis pour véhiculer les informations sous forme média discrets.

3.9.2 MMS ne garantit pas de débit soutenu

Imaginons en effet qu'on veuille utiliser les objets variables et événements MMS pour la transmission d'un flot de médium continu de vidéo. Soit un segment de données correspondant à une image numérisée. Associons le à une variable *SampledFrame* (au sens MMS) de type *ImageFrame* [3]. La variable est remontée périodiquement à un client MMS (VMD ou application).

[2] Une spécification ASN.1 de MMS est disponible en ligne par WWW [39]

[3] le type *ImageFrame* est par exemple une chaîne ou un tableau d'octets, ou une structure plus complexe.

Cette remontée périodique nécessite la création et la gestion d'objets condition, action et enveloppe événementielles appropriés.

Définissons un objet condition événementielle qui devient actif chaque fois que la variable *SampledFrame* est remise à jour, un objet action événementielle consistant en un service *Read* de la variable *SampledFrame* pour le compte d'une application cliente définie par un objet enveloppe événementielle.

La gestion des objets événements permet d'assurer le synchronisme entre éléments consécutifs d'un médium continu. La condition événementielle passe à l'état ACTIVE tous les 25ème de seconde (25 images par seconde). L'action événementielle consistant en la requête de service *Read* de la variable *SampledFrame* est exécutée et le résultat de la lecture est envoyé au client MMS concerné dans une requête *EventNotification*.

Un tel artifice résoudrait le problème de synchronisme de transfert entre éléments consécutifs des média continus à l'émission, mais resterait impuissant au niveau de la transmission réseau, étant donné qu'une communication assurée de bout-en-bout est utilisée.

On peut même choisir de ne pas faire la requête de service d'acquittement de notification chez le client tel que le permet MMS, mais cet artifice demeure inapproprié au transfert de média continus.

Qui plus est, la fonction exécutive du VMD risque de se retrouver occupée uniquement à effectuer ce transfert au détriment des autres services MMS. En supposant qu'une bonne gestion des événements permette de garantir le débit de transfert par la fonction exécutive. MMS ne dispose pas de mécanisme explicite de garantie de débit de transfert sur le réseau, ce qui risque d'affecter négativement la qualité de service de l'information sous forme de médium continu.

3.9.3 Transferts de domaines et de fichiers inefficaces

Imaginons un autre cas de figure où le médium continu est représenté par un objet domaine MMS. Considérons le de taille fixe et cyclique [4]. Le médium continu peut alors être transmis par le biais des services de transfert de domaines. Ici aussi les éléments du média sont transmis sous forme de requêtes de services qui empruntent des chemins

[4] une structure de tampon circulaire dans la mémoire de la carte de compression par exemple.

de communication conçus pour des données sûres. On est loin de pouvoir respecter la qualité de service de l'information contenue dans le médium continu. Cette remarque est aussi valable pour une représentation du médium continu sous forme de fichier.

Les services MMS de transfert de domaines et de fichiers ne seront jamais en mesure de respecter le synchronisme entre les éléments consécutifs du médium continu, et encore moins de garantir les aspects de débits. De plus, les mécanismes obligatoires de reprise sur erreur dans ce cas, vont également à l'encontre des caractéristiques temps-réel recherchées.

3.9.4 Une interface de station opérateur limitée

L'interface de la station opérateur du VMD se limite à un clavier et un terminal alphanumériques évidemment inadaptés aux nouveaux types d'informations audio, vidéo, animations et autres, véhiculées par des média continus. Les chaînes de caractères alphanumériques entrées et sorties par la station opérateur sont insuffisantes.

Il faut intégrer des périphériques tels que le microphone, la caméra vidéo, les haut-parleurs et le moniteur vidéo à l'objet station opérateur pour les nouveaux types de média continus.

3.10 Conclusion

Après une présentation du cadre historique de la définition de MMS, de ses avantages, de ses services d'application, de son modèle d'équipement virtuel VMD, nous avons mis en évidence les aspects de MMS qui nous paraissent inadéquats aux flots de média continus.

Dans une perspective de systèmes automatisés multimédia du futur appelés à générer, traiter, transférer ou exploiter des informations sous forme de média continus telles que les séquences d'imagerie échographique ou de rayons X ou infrarouges, il est nécessaire de pourvoir le VMD de nouveaux objets et services appropriés qui permettront la génération, le traitement, le transfert ou l'exploitation de média continus.

Le chapitre suivant est notre propostion d'extension de MMS pour combler ces insuffisances.

Chapitre 4

Propositions d’extensions de la norme MMS

4.1 Introduction

Les chapitres 2 et 3 précédents nous ont permis de faire le tour de quelques approches d'interface de programmation de systèmes multimédia distribués, et de présenter la messagerie MMS, et ses limites pour le traitement, la génération, le transfert et la restitution de l'information sous forme de média continus. Dans ce chapitre nous présentons une approche d'extension de MMS. Nous commençons par présenter une première approche d'extension que nous avons proposée et qui ne couvre pas la totalité des fonctionalités d'une chaîne de production-restitution de média continus. Nous proposons par la suite une décomposition fonctionnelle d'une chaîne complète de production-restitution de média continus. Cette décomposition nous permet alors de définir des objets relatifs aux média continus. Ces objets, ainsi que d'autres que nous définissons sont ensuite intégrés à l'environnement de communication MMS. Le chapitre se termine par la spécification ASN.1 des services MMS de gestion de l'objet médium continu. Une description verbale des autres nouveaux services est également donnée.

4.2 Un modèle VMD multimédia (MM-VMD) : approche I

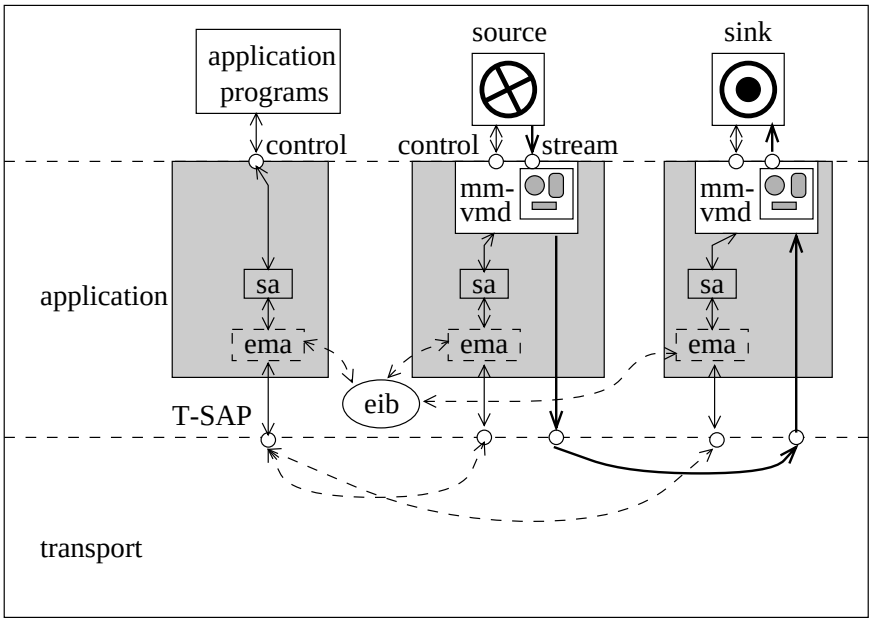


Figure 4.1 : Un modèle de VMD multimédia reportant les traitements synchrones sur la fonction exécutive

L'intégration de média continus au modèle VMD donne naissance à ce que nous appelons l'équipement virtuel multimédia de production (MM-VMD). Le MM-VMD contient des objets relatifs aux média continus en plus des objets conventionnels du VMD. La première proposition que nous avons faite dans ce sens au début du projet [42,43,30] prend en compte uniquement les aspects de production (source) et de restitution (puits) de l'information son et image (voir figure 4.1). Les programmes d'application et les équipements terminaux multimédia sont interfacés par des ports de contrôle pour les commandes et les retours d'état, et des ports flots (stream) pour les flots de média continus. Un agent de service *sa* assure la réalisation du protocole client-serveur pour les services MMS classiques. L'agent de service utilise les services d'un agent de gestion d'environnement *ema* lorsqu'il s'agit de services de gestion d'environnement MMS. L'agent de gestion d'environnement a accès aux données d'environnement contenues dans la base de données d'environnement, le *eib*. Sur la figure 4.1, une application cliente configure une connexion entre un équipement source et un équipement puits de flots de média continus. Des services de démarrage, de pause, de stop, d'adjonction de nouveaux puits ou d'élimination de puits peuvent ensuite être demandés par cette application cliente. Les notions de source et puits telles que nous les présentons ici sont plus générales que celles du modèle DAVE [48] dans lequel les sources et les puits représentent essentiellement les points d'accès aux services de communication.

Le MM-VMD comporte un port de contrôle (*control*) et un port de flots de média continus (*stream*) avec les équipements source et puits de média continus. Tous les traitements intermédiaires entre la production et la restitution de flots sont reportés au coeur du VMD. La fonction exécutive est alourdie par des traitements périodiques tels que le traitement du signal pour évaluation de conditions événementielles, et la transmission ou la réception des flots.

Cette approche pèche donc par sa non virtualisation des traitements intermédiaires entre génération et restitution de média continus. Le nouveau modèle que nous décrivons dans la suite part de la décomposition d'une chaîne complète de production-restitution de média continus. Il intègre tous les objets fonctionnels de cette chaîne qui, virtualisés sont contrôlables par des services MMS.

4.3 Les objets relatifs aux média continus

À la faveur des promesses des progrès technologiques, nous nous permettons de faire une décomposition fonctionnelle, aussi fine que possible, d'une chaîne de production et de restitution de l'information sous forme de média continus. Les éléments fonctionnels que nous identifions sont ensuite considérés comme des agents de média continus auxquels d'autres objets sont associés pour en assurer le fonctionnement et le contrôle.

4.3.1 De la production à la restitution des média continus

Conversion A/N : La première étape de la chaîne est la conversion analogique-numérique. L'information passe de son support sous forme de signal continu (grandeur physique variable dans le temps) à une représentation sous forme de séquences de média discrets constituant un médium continu.

Filtrage : Le médium continu ainsi obtenu peut éventuellement passer par un étage de filtrage numérique qui permet d'éliminer les bruits parasites et de ne retenir que la composante significative pour l'information supportée.

Traitement du signal : Le filtrage est un algorithme de traitement de signal, mais il en existe d'autres tels que la reconnaissance de la parole ou la reconnaissance de forme. Le médium continu peut passer à travers une étape de traitement du signal où un tel traitement est effectué.

Compression : L'étape suivante éventuelle est la compression dont le but est de réduire le volume de données requis pour représenter l'information sans en perdre la signification.

Multiplexage : Il peut arriver que deux flots de média continus compressés ou non soient multipléxés et synchronisés, à une étape de multiplexage, pour ne représenter qu'un nouveau type d'information combinant les types d'information des média multipléxés. C'est le cas, notamment lorsque les média de vidéo et d'audio compressés sont multipléxés pour former le médium *système MPEG*[29].

Transmission : Une fois le débit du médium continu réduit, on peut dans une étape de transmission émettre la suite de média discrets le constituant au moyen d'une infrastructure de communication garantissant des exigences en débit, en délai et en gigue de délai relatives à l'information supportée.

Réception : Dans une étape de réception, les média discrets constituants du médium continu sont reçus de l'infrastructure de communication garantissant les exigences en débit, en délai et en gigue de délai relatives à l'information supportée.

Stockage : Le médium continu peut être stocké (sur un disque dur par exemple) pour usage ultérieur dans une étape de stockage.

Extraction : Le médium continu de durée finie stocké peut être accédé dans une étape d'extraction.

Démultiplexage : Un médium continu composé de plusieurs média continus multiplexés peut être décomposé en ses éléments de base dans une étape de démultiplexage.

Décompression : Un médium compressé doit être décompressé dans une étape de décompression afin de permettre la restitution de l'information.

Transcodage : Un médium continu se trouvant dans un format donné peut nécessiter une étape de transcodage dans laquelle la transformation du codage est effectuée (ex: du format YUV au format RGB pour les images).

Conversion N/A : Un médium démultiplexé, transcodé ou décompressé passe par une étape de conversion numérique-analogique dans laquelle les média discrets le composant sont traduits en un signal analogique que l'utilisateur humain est capable d'interpréter.

Synchronisation inter-flots : Il est parfois nécessaire de synchroniser deux média continus indépendants afin d'obtenir une meilleure qualité du rendu simultané de ces média (synchronisation labiale entre le médium audio et le médium vidéo d'un film ou d'une application de vidéo-conférence).

Discussion

Cette approche par décomposition en éléments fonctionnels n'est pas nouvelle dans le monde du multimédia. Certains modèles de la littérature proposent un schéma de décomposition similaire. Ce qui leur fait parfois défaut, par rapport à notre objectif d'intégration de média continus au modèle de MMS, c'est la nature des besoins pour lesquels ils ont été spécifiés. Ces modèles se limitent en effet à l'exploitation de l'information par des humains et non pas nécessairement par des applications d'automatisme. A titre d'exemple, le "transformateur" du modèle décrit par Gibbs et al. [19] sert plus à effectuer des opérations de transcodage (transformation) qu'à aider à l'évaluation

d'une condition événementielle. Les spécifications objets de leur modèle sont bien traitées, mais il manque les aspects de leur utilsation par des applications. Il est alors important d'inclure de tels objets dans un modèle client-serveur de type MMS pour en facilité l'utilisation, la gestion et le contrôle.

4.3.2 Exemple de chaîne de production-restitution

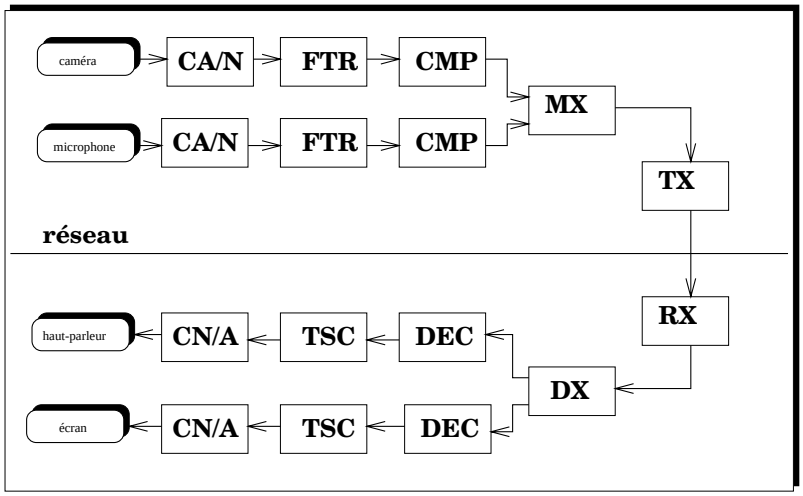


Figure 4.2 : Un exemple de chaîne de production-restitution

Un exemple de chaîne de production-restitution de média continus entre une caméra et un microphone d'un ordinateur, et un écran et un haut-parleur d'un deuxième ordinateur est illustré sur la figure 4.2. Deux organes de conversion analogique-numérique d'un ordinateur convertissent respectivement le signal audio provenant d'un microphone et le signal vidéo provenant d'une caméra vidéo. Les média continus ainsi produits traversent successivement des étapes de filtrage et de compression pour être finalement multiplexés et émis sur le réseau. Il sont reçus du côté du second ordinateur par un organe de démultiplexage qui les décompose en composante audio et vidéo. Ces composantes passent chacune par des étapes successives de décompression et de transcodage avant d'être reconvertie en signaux analogiques audio et vidéo restitués sur un haut parleur et un écran.

4.3.3 Classement par affinités de fonction

Les éléments fonctionnels décrits précédemment peuvent être classés par affinités de fonction sur les flots de média continus au niveau d'un noeud participant à la chaîne de production, traitement, transmission et restitution de l'information sous forme de média continu.

Les sources

Les sources sont des agents de médias continus capables de générer un ou plusieurs média continus. Les sources effectuent l'une des fonctions de lecture à partir d'un support de stockage, de conversion A/N, ou l'une des sous-chaînes suivantes : 1)- conversion A/N + filtrage, 2)- conversion A/N + filtrage + compression, 3)- conversion A/N + filtrage + transcodage + compression ou 4)-conversion A/N + filtrage +transcodage + compression + transmission. D'autres combinaisons restent possibles.

Les puits

Les puits ont une fonctionnalité inverse de celle des sources. Les puits sont des agents de média continus capables d'absorber un ou plusieurs média continus. Les puits assurent l'une des fonctions de stockage, de conversion N/A, ou une des sous-chaînes suivantes : 1)- filtrage + conversion N/A, 2)- décompression + filtrage + conversion N/A, 3)- décompression + filtrage + transcodage + conversion N/A ou 4)- réception + décompression + filtrage + transcodage + conversion N/A.

L'émetteur

L'émetteur est un agent qui effectue la transmission des éléments d'un médium continu d'un ordinateur vers un autre distant.

Le récepteur

Le récepteur réalise la fonction inverse de l'émetteur. Il effectue la réception des éléments d'un médium continu en provenance d'un ordinateur distant.

Le multiplexeur

Le multiplexeur assure uniquement la fonction de multiplexage. Il prend plusieurs média continus synchronisés en entrée et en construit un en sortie.

Le démultiplexeur

Le démultiplexeur assure la fonction réciproque de celle du multiplexeur, c'est-à-dire qu'il reçoit un média continu en entrée et en produit plusieurs synchronisés en sortie.

Le synchronisateur

Le synchronisateur assure uniquement la fonction de synchronisation entre média continus. Il accepte en entrée un ou plusieurs flux et produit en sortie le même nombre de média mais synchronisés.

Les processeurs

Les processeurs sont des entités programmables capables d'effectuer des traitements algorithmiques configurables. Ces traitements sont par exemple les algorithmes de traitement de signal tels que la reconnaissance de forme, ou la détection de présence.

Conclusion

Les entités fonctionnelles que nous avons identifiées ont des comportements similaires et doivent être gérées de façon similaire dans une chaîne de production-restitution de l'information sous forme de média continus. Ce qui va nous conduire à définir l'objet **agent de média continus**. De plus, d'autres objets sont spécifiés pour faciliter l'intégration des média continus à MMS. On obtient ainsi la liste des objets suivante :

1. l'objet **agent de média continus**,
2. l'objet **médium continu**,

3. l'objet **programme**,
4. l'objet **port**,
5. l'objet **connecteur**,
6. l'objet **noeud**,
7. et l'objet **session**.

La spécification de ces objets est faite dans le modèle objet de la méthode OMT [49]. Les notations **n+** ou **n** précédant une classe d'objet associé dans les tables de spécification indiquent que l'objet spécifié est associé à exactement **n** (**n**), ou à au moins **n** (**n+**) objets associés.

4.3.4 L'objet agent de média continus (CMA)

Nous définissons une classe d'objet générique, la classe **agent de média continus (CMA)** qui reflète les propriétés similaires des différentes unités fonctionnelles identifiées dans la chaîne production-restitution de média continus. les classes d'objet **source**, **puits**, **processeur**, **multiplexeur**, **démultiplexeur**, **émetteur**, **récepteur**, et **synchronisateur** seront dérivées de cette classe.

Un objet de classe agent de média continus doit garantir le synchronisme des éléments constitutifs de média continus nécessaire à la qualité de service de l'information représentée. Pour cette raison, il comporte un objet de classe **ReferenceClock** qui cadence la réalisation périodique de ses opérations fonctionnelles. Les média continus rentrent et sortent d'un objet de classe agent de média continus par l'intermédiaire d'objets de classe **CMPort** (voir la figure 4.3).

Les attributs de l'objet agent de média continus

CMA-Identifier : contient l'identificateur d'une instance d'objet CMA.

Rate : contient la valeur en nombre de cycles par seconde de la fréquence des traitements de l'agent de média continus. (Il est égal à l'attribut **Rate** de son objet **ReferenceClock**).

CMA
Hérite de :
Est associé à :
Possède : 0+ CMPorts, 1 ReferenceClock, 1 StatusQueue, 1 ControlQueue
Attributs CMA-Identifier,Rate, Status, ListOfCMPorts
Opérations create, delete, start, stop, setRate, getAttributs, getStatus, setStatus, addCMPorts, connectCMPorts, disconnectCMPorts, removeCMPorts, startCMPorts, stopCMPorts, alterCMPortMode, getCMPortMode

Figure 4.3 : La Classe d'objet **agent de média continus**

Status : contient l'état de l'instance d'objet CMA.

Cet attribut peut prendre deux valeurs :

OPERATIONAL : signifie que l'agent de média continus est en bon état de fonctionnement.

NEEDS-SERVICING : signifie que l'agent de média continus est hors service suite à une situation d'exception. Dans ce cas un champ additionnel indique la nature de l'exception.

ListOfCMPorts : contient la liste d'identificateurs d'objets ports définis sur une instance d'objet CMA.

Les opérations de l'objet agent de média continus

create : crée une nouvelle instance d'objet CMA.

delete : détruit une instance d'objet CMA.

start : met en marche une instance d'objet CMA.

stop : arrête une instance d'objet CMA.

setRate : affecte une nouvelle valeur à l'attribut **Rate** d'une instance d'objet CMA.

getAttributes : retourne les atributs **Rate**, **Status** et **ListOfCMPorts** d'une instance d'objet CMA.

addCMPorts : définit de nouveaux ports sur une instance d'objet CMA.

connectCMPorts : lie des objets ports d'une instance d'objet CMA à des objets connecteurs.

disconnectCMPorts : délie des objets ports d'une instance d'objet CMA d'objets connecteurs.

removeCMPorts : élimine des objets ports d'une instance d'objet CMA.

startCMPorts : met en marche des objets ports (émission ou réception) d'une instance d'objet CMA.

stopCMPorts : arrête des objets ports d'une instance d'objet CMA.

alterCMPortMode : modifie le mode de fonctionnement d'un objet port d'une instance d'objet CMA.

getCMPortMode retourne le mode de fonctionnement d'un objet port d'une instance d'objet CMA.

Les opérations getListOfCMPorts, addCMPorts, deleteCMPorts, stopCMPorts, startCMPorts, alterCMPortMode et getCMPortMode sont liées à la gestion des objets CMPort de l'agent de média continus. En particulier, l'opération **addCMPorts** devra tenir compte de la qualité de service de débit requise par le médium continu de chaque CMPort pour en admettre d'autres.

4.3.5 L'objet médium continu (CMedia)

Le médium continu peut être défini comme étant un véhicule d'information continue (évoluant dans le temps). Les exemples de tels types d'information sont : la vidéo, l'audio et l'animation. Le médium continu est alors une suite d'éléments numériques consécutifs ayant une relation de synchronisme entre eux. La spécification sous forme d'attributs et d'opérations de l'objet médium continu est donnée sur la figure 4.4.

CMEDIA
Attributs CMedia-Identifier, Rate, Mode, SegmentSize, CodingFormat, MediaType, Status
Opérations setRate, alterMode, setSegmentSize, getAttributes, create, delete, getStatus, setStatus

Figure 4.4 : La classe d'objet **Médium Continu**

Les attributs de l'objet médium continu

CMedia-Identifier : contient l'identificateur d'une instance d'objet CMedia.

Rate : contient la valeur du débit requis par le médium continu. Cet attribut prendra par exemple la valeur de 221184000 bits/sec pour une vidéo en vraie couleur à 30 images par seconde et de résolution 640x480 (221184000 bits/sec = 640x480 pixels/image x 24 bits/pixel x 30 images/sec).

Mode : contient le mode de transfert du médium continu. Cet attribut est une structure comportant trois champs :

- *Direction* : champ booléen indiquant le sens du médium continu : la valeur TRUE correspond à l'avance et la valeur FALSE au retour.
- *Continuity* : champ booléen indiquant si le médium continu est effectivement continu (TRUE) ou s'il y a des sauts de discontinuité (FALSE).
- *Offset* : champ contenant la valeur du saut (nombre d'éléments intermédiaires non traités) dans le cas de non continuité. C'est un entier non signé.

CodingFormat : entier non signé contenant le type de codage de l'objet médium continu.

SegmentSize : entier on signé contenant la taille en octets de l'élément constitutif de l'objet médium continu.

MediaType : entier non signé contenant le type d'information supportée par le médium continu (AUDIO, VIDÉO, etc.)

Status : entier non signé contenant l'état d'une instance d'objet CMedia.

Les opérations de l'objet médium continu

setRate : affecte une nouvelle valeur à la fréquence de transfert d'une instance d'objet CMedia.

alterMode : modifie l'attribut **Mode** d'une instance d'objet CMedia.

setSegmentSize : modifie l'attribut **SegmentSize** d'une instance d'objet CMedia.

getAttributes : retourne les valeurs des attributs d'une instance d'objet CMedia.

create : crée une nouvelle instance d'objet CMedia.

delete : détruit une instance d'objet CMedia.

getStatus : retourne l'état d'une instance d'objet CMedia.

setStatus : affecte une valeur à l'état d'une instance d'objet CMedia.

Une spécification ASN.1 des services de gestions de l'objet médium continu sera donnée en guise d'exemple plus détaillée de nouveaux services MMS dans le paragraphe 4.4.2.

4.3.6 L'objet programme (DSP-Progam)

Cet objet sera utile pour le fonctionnement des objets processeurs. En effet les algorithmes de traitement de signaux qui se déroulent sur l'objet processeur ne sont pas figés dans le temps. Il faut prévoir la possibilité de modification de ces algorithmes (d'où d'ailleurs le choix de la dénomination de Processeur).

Le processeur exécutera obligatoirement un algorithme contenu dans un objet programme.

La spécification de l'objet DSP-Program est donnée sur la figure 4.5. Il hérite des attributs et opérations de l'objet **ProgramInvocation** du modèle VMD conventionnel. Il est associé à un objet **Result** (non spécifié ici) et qui contient le résultat de l'exécution de l'algorithme de traitement de signal de l'instance d'objet DSP-Program. L'unique opération, l'opération **getResult**, retourne le résultat d'exécution de l'algorithme.

DSP-PROGRAM
Hérite de : ProgamInvocation
Est associé à : 1 + Result
Attributs DSP-Program-Identifier
Opérations getResult

Figure 4.5 : La classe d'objet **DSP-Program**

4.3.7 L'objet port (CMPort)

Les média continus traversent les agents de média continus par l'intermédiaire de ports. La spécification de l'objet **CMPort** est donnée à la figure 4.6.

CMPORT
Possède : 1 CMA
Est associé à : 1 CMedia, 1 CMConnector, 1 CMA
Attributs CMPort-Identifier,Type, Rate, Status
Opérations create, delete, setType, getType, alterMediaMode, getMediaMode, start, stop, getStatus, setStatus

Figure 4.6 : La classe d'objet **CMPort**

Les attributs de l'objet port

CMPort-Identifier : contient l'identificateur d'une instance d'objet CMPort.

Type : contient la valeur du type d'une instance d'objet CMPort. La valeur INPUT (0) indique qu'il s'agit d'un port en entrée, et la valeur OUTPUT (1) indique qu'il s'agit

d'un port en sortie d'un agent de média continus.

Rate : contient la valeur du nombre de cycles d'opérations à effectuer (émission ou réception) sur une instance d'objet CMPort par unité de temps.

Status : contient l'état de fonctionnement d'une instance d'objet CMPort.

Les opérations de l'objet port

create : crée une nouvelle instance d'objet CMPort.

delete : détruit une instance d'objet CMPort.

setType : définit le type d'une instance d'objet CMPort.

getType : retourne le type d'une instance d'objet CMPort.

alterMediaMode : modifie le mode de l'objet CMedia subordonné à une instance d'objet CMPort.

getMediaMode : retourne le mode de l'objet CMedia subordonné à une instance d'objet CMPort.

start : met en marche (émission ou réception de l'objet CMedia subordonné) d'une instance d'objet CMPort.

stop : arrête (émission ou réception de l'objet CMedia subordonné) une instance d'objet CMPort.

getStatus : retourne l'état de fonctionnement dans lequel se trouve une instance d'objet CMPort.

setStatus : affecte une valeur à l'état de fonctionnement d'une instance d'objet CMPort.

4.3.8 L'objet connecteur (CMConnector)

Deux objets CMPorts appartenant à deux agents de média continus communiquent par l'intermédiaire d'un objet **CMConnector**. Un connecteur est une structure de communication entre des objets CMPort. Parce que l'objet CMConnector peut avoir un objet CMPort source et plusieurs objets CMPorts puits, il permet également de supporter la diffusion de groupe. Un connecteur peut exister entre deux agents d'un même noeud

de communication, ou entre deux agents de noeuds distincts. Les données du médium continu émis sur le port en sortie de l'agent émetteur sont reçues sur le port en entrée de l'agent récepteur. La spécification de l'objet **CMConnector** est donnée sur la figure 4.7.

CMCONNECTOR
Possède : 1 Inport, 1+ Outports
Attributs CMConnector-Identifier, Status, ListOfOutPorts
Opérations create, getListOfOutPorts, addOutPorts, removeOutPorts, delete, start, stop, get Status, setStatus

Figure 4.7 : La classe d'objet **CMConnector**

L'objet connecteur est le moyen de connexion utilisé entre un port de sortie d'un agent (Inport) et un ou plusieurs ports d'entrée (OutPorts) d'autres agents.

Les attributs de l'objet connecteur

CMConnector-Identifier : contient l'identificateur d'une instance d'objet CMConnector.

Status : contient l'état d'une instance d'objet CMConnector.

ListOfOutPorts : contient la liste d'identificateurs d'objets ports en sortie d'une instance d'objet CMConnector.

Les opérations de l'objet connecteur

create : crée une instance d'objet CMConnector.

getListOfPorts : retourne la liste d'identifcateurs d'objets ports connectés en sortie sur une instance d'objet CMConnector.

addOutPorts : ajoute un ou plusieurs ports en sortie d'une instance d'objet CMConnector.

removeOutPorts : élimine un ou plusieurs objets ports de la liste des ports en sortie d'une instance d'objet CMConnector (opération inverse de **addOutPorts**).

delete : détruit le port en entrée d'une instance d'objet CMConnector (ce qui signifie la destruction de l'objet CMConnector).

start : met en marche une instance d'objet CMConnector (début d'émission à partir de son port d'entrée et début de réception sur les port de sortie).

stop : met fin à l'émission à partir du port d'entrée d'une instance d'objet CMConnector.

getStatus : retourne l'état d'une instance d'objet CMConnector.

setStatus : affecte une valeur à l'état d'une instance d'objet CMConnector.

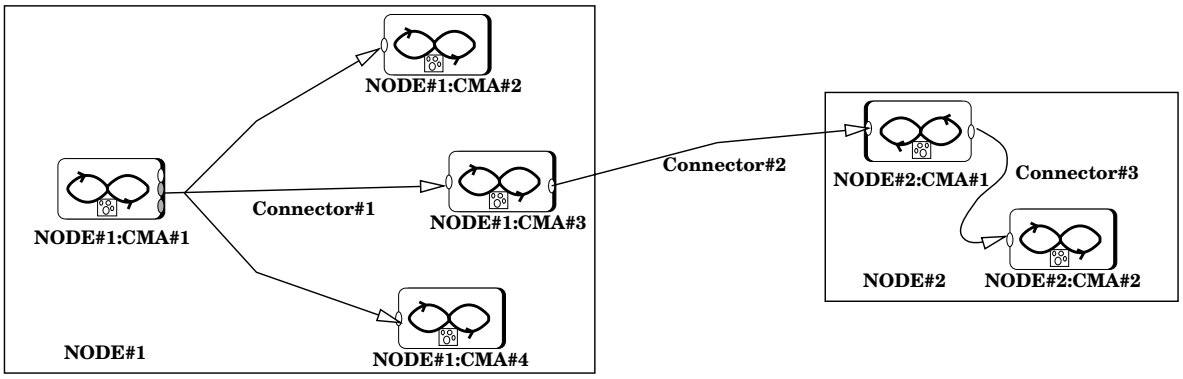


Figure 4.8 : Agents, ports et connecteurs de média continus entre deux noeuds de communication

La figure 4.8 présente un exemple de configuration de communication de média continus entre deux noeuds. Sur le noeud *NODE#1*, un connecteur (*Connector#1*) relie un port en sortie d'un agent de média continus (un InPort pour le connecteur) et trois ports en entrée de trois autres agents de média continus (des OutPorts pour le connecteur). Un second connecteur (*Connector#2*) relie les agents *NODE#1:CMA#3* du noeud *NODE#1* et *NODE#2:CMA#1* du noeud *NODE#2*, effectuant ainsi une communication inter-noeud. Sur le noeud *NODE#2*, un troisième connecteur (*Connector#3*) relie l'agent *NODE#2CMA#1* à l'agent *NODE#2:CMA#2*. Le transport des données de média continus est assuré par des agents émetteurs et récepteurs. Cette possibilité de multiplicité de ports en sortie d'un connecteur permet de supporter la diffusion de groupe dans la transmission de média continus.

4.3.9 L'objet noeud (CMNode)

L'objet noeud est la représentation d'une machine informatique, de son ou ses processeurs (au sens classique du terme), de ses périphériques et de son système d'exploitation. L'objet noeud abrite un ou plusieurs agents de média continus dont les ports sont connectés par des objets connecteurs. Les agents de noeuds différents sont eux aussi connectés par des objets connecteurs.

CMNODE
Possède : 1+ CMA, 0+ CMSession
Attributs CMNode-Identifier, Status, ListOfCMAs, ListOfCMSessions
Opérations getStatus, setStatus, getListOfCMAs, addCMAs, removeCMAs, getListOfCMSessions, addCMSessions, removeCMSessions, start, stop

Figure 4.9 : La classe d'objet **CMNode**

La figure 4.9 donne la spécification de l'objet CMNode.

Les attributs de l'objet noeud

CMNode-Identifier : contient l'identificateur d'une instance d'objet CMNode.

Status : contient l'état logique du noeud, lequel état a la valeur OPERATIONAL (quand le noeud est en état de fonctionnement) ou NEEDS-SERVICING (quand le noeud nécessite une intervention).

ListOfCMAs : contient la liste d'identificateurs d'objets agents de média continus existant sur une instance d'objet CMNode.

ListOfCMSessions : contient la liste d'identificateurs d'objets sessions passant par une instance d'objet CMNode à un instant donné.

Les opérations de l'objet noeud

Les opérations sur les objets noeuds sont celles qui permettent de consulter son état, de le réinitialiser, et de gérer les objets CMAs et CMSessions qui lui sont subordonnés.

getStatus : retourne l'état de fonctionnement dans lequel se trouve une instance d'objet CMNode.

setStatus : affecte une valeur à l'état de fonctionnement dans lequel se trouve une instance d'objet CMNode.

reset : réinitialise une instance d'objet CMNode dans son intégralité.

getListOfCMAs : retourne la liste d'identificateurs d'objets CMAs définis sur une instance d'objet CMNode.

addCMAs : crée de nouveaux objets CMAs sur une instance d'objet CMNode.

removeCMAs : élimine des objets CMAs ayant été définis sur une instance d'objet CMNode.

getListOfCMSessions : retourne la lsite d'identificateurs d'objets sessions passant par une instance d'objet CMNode,

addCMSessions : définit de nouveaux objets CMSessions sur une instance d'objet CMNode.

removeCMSessions : élimine un nombre donné d'objets CMSessions d'une instance d'objet CMNode.

start : met en marche une instance d'objet CMNode dans son intégralité.

start : arrête une instance d'objet CMNode dans son intégralité.

Ces opérations seront utilisées par la fonction exécutive du VMD pour réaliser les requêtes de service MMS d'applications client. Le modèle de noeud est réparti. On spécifie pour cela une classe de noeud maître, la classe (**CM-MASTER-NODE**), qui centralise la gestion des noeuds esclaves (voir annexe 8.2).

4.3.10 L'objet session (CMSession)

Nous définissons l'objet CMSession (figure 4.10) pour décrire l'ensemble des objets agents et connecteurs existant entre un agent source et un ou plusieurs agents puits terminaux. L'objet CMSession est distribué sur les noeuds abritant les objets CMConnectors qu'il utilise. Un objet CMsession assure le transfert de bout en bout d'une information sous forme de médium continu.

CMSESSION
Possède : 1+ CMConnector
Attributs CMSession-Identifier, Status, ListOfCMConnectors
Opérations create, delete, addCMConnectors, getListOfCM-Connectors, removeCMConnectors, start, stop, getStatus, setStatus

Figure 4.10 : La classe d'objet **CMSession**

Les attributs de l'objet session

CMSession-identifier : contient l'identificateur d'une instance d'objet CMSession dans l'environnement de communication de média continus.

Status : contient l'état logique de la partie locale d'une instance d'objet CMSession.

ListOfCMConnectors : contient la liste d'identificateurs d'objets connecteurs locaux qui constituent une instance d'objet CMSession.

Les opérations de l'objet session

create : crée une nouvelle instance d'objet CMSession,

delete : détruit une instance d'objet CMSession existant.

addCMConnectors : ajoute les objets CMConnectors de la liste donnée en paramètre à une instance d'objet CMSession.

getListOfCMConnectors : retourne la liste d'identificateurs d'objets CMConnectors locaux d'une instance d'objet CMSession.

removeCMConnectors : supprime les objets CMConnectors locaux de la liste donnée en paramètre d'une instance d'objet CMSession.

start : met en marche une instance d'objet CMSession.

stop : arrête une instance d'objet CMSession.

getStatus : retourne l'état d'une instance d'objet CMSession.

setStatus : affecte une valeur à l'état d'une instance d'objet CMSession.

Remarquons que l'objet CMSession hérite de ses objets subordonnés CMConnectors la possibilité de communication de groupe. Sur l'exemple de la figure 4.8, une session est créée entre la source terminale *NODE#1:CMA#1* du noeud *NODE:#1* et le puits terminal *NODE#2:CMA#2* du noeud *NODE#2*. Cette session est constituée des connecteurs *Connector#1*, *Connector#2* et *Connector#3*.

Conclusion

Nous avons proposé un modèle objet des éléments constitutifs de la chaîne de génération-restitution de média continus avec garantie de qualités de service.

Les objets qui ont été retenus sont :

1. l'**agent de média continus (CMA)**,
2. le **médium continu (CMedia)**,
3. le **programme de traitement de signaux (DSP-Program)**,
4. le **port de média continus (CMPort)**,
5. le **connecteur de média continus (CMConnector)**,
6. le **noeud (CMNode)**,
7. et la **session de média continus (CMSession)**.

La récapitulation de cette description dans le modèle objets OMT se trouve en annexe 8.2 du rapport.

Ces objets vont être utilisés dans la proposition d'extension de MMS qui va être présentée maintenant.

4.4 Le modèle VMD multimédia (MM-VMD) : approche II

Les objets relatifs aux média continus (CMA, CMedia, DSP-Program , CMPort, CM-Connector, CMNode, CMSession) définis dans le paragraphe précédent seront intégrés à MMS par des représentants virtuels. Nous les incluons ici au modèle MMS et spécifions des objets complémentaires propres à ce dernier. Le résultat est un modèle VMD multimédia réparti. Nous terminons le paragraphe par une spécification ASN.1 de nouveaux services de gestion de l'objet CMedia du modèle MM-VMD. Une brève description des autres nouveaux services MMS est donnée par la suite.

4.4.1 MMS revisitée

Avec l'intégration des média continus, certains aspects du modèle VMD conventionnel doivent être revus. C'est le cas notamment de l'objet journal, de l'événement qui peut être issus de média continus, du nommage des objets et de la répartition du MM-VMD.

Le journal multimédia (MMJournal)

Les journaux sont utilisés pour stocker des enregistrements datés de contenus de variables étiquetées, des commentaires d'utilisateurs ou des combinaisons d'événements et de contenus de variables étiquetées. Les entrées de journal contiennent une étiquette de temps qui indique la date à laquelle les données dans l'entrée ont été produites. L'objet journal actuel de MMS permet d'enregistrer, et de lire ultérieurement, des informations datées sous forme de média discrets. Le journal multimédia permettra, en plus des média discrets, la journalisation d'information sous forme de média continus de durée finie. À un journal multimédia sont associés un agent de média continus de type source qui est chargé de faire la lecture des composantes média continus des entrées du journal,

et un agent de média continus de type puits chargé d'effectuer l'enregistrement de ces composantes.

MMJOURNAL
Hérite de : L'objet journal conventionnel
Possède : 1 puits, 1 source
Attributs ListOfCMedia,
Opérations intitCreateCMedia, startCreateCMedia, pauseCreateCMedia, endCreateCMedia, intitGetCMedia, startGetCMedia, pauseGetCMedia, endGetCMedia, deleteCMedia, getListOfCMedia

Figure 4.11 : La classe d'objet **Journal Multimédia**

L'objet journal multimédia (figure 4.11) hérite des attributs et opérations de l'objet journal conventionnel. Il dispose en plus d'un attribut **ListOfCMedia** contenant la liste d'identificateurs d'objets média continus dans le journal.

Les opérations additionnelles de l'objet MMJournal sont :

intitCreateCMedia : initialise la création d'une entrée médium continu sur une instance d'objet MMJournal (création de la session correspondante dont le point terminal récepteur se trouve sur le puits d'une instance d'objet MMJournal).

startCreateCMedia : démarre l'émission à partir du port générateur de la session créée.

pauseCreateCMedia : arrête momentanément l'émission à partir du port terminal générateur de la session créée.

endCreateCMedia : met fin définitivement à l'émission à partir du port terminal générateur de la session créée.

intitGetCMedia : initialise la lecture d'une entrée médium continu d'une instance d'objet MMJournal (création de la session support dont le point terminal émetteur se trouve sur l'objet source de l'objet MMJournal).

startGetCMedia : démarre l'émission à partir de la source d'une instance d'objet MMJournal (vers un puits quelconque de l'environnement de communication de média continus).

pauseGetCMedia : arrête momentanément l'émission d'une entrée médium continu à partir de la source d'une instance d'objet MMJournal.

endGetCMedia : met fin définitivement à l'émission d'une entrée de médium continu par une instance d'objet MMJournal.

deleteCMedia : élimine une entrée médium continu d'une instance d'objet MMJournal.

getListOfCMedia : retourne la liste d'objets CMedia entrées d'une instance d'objet MMJournal.

Le chapitre 5 présentera un prototype d'implémentation des aspects média continus du journal multimédia.

Événements issus de média continus

MMS ne décrit pas explicitement comment sont évaluées les conditions événementielles qu'une application client pourrait désirer contrôler. Par exemple, dans une application de contrôle de procédé, il est fréquent pour un système de contrôle de générer une alarme quand la variable du procédé (ex. la température) dépasse une certaine valeur initialement fixée appelée la limite supérieure d'alarme. Dans une application de distribution d'énergie une alarme pourrait être déclenchée quand la différence d'angles de phase du courant et de la tension d'une ligne dépasse un certain nombre de degrés initialement fixé. Cette liberté de choix ne peut que faciliter l'exploitation de l'information sous forme de média continus. En effet, la prise en compte de résultats d'algorithmes de traitement de signal exécutés par un objet agent de média continus de type processeur est facilitée. Il s'agit d'algorithmes tels que la reconnaissance de forme, la détection de contour, la détection de présence ou la détection de mouvement dans une séquence d'images.

Le nommage des objets

Il est nécessaire de doter le modèle d'un mécanisme d'identification efficace. MMS utilise la notion de portée pour identifier les objets de son environnement. Il est donc important de définir les portées des nouveaux objets introduits.

- Un objet CMA est soit de portée AA-Specific, soit de portée MM-VMD-Specific :
  - Il est de portée AA-Specific lorsqu'il est créé par un client MMS sur une association application. Ce sera le cas par exemple lorsqu'un client demandera la création d'un processeur de traitement de signal pour évaluer une quelconque condition événementielle. L'objet CMA ainsi créé doit être détruit en même temps que l'association sur laquelle il a été créé.
  - L'objet CMA est de portée MM-VMD lorsque son existence est liée à celle du MM-VMD tout entier. Ce sera le cas par exemple lorsqu'un agent de média continus servira de source de flots de média continus pour plus d'un client MMS.
- Un objet CMPort est de portée CMA-Specific.
- Un objet CMConnector est de portée CMSession-Specific.
- Un objet CMSession est de portée AA-Specific.
- Un objet CMedia est de portée AA-Specific, MM-VMD-Specific ou MMJournal-Specific.

Le MM-VMD est réparti

Tel qu'il a été décrit, le modèle du MM-VMD est nécessairement réparti [1]. En effet, le modèle MM-VMD utilise des objets agents qui traitent des flots pouvant être issus de n'importe quel noeud du système de communication. Cette distribution est surtout marquée par l'objet CMSession qui est construit sur des objets connecteurs appartenant à des noeuds distincts ou connectant des agents situés sur des noeuds différents. De plus, la centralisation des objets nouveaux relatifs aux média continus sous forme de MM-VMD réparti permet une meilleure gestion de ces objets et offre un jeu minimum de services de contrôle et de gestion. Un tel modèle réparti laisse aussi la liberté aux implémenteurs du MM-VMD de choisir le protocole de communication inter-noeuds, ainsi que leur politique de garantie de qualité de service pour les média continus. Si les objets relatifs aux média continus n'étaient pas ainsi regroupés, les clients créant une session seraient submergés par un grand nombre d'associations applications et de requêtes de service nécessaires à sa gestion. On pourrait, dans ce cas, être amené à définir une notion d'association

[1] Ceci rejoint les préoccupations de Dakoury Y. et Elloy J.P. dans leur approche d'extension de MMS pour un VMD multi-serveur [8]

multipoint [2] qui mette l'application client en relation directe avec les VMD contenant un sous-ensemble des agents qu'elle utilise dans sa session. Le modèle du MM-VMD réparti est schématisé sur la figure 4.12. Nous mettons en évidence le fait que les objets contenus dans le MM-VMD puissent être localisés sur des noeuds différents [3]. Ce modèle inclut les objets relatifs aux média continus que nous avons retenus. Le modèle objets OMT du MM-VMD est donné en annexe 8.3.

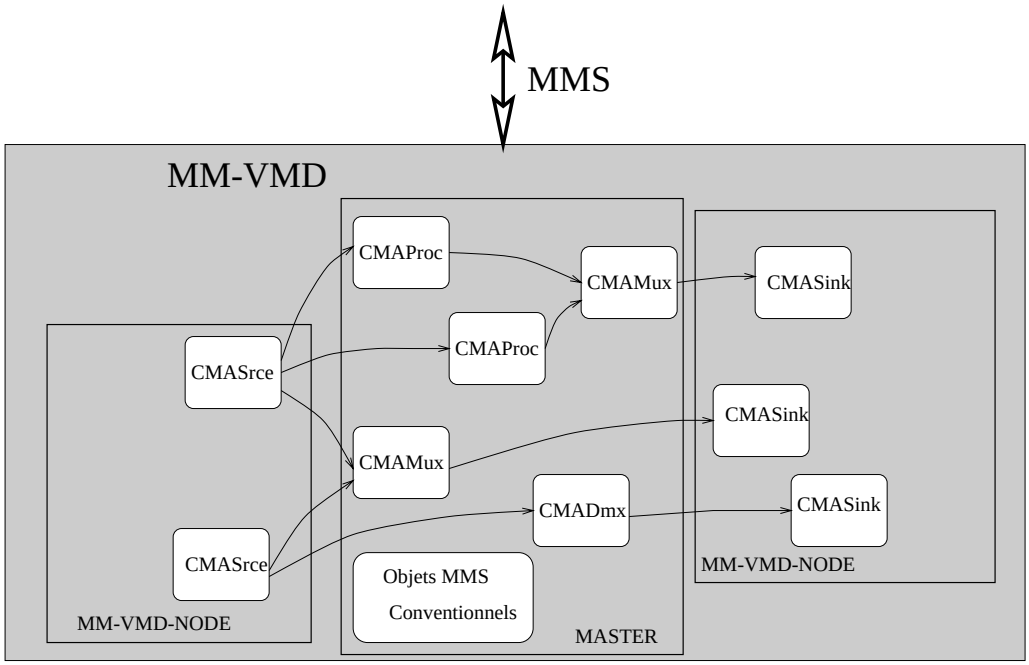


Figure 4.12 : Le modèle du MM-VMD est réparti

Conclusion

Les objets CMA, CMedia, DSP-Program, CMPort, CMConnecteur, CMNode, CM-Session, et MMJournal sont introduits en plus des objets conventionnels de l'équipement

[2] Nous avons défini à ce propos [38], élargissant le champ des qualités de service de la communication temps-réel point-à-point, la qualité de service de cohérence spatiale dans une diffusion de groupe temps-réel. Il est question de pouvoir garantir que le message d'une application qui utilise une telle connexion multipoint, soit reçu aux différents récepteurs à des différences de dates que l'on peut borner.

[3] Nous présentons en annexe 8.1 le modèle objets OMT du VMD réparti. Chaque noeud VMD (*VMD-NODE*) possède une interface de communication MMS supportant la communication avec les clients MMS, et une interface de communication inter-noeud assurant la communication entre noeuds pour la gestion ou le contrôle d'objets répartis. Un noeud maître, le (*VMD-MASTER-NODE*), assure la gestion et le contrôle des noeuds qui lui sont subordonnés.

virtuel de production (VMD). Le résultat est un équipement virtuel multimédia de production réparti, le MM-VMD. Il est constitué de noeuds MM-VMD-NODE mis sous l'autorité d'un noeud maître, le MM-VMD-MASTER-NODE. L'intégration des abstractions de ces nouveaux objets dans la messagerie MMS implique la spécification de nouveaux services, ainsi que d'un nouvel environnement de communication.

4.4.2 Les services de gestion de média continus

En plus des 87 services définis par le messagerie MMS, il est nécessaire de définir une autre gamme de services de gestion des objets CMA, CMedia, DSP-Program, CMPort, CMNode, CMSession et MMJournal. La fonction opératoire s'aidera des opérations de ces objets pour réaliser les services requis par les clients. Nous donnons ici une spécification ASN.1 de la famille de services confirmés de gestion d'objets CMedia. Une description verbale des autres familles de services sur les nouveaux objets est donnée par la suite.

À titre d'exemple de spécification de nouveaux services MMS, nous donnons une spécification ASN.1 de la famille de services de gestion des objets média continus.

SetCMediaRate-Request :

```
SetCMediaRate-Request ::= SEQUENCE
{
  CMediaName [0] IMPLICIT Identifier,
  Rate [1] IMPLICIT Unsigned8
}
```

SetCMediaRate-Response :

```
SetCMediaRate-Response ::= NULL
```

SetCMediaRate-Error :

```
SetCMediaRate-Error ::= CMediaStatus
```

AlterCMediaMode-Request :

```
AlterCMediaMode-Request ::= SEQUENCE
{
```

```
    CMediaName   [0] IMPLICIT Identifier,
    Mode    [1] IMPLICIT SEQUENCE
      {
        Direction    [0] IMPLICIT BOOLEAN,
        Continuity    [1] IMPLICIT BOOLEAN,
        Offset       [2] IMPLICIT Unsigned8
      }
  }
```

AlterCMediaMode-Response :

```
  AlterCMediaMode-Response ::= NULL
```

AlterCMediaMode-Error :

```
  AlterCMediaMode-Error ::= CMediaStatus
```

SetCMediaSegmentSize-Request :

```
  SetCMediaSegmentSize-Request ::= SEQUENCE
  {
    CMediaName [0] IMPLICIT Identifier,
    Rate [1] IMPLICIT Unsigned8
  }
```

SetCMediaSegmentSize-Response :

```
  SetCMediaSegmentSize-Response ::= NULL
```

SetCMediaSegmentSize-Error :

```
  SetCMediaSegmentSize-Error ::= CMediaStatus
```

GetCMediaAttributes-Request :

```
  GetCMediaAttributes-Request ::= CMediaName Identifier
```

GetCMediaAttributes-Response :

```
GetCMediaAttributes-Response ::= SEQUENCE
{
        CMediaName [0] IMPLICIT Identifier ,
        Rate               [1] IMPLICIT Unsigned8 ,
        SegmentSize        [2] IMPLICIT Unsigned16,
        Mode               [3] IMPLICIT SEQUENCE
           {
             Direction   [0] IMPLICIT BOOLEAN DEFAULT TRUE,
             Continuity  [1] IMPLICIT BOOLEAN DEFAULT TRUE,
             Offset      [2] IMPLICIT Unsigned8 DEFAULT 0,
           },
        MediaType          [4] IMPLICIT  Unsigned8,
        CodingFormat       [5] IMPLICIT Unsigned8,
}
```

GetCMediaAttributes-Error :

```
GetCMediaAttributes-Error ::= CMediaStatus
```

CreateCMedia-Request :

```
CreateCMedia-Request ::= SEQUENCE
      {
        CMediaName   [0] IMPLICIT Identifier,
        Rate                  [1] IMPLICIT Unsigned8 ,
        SegmentSize           [2] IMPLICIT Unsigned16,
        Mode                  [3] IMPLICIT SEQUENCE
           {
             Direction   [0] IMPLICIT BOOLEAN DEFAULT TRUE,
             Continuity  [1] IMPLICIT BOOLEAN DEFAULT TRUE,
             Offset      [2] IMPLICIT Unsigned8 DEFAULT 0,
           }
        MediaType             [4] IMPLICIT  Unsigned8
          {
              VIDEO (0),
               AUDIO (1),
               AUDIO-VIDEO (2),
               ANIMATION (3)
```

```
        },
        CodingFormat          [5] IMPLICIT Unsigned8
          {
                RAW (0),
                MPEG (1),
                JPEG (2),
                PCM (3),
                ADPCM (4),
                AVI (5),
                BMP (6),
                GIF (7)
          } DEFAULT RAW
      }
```

CreateCMedia-Response :

```
CreateCMedia-Response ::= NULL
 -- Lorsque la définition de l'objet CMedia
 -- s'est effectuée avec succès.
```

CreateCMedia-Error :

```
CreateCMedia-Error ::= CMNodeStatus
 -- L'état de l'objet CMNode devant abriter
 -- le médium continu est retourné en cas d'échec.
```

DeleteCMedia-Request :

```
DeleteCMedia-Request ::= CMediaName Identifier
```

DeleteCMedia-Response :

```
DeleteCMedia-Request ::= NULL
```

DeleteCMedia-Error :

```
DeleteCMedia-Request ::= CMediaStatus
```

4.4.3 Les autres services

Les autres services de gestion d'objets relatifs aux média continus concernent les objets CMA, DSP-Program, CMPort, CMConnector, CMNode, CMSession et MMJournal.

Les services de gestion de l'objet CMA

CreateCMA : Service confirmé permettant la création d'une nouvelle instance d'objet CMA.

DeleteCMA : Service confirmé permettant de détruire une instance d'objet CMA.

StartCMA : Service confirmé de mise en marche d'un agent de média continus.

StopCMA : Service confirmé d'arrêt d'un agent de média continus initialement mis en marche.

SetRateCMA : Service confirmé de modification de la fréquence d'une instance d'objet CMA.

GetAttributesCMA : Service confirmé de demande d'attributs d'une instance d'objet CMA.

GetStatusCMA : Service confirmé de demande de l'état d'une instance d'objet CMA.

SetStatusCMA : Service confirmé d'affectation d'état d'une instance d'objet CMA.

UnsolliocitedStatusCMA : Service non confirmé de rapport d'état d'une instance d'objet CMA.

AddCMPortsCMA : Service confirmé de définition de nouveaux objets ports sur une instance d'objet CMA.

ConnectCMPortsCMA : Service confirmé de connexion d'objets ports d'une instance d'objet CMA.

DisconnectCMPortsCMA : Service confirmé de déconnexion d'objets ports d'une instance d'objet CMA.

RemoveCMPortsCMA : Service confirmé d'élimination d'objets ports d'une instance d'objet CMA.

StartCMPortsCMA : Service confirmé de mise en marche (émission ou réception) d'objets ports d'une instance d'objet CMA.

StopCMPortsCMA : Service confirmé d'arrêt (émission ou réception) d'objets ports d'une instance d'objet CMA.

AlterCMPortModeCMA : Service confirmé de modification de mode de fonctionnement d'un objet port d'une instance d'objet CMA.

GetCMPortModeCMA : Service confirmé de demande du mode de fonctionnement d'un objet port d'une instance d'objet CMA.

Les services de gestion de l'objet DSP-Program

GetResultDSP-Program : Service confirmé de demande de résulats de l'exécution de l'algorithme de traitement de signaux contenu dans une instance d'objet DSP-Program.

Les services de gestion de l'objet CMPort

CreateCMPort : Service confirmé de création d'une instance d'objet CMPort.

DeleteCMPort : Service confirmé de destruction d'une instance d'objet CMPort.

SetTypeCMPort : Service confirmé de définition de type d'une instance d'objet CMPort.

GetTypeCMPort : Service confirmé de demande du type d'une instance d'objet CMPort.

AlterMediaModeCMPort : Service confirmé de changement de mode d'opération d'une instance d'objet CMPort.

GetMediaModeCMPort : Service confirmé de demande du mode d'opération d'une instance d'objet CMPort.

StartCMPort : Service confirmé de mise en marche (émission ou réception) d'une instance d'objet CMPort.

StopCMPort : Service confirmé d'arrêt (émission ou réception) d'une instance d'objet CMPort.

GetStatusCMPort : Service confirmé de demande d'état d'une instance d'objet CMPort.

UnsolicitedStatusCMPort : Service non confirmé de rapport d'état d'une instance d'objet CMPort.

SetStatusCMPort : Service confirmé d'affectation d'état d'une instance d'objet CMPort.

Les services de gestion de l'objet CMConnector

CreateCMConnector : Service confirmé de création d'une instance d'objet CMConnector.

GetLisOfOutPortsCMConnector : Service confirmé de demande de la liste d'objets CMPort en sortie d'une instance d'objet CMConnector.

AddOutPortsCMConnector : Service confirmé d'adjonction de nouveaux ports en sortie à une instance d'objet CMConnector.

RemoveOutPortsCMConnector : Service confirmé d'élimination de ports en sortie d'une instance d'objet CMConnector.

DeleteCMConnector : Service confirmé de destruction d'une instance d'objet CMConnector.

StartCMConnector : Service confirmé de mise en marche d'une instance d'objet CMConnector.

StopCMConnector : Service confirmé d'arrêt d'une instance d'objet CMConnector.

GetStatusCMConnector : Service confirmé de demande d'état d'une instance d'objet CMConnector.

UnsolicitedStatusCMConnector : Service non confirmé de rapport d'état d'une instance d'objet CMConnector.

SetStatusCMConnector : Service confirmé d'affectation de valeur à l'état d'une instance d'objet CMConnector.

Les services de gestion de l'objet CMSession

CreateCMSession : Service confirmé de création d'une nouvelle instance d'objet CMSession.

DeleteCMSession : Service confirmé de destruction d'une instance d'objet CMSession.

AddCMConnectorsCMSession : Service confirmé de création de nouveaux objets CM-Connectors pour une instance d'objet CMSession.

GetListOfCMConnectorsCMSession : Service confirmé de demande de la liste d'objets CMConnectors subordonnés à une instance d'objet CMSession.

RemoveCMConnectorsCMSession : Service confirmé d'élimination d'objets CMConnectors d'une instance d'objet CMSession.

StartCMSession : Service confirmé de mise en marche d'une instance d'objet CMSession.

StopCMSession : Service confirmé d'arrêt d'une instance d'objet CMSession.

GetStatusCMSession : Service confirmé de demande d'état d'une instance d'objet CM-Session.

UnsolicitedStatusCMSession : Service non confirmé de rapport d'état d'une instance d'objet CMSession.

SetStatusCMSession : Service confirmé d'affectation de l'état d'une instance d'objet CMSession.

Les services de gestion de l'objet CMNode

GetStatusCMNode : Service confirmé de demande d'état d'une instance d'objet CMNode.

UnsolicitedStatusCMNode : Service non confirmé de rapport d'état d'une instance d'objet CMNode.

SetStatusCMNode : Service confirmé d'affectation d'état d'une instance d'objet CMNode.

GetListOfCMAsCMNode : Service confirmé de demande de la liste d'objets CMAs subordonnés à une instance d'objet CMNode.

AddCMAsCMNode : Service confirmé de création de nouveaux objets CMAs subordonnés sur une instance d'objet CMNode.

RemoveCMAsCMNode : Service confirmé d'élimination d'objets CMAs d'une instance d'objet CMNode.

GetListOfCMSessionsCMNode : Service confirmé de demande de la liste d'identificateurs d'objets CMSessions partiellement abrités par une instance d'objet CMNode.

AddCMSessionsCMNode : Service confirmé de création de nouveaux objets CMSessions sur une instance d'objet CMNode.

RemoveCMSessionsCMNode : Service confirmé d'élimination d'objets CMSessions d'une instance d'objet CMNode.

StartCMNode : Service confirmé de mise en marche d'une instance d'objet CMNode.

StopCMNode : Service confirmé d'arrêt d'une instance d'objet CMNode.

Les services de gestion de l'objet MMJournal

InitCreateMMJournalEntry : Service confirmé permettant d'initialiser l'entrée d'une séquence de médium continu dans une instance d'objet MMJournal.

StartCreateMMJournalEntry : Service confirmé permettant de démarrer l'enregistrement effectif d'une séquence de médium continu par l'objet puits d'une instance d'objet MMJournal.

PauseCreateMMJournalEntry : Service confirmé permettant d'arrêter momentanément l'enregistrement d'une séquence de médium continu par l'objet puits d'une instance d'objet MMJournal.

EndCreateMMJournalEntry : Service confirmé permettant de mettre fin à l'enregistrement d'une séquence de médium continu par l'objet puits d'une instance d'objet MMJournal.

InitGetMMJournalEntry : Service confirmé permettant d'initialiser la lecture d'une séquence de médium continu de l'objet source d'une instance d'objet MMJournal.

StartGetMMJournalEntry : Service confirmé permettant de démarrer la lecture effective d'une séquence de médium continu d'une instance d'objet MMJournal.

PauseGetMMJournalEntry : Service confirmé permettant d'arrêter momentanément la lecture d'une séquence de médium continu d'une instance d'objet MMJournal.

EndGetMMJournalEntry : Service confirmé permettant de mettre fin à la lecture d'une séquence de médium continu d'une instance d'objet MMJournal.

4.5 Conclusion

Nous avons défini dans ce chapitre le modèle réparti de l'équipement virtuel de production multimédia qui est une extension de l'équipement virtuel de production de la norme MMS. Nous ajoutons aux objets conventionnels de MMS :

- l'objet agent de média continus effectuant des traitements périodiques sur les éléments d'un médium continu,
- l'objet CMPort qui est la frontière de l'agent de média continus avec l'environnement du MM-VMD,
- le connecteur qui connecte les objets CMPorts d'agents de média continus d'un MM-VMD,
- l'objet
  noeud qui gère les ressources des objets relatifs aux média continus sur une machine informatique.
- l'objet session qui représente une chaîne d'agents de média continus et connecteurs par laquelle circule les éléments d'un médium continu.
- l'objet événement multimédia qui donne une abstraction des événements résultant d'algorithmes de traitement de signal sur les média continus et
- l'objet médium continu qui donne une trace du médium continu réel.

Le chapitre 5 est consacré à la présentation d'une plate-forme d'émulation des agents de média continus, des ports, des connecteurs et des média continus dans le but de montrer les possibilités de réalisation du modèle MM-VMD que nous avons défini.

Chapitre 5

Implémentation du journal multimédia

5.1 Objectifs et démarche mise en oeuvre

Après avoir proposé un modèle d'intégration de média continus dans les systèmes automatisés, il est important de le valider et d'en évaluer les performances. Nous ne proposons pas ici une étude de vérification et de validation formelles du modèle d'extension de MMS. Nous traitons plutôt les aspects d'implémentation et d'évaluation de performances d'un objet particulier du modèle, l'objet journal multimédia MMJournal. Après la description d'un exemple de lecture des composantes média continus de l'objet MMJournal, nous décrivons la plate-forme support utilisée pour cet exemple. Nous donnons ensuite la stratégie de gestion des objets du modèle MM-VMD retenus pour l'exemple. Nous étudions alors l'implémentation de l'objet MMJournal multimédia proprement dit dans un environnement de processus et de communications inter-processus. Nous tirons quelques conclusions sur les performances et proposons ensuite le port du modèle MM-VMD sur la carte d'interface de communication ATM à intelligence locale définie dans le cadre du projet RACINES.

5.2 Présentation de l'exemple

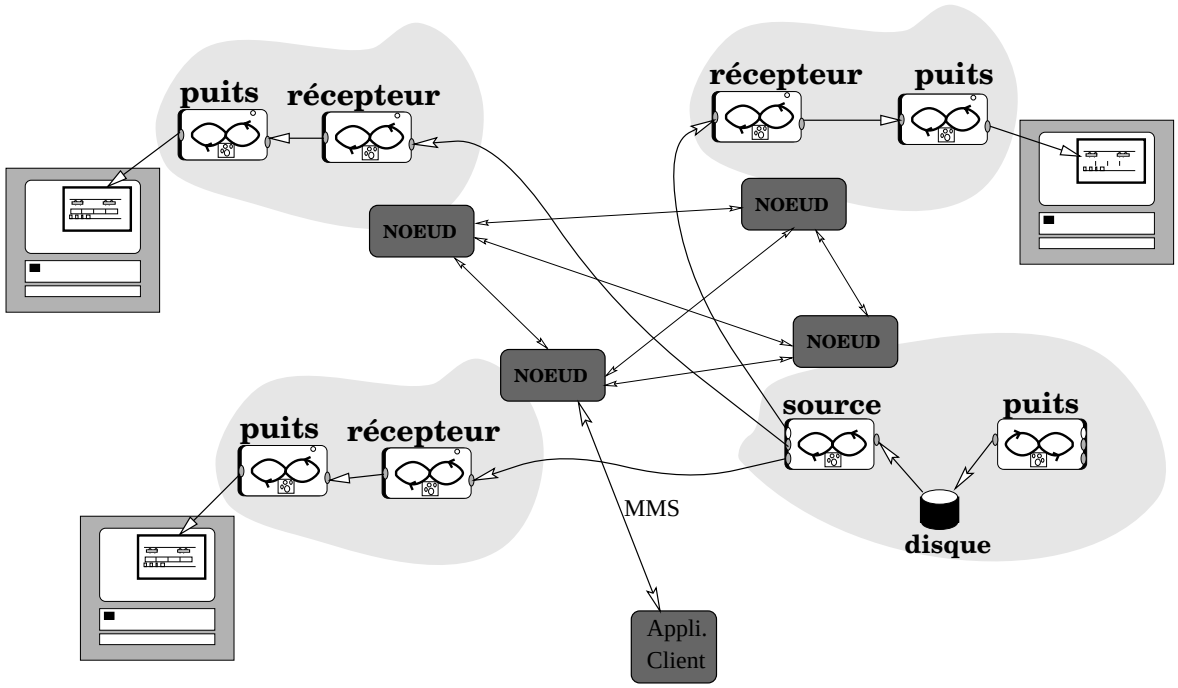


Figure 5.1 : Exemple de consultation du journal multimédia

Imaginons que des séquences de vidéo-audio codées au format MPEG soient stockées dans un objet journal multimédia à la suite d'un événement quelconque sur le site d'un système d'automatisme. L'accès aux composantes MPEG du journal multimédia nécessite des agents de média continus et des connecteurs garantissant les contraintes de débit et de délai relatives à celles-ci. Le modèle MM-VMD est une réponse à ce problème puisqu'il permet la configuration d'agents de média continus et de connecteurs tels que l'illustre la figure 5.1. Les séquences MPEG sont stockées sur le disque dur d'un des noeuds, le noeud serveur de séquences à partir duquel elles seront générées. Les séquences sont restituées sur d'autres noeuds équipés de cartes de décompression MPEG, les noeuds clients de séquences MPEG. Les noeuds sont connectés par une infrastructure de communication temps-réel basée sur ATM. Cette infrastructure supporte l'acheminement des flots de média continus entre le noeud serveur et les noeuds clients de séquences MPEG. Une application enregistrée à l'un des noeuds clients communique par le biais de primitives de service pour la configuration et le contrôle des objets utilisés pour la génération et la restitution des séquences MPEG. Partant du modèle MM-VMD, nous proposons un prototype d'implémentation d'objets agents de média continus et connecteurs utilisés dans la lecture d'un journal multimédia. Nous insistons plus particulièrement sur l'agent source du journal sur lequel reposent tous les aspects de performance de la lecture du journal multimédia.

5.3 Description de la plate-forme cible

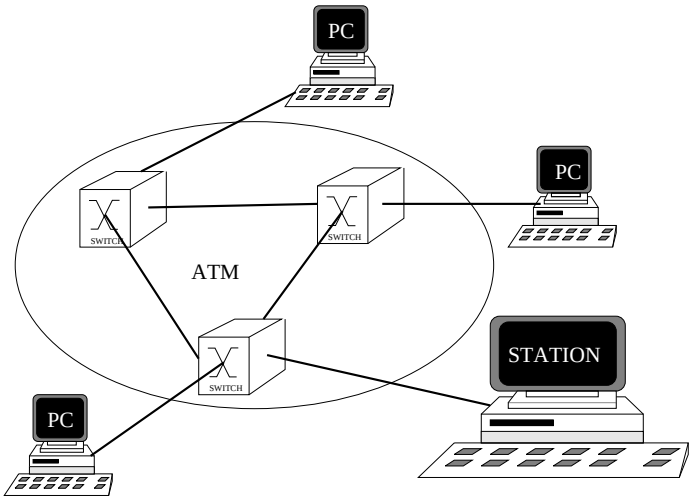


Figure 5.2 : La plate-forme expérimentale

L'objet MMJournal est supporté par la plate-forme expérimentale de la figure 5.2. Elle est constituée d'un infrastruture de communication basée sur ATM, une station HP 9000 série 715/50 Mhz ayant pour système d'exploitation HP-UX 9.0 et des ordinateurs personnels PC ayant Windows NT 3.5 pour système d'exploitation. En plus des cartes d'interface de communication ATM dont dispose chacune des machines, les ordinateurs personnels sont équipés de carte de décompression MPEG permettant la restitution des flots délivrés par la station de travail.

5.4 La gestion des objets virtuels

Le problème de gestion des objets subordonnés au MM-VMD se pose de la même manière que celui du VMD classique. La différence principale se situe au niveau du nombre plus important des objets, du fait qu'ils soient distribués sur des noeuds distincts et que cette distribution doive être transparente aux clients. Il est donc nécessaire de proposer une méthodologie de gestion des objets à travers les noeuds du modèle MM-VMD. Les objets CMA, CMedia, CMPort, CMConnector, MM-VMD-NODE, MM-VMD-MASTER-NODE, CMSession, Association, MMJournal, ainsi que les objets du VMD conventionnel sont gérés dans la globalité du MM-VMD par le mise en place d'une base de données répartie sur ses différents noeuds. Nous nous limiterons ici aux aspects de cette base relatifs à la lecture des composantes média continus de l'objet MMJournal. La figure 5.3 donne une illustration des objets retenus tels qu'ils sont représentés dans la base de données [1].

Dans le symbolisme utilisé, une flèche orientée allant d'une classe d'objet vers une autre indique que les objets de la classe origine de la flèche contiennent une liste d'identificateurs d'objets de la classe extrémité de la flèche, et que les objets de la classe extrémité de la flèche ont également une liste d'identificateurs d'objets de la classe origine de la flèche. Ainsi un objet de classe MM-VMD-MASTER-NODE contient un tableau d'identificateurs d'objets de classe MM-VMD-NODE, de même qu'un objet de classe MM-VMD-NODE contient aussi un tableau d'identificateurs d'objets de classe MM-VMD-MASTER-NODE.

De plus, les chiffres notés à l'extrémité des flèches indiquent le nombre possible d'éléments dans le tableau d'identificateurs d'objets de classe extrémité de la flèche.

Ainsi :

[1] Voir les contraintes de multiplicité du modèle objets OMT de l'annexe 8.3.

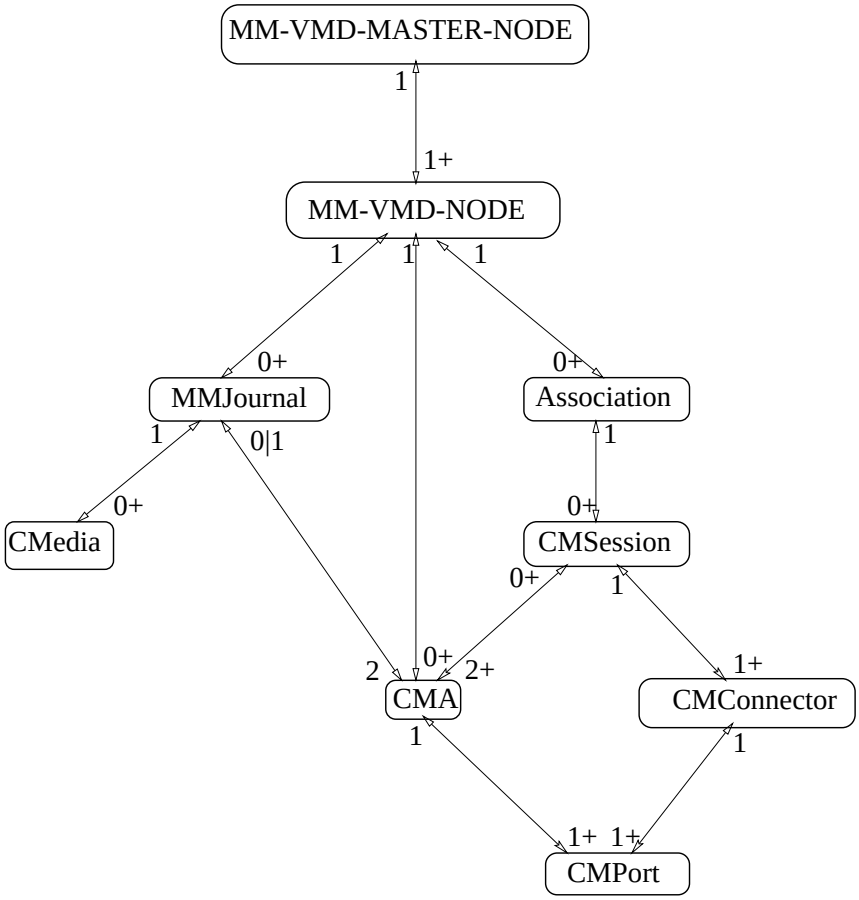


Figure 5.3 : La gestion des objets du modèle du MM-VMD

- la notation **0+** indique que le tableau d'identificateurs d'objets de la classe origine de la flèche vers des objets de classe extrémité de la flèche peut avoir zéro ou plusieurs éléments,
- la notation **0|1** indique qu'un objet de classe origine de la flèche pourra avoir soit zéro, soit exactement un identificateur d'un objet de la classe extrémité de la flèche (optionnel),
- la notation **1+** indique qu'un objet de la classe origine de la flèche pourra avoir un ou plusieurs identificateurs d'objets de la classe extrémité de la flèche,
- la notation **2+** indique qu'un objet de la classe origine de la flèche pourra avoir 2 (ou plus) identificateurs d'objets de la classe extrémité de la flèche,
- la notation **1** indique qu'un objet de la classe origine de la flèche ne pourra avoir qu'un et un seul identificateur d'un objet de la classe extrémité de la flèche.

La figure 5.3 peut être résumée en ces termes :

- Une structure de données représentant un objet MM-VMD-MASTER-NODE contient une liste d'au moins un identificateur d'instance d'objet MM-VMD-NODE.
- Une structure de données représentant un objet MM-VMD-NODE contient :
  - exactement un identificateur d'objet MM-VMD-MASTER-NODE,
  - une liste de zéro ou plus identificateurs d'objets MMJournal,
  - une liste de zéro ou plus identificateurs d'objets Association correspondant à celles des associations applications qui ont été établies sur cette instance d'objet MM-VMD-NODE,
  - une liste de zéro ou plus identificateurs d'objets CMA.
- Une structure de données représentant un objet journal multimédia contient :
  - un unique identificateur d'objet MM-VMD-NODE,
  - une liste de zéro ou plus d'identificateurs d'objets CMedia.
  - une liste de deux identificateurs d'objets CMA, l'un faisant office de source et l'autre office de puits.
- Un objet Association possède :
  - exactement un identificateur d'un objet MM-VMD-NODE,
  - une liste de zéro ou plus d'identificateurs d'objets CMSession.
- Une structure de données représentant un objet CMSession contient :
  - exactement un identificateur d'un objet Association,
  - une liste d'au moins deux identificateurs d'objets CMA du MM-VMD,
  - une liste d'un ou plusieurs identificateurs d'objets CMConnectors qu'il utilise.
- Une structure de données représentant un objet CMConnector contient :
  - exactement un identificateur d'objet CMSession,
  - une liste d'au moins deux identificateurs d'objets CMPort, l'un des objets CM-Port étant le port d'entrée du connecteur, et les autres des ports de sortie.
- Une structure de données représentant un objet CMA contient :

  - exactement un identificateur d'un objet MM-VMD-NODE,
  - zéro ou exactement un identificateur d'un objet MMJournal,
  - une liste de zéro ou plus d'identificateurs d'objets CMSession,
  - une liste d'un ou plusieurs identificateurs d'objets CMPort.

- Une structure de données représentant un objet CMPort contient :
  - exactement un identificateur d'objet CMA,
  - exactement un identificateur d'objet CMConnector.

5.5 Processus et communications inter-processus

Les processus et les mécanismes de communication sont des techniques d'implémentation logicielle des plus répandues dans les systèmes d'exploitation multi-tâche. Un processus peut se définir comme étant une entité de localisation bien déterminée par un système d'exploitation et dont l'exécution est contrôlée par ce dernier. Un processus dispose d'une horloge propre lui permettant de gérer le temps et les événements temporels. Un processus communique avec l'environnement du système d'exploitation par des signaux.

Les mécanismes de communication entre de tels processus sont :

1. la file de messages,
2. la mémoire partagée,
3. le tube nommé,
4. les sémaphores.

Nous décrivons dans ce paragraphe l'implémentation de l'objet MMJournal du modèle MM-VMD suivant les techniques de processus et de communications inter-processus.

L'objet MMJournal est supposé contenu dans un MM-VMD dont nous émulons le fonctionnement par des noeuds client-serveur symétriques coopérant pour la configuration et le contrôle des objets CMSession et CMA.

Après une description du modèle de contrôle du modèle d'émulation, nous présentons l'implémentation des objets relatifs aux média continus, le cas d'une réalisation dans l'environnement IPC du Système V offert par HP-UX 9.0 nous permet ensuite d'avoir une première estimation des performances, et nous terminons par une proposition d'intégration au noyau système.

5.5.1 Le contrôle

Le contrôle concerne tout ce qui relève de la configuration et du contrôle des objets relatifs aux flots de média continus.

Il s'agit de :

1. l'interface application,
2. les services inter-noeud du modèle MM-VMD,
3. le contrôle des objets CMA.

Après une description du modèle d'émulation du MM-VMD, nous présentons les différents aspects du MM-VMD en terme de techniques de processus et communications inter-processus.

La figure 5.4 représente une émulation du contrôle du MM-VMD. La partie gauche du schéma représente la modèle réel de contrôle du MM-VMD, et la partie droite en donne le modèle d'émulation.

On peut remarquer sur cette figure que le processus application client (**Appli**) communique avec le processus noeud du MM-VMD (**Cli-Serv**) par communication inter-processus; cette forme de communication est utilisée en lieu et place de la messagerie MMS. Les noeuds du modèle émulé sont réalisés par des processus au même titre que les processus application; ces noeuds sont des clients-serveurs symétriques. Cette symétrie s'exprime par le fait qu'un noeud doit pouvoir jouer simultanément les rôles de client et de serveur (demander et rendre des services) par rapport aux autres noeuds du MM-VMD. Le protocole inter-noeud est supporté par une infrastructure de transfert entre les ordinateurs hôtes des noeuds (par exemple TCP/IP).

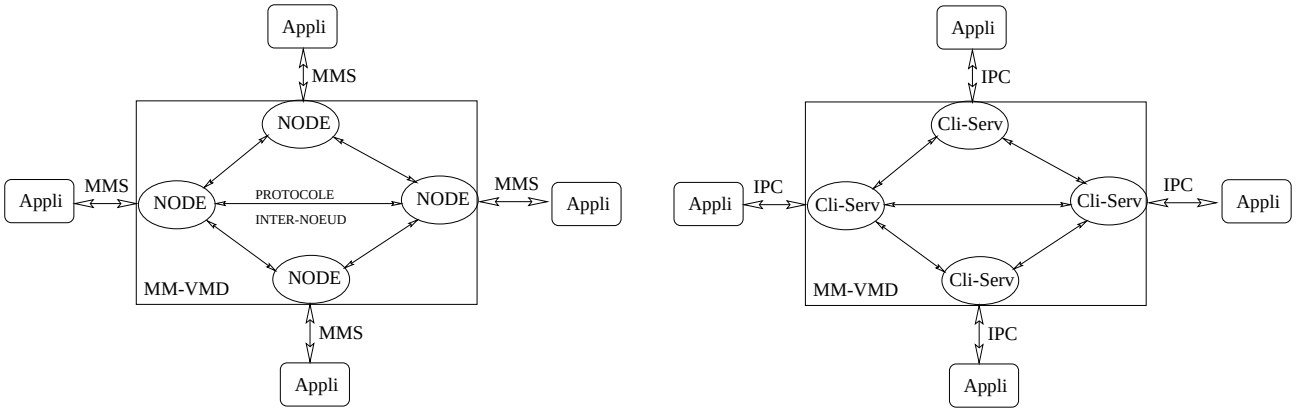


Figure 5.4 : L'émulation du modèle de contrôle du MM-VMD

L'interface application

L'interface application définit la manière dont les applications accèdent aux services du MM-VMD. Elle concerne les primitives de gestion de l'environnement MMS d'une part, et les moyens de communication des primitives de service MMS d'autre part.

PRIMITIVES DE GESTION DE L'ENVIRONNEMENT MMS :

Nous retenons deux primitives de gestion de l'environnement de communication entre l'application et le MM-VMD :

- La primitive d'enregistrement de l'application auprès du MM-VMD :

  `MsgQH RegisterToMM-VMD()`

  Elle permet à l'application qui l'appelle de s'enregistrer auprès du noeud local du MM-VMD. Cet enregistrement est en réalité une émulation de la création d'un objet association application entre l'application et le MM-VMD. Une file de messages est alors associée à l'application. Elle servira de boîte de messages pour les futures communications entre l'application et le MM-VMD. La valeur retournée par la primitive *RegisterToMM-VMD()* est justement l'identificateur de cette file de messages . L'application client envoie dans la file des messages de type requête de service (*Service-Request*) que reçoit le noeud du MM-VMD. Elle y reçoit les messages de type réponse positive (*Service-Reponse*) ou négative (*Service-Error*) qui y sont envoyées par le MM-VMD.
- La primitive de désenregistrement de l'application auprès du noeud local du MM-VMD.

En fin de communication, ou avant la fin d'exécution du programme d'application, celle-ci se désenregistre du MM-VMD au moyen de la primitive :

```
UnregisterFromMM-VMD()
```

Cette primitive libère la file de messages construite par la primitive *RegisterToMM-VMD()*. Elle correspond au service de fermeture d'une association application.

LES PRIMITIVES DE SERVICE DE LECTURE DES COMPOSANTES MÉDIA CONTINUS DU JOURNAL MULTIMÉDIA :

Une fois qu'on sait établir une association-application entre un MM-VMD et une application client, on peut traiter les primitives de service utilisées pour la lecture des composantes média continus d'un journal multimédia.

Il s'agit de :

- Le service confirmé d'initialisation des objets CMA, CMConnector et CMSession :

```
InitGetMMJournalEntry-Request ::= CMediaName Identifier
InitGetMMJournalEntry-Response ::= CMediaName Identifier
InitGetMMJournalEntry-Error ::=  CMSessionStatus Unsigned8
```

- Le service confirmé de démarrage de lecture :

```
StartGetMMJournalEntry-Request ::=  CMediaName Identifier
StartGetMMJournalEntry-Response ::= CMediaName Identifier
StartGetMMJournalEntry-Error ::=  CMSessionStatus Unsigned8
```

- Le service confirmé de demande de pause :

```
PauseGetMMJournalRead-Request ::=  CMediaName Identifier
PauseGetMMJournalEntry-Response ::= CMediaName Identifier
PauseGetMMJournalEntry-Error ::=  CMSessionStatus Unsigned8
```

- Le service confirmé d'arrêt de lecture :

```
EndGetMMJournalEntry-Request ::= CMediaName Identifier
EndGetMMJournalEntry-Response ::= CMediaName Identifier
EndGetMMJournalEntry-Error ::=  CMSessionStatus Unsigned8
```

- Le service non confirmé de retour d'état de l'objet CMSession :

```
SessionInformationReport-Request ::= CMSessionStatus
```

Primitives de services inter-noeud du MM-VMD

- Service confirmé d'initialisation partielle des objets CMA, CMConnector et CMSession :

```
PartialInitGetMMJournalEntry-Request :: =  SEQUENCE
{
MMJournalName [0] IMPLICIT Identifier,
CMediaName [1] IMPLICIT Identifier,
SourceCMPortName [2] IMPLICIT Identifier
DestinationCMPortName [3] IMPLICIT Identifier,
CMSessionName [4] IMPLICIT Identifier
}

PartialInitGetMMJournalEntry-Response
::= CMSession-Identifier Identifier

PartialInitGetMMJournalEntry-Error ::=  SEQUENCE
{
MMJournalStatus [0] IMPLICIT Unsigned8,
MMJournalSourceStatus [1] IMPLICIT Unsigned8,
MM-VMD-NODEStatus [2] IMPLICIT Unsigned8,
CMSessionName [3] IMPLICIT Identifier
}
```

- Plus généralement, service confirmé de réalisation partielle du service_XXX_ (représentant soit Init, Start, Pause, End) :

```
Partial_XXX_GetMMJournalEntry-Request
::=  CMSessionName Identifier

Partial_XXX_GetMMJournalEntry-Response
::= CMSessionName Identifier

Partial_XXX_GetMMJournalEntry-Error ::=  SEQUENCE
{
MMJournalStatus [0] IMPLICIT Unsigned8,
MMJournalSourceStatus [1] IMPLICIT Unsigned8,
```

```
MM-VMD-NODEStatus [2] IMPLICIT Unsigned8,
CMSessionName [3] IMPLICIT Identifier
}
```

- Service non confirmé de rapport d'état partiel de l'objet CMSession :

```
PartOfSessionInformationReport-Request ::= SEQUENCE
{
MMJournalStatus [0] IMPLICIT Unsigned8,
MMJournalSourceStatus [1] IMPLICIT Unsigned8,
MM-VMD-NODEStatus [2] IMPLICIT Unsigned8,
CMSessionName [3] IMPLICIT Identifier
}
```

Le contrôle des agents de média continus

Une fois qu'un agent de média continu existe, il faut disposer d'un moyen de communication entre le noeud dans lequel il se trouve et lui. Tout comme pour l'interface application, il y a une file de messages entre le processus noeud et chaque processus agent de média continus sous son autorité. Le noeud y poste des messages de type *Control-Request* qui sont reçus par l'agent de média continus, et l'agent y poste des messages de types *Control-Response*, *Control-Error* et *StatusReport-Request* reçus par le noeud. Les messages de type *Control-Request* codent toutes les opérations de l'objet CMA définies au chapitre 4 (lors de la spécification des objets relatifs aux média continus), à l'exception des opérations de création.

Il s'agit des opérations : delete, start, stop, setRate, getAttributs, getStatus, setStatus, addCMPorts, connectCMPorts, disconnectCMPorts, removeCMPorts, startCMPorts, stopCMPorts, alterCMPortMode, et getCMPortMode.

Les messages de type *StatusReport-Request* postés par les agents de média continus permettent à ces derniers de rapporter les conditions d'exception survenues au cours de leur fonctionnement.

5.5.2 Les média continus

Les flots de média continus du journal sont transmis par des objets CMA, CMPort et CMConnector. Nous présentons dans ce paragraphe la réalisation en termes de processus et de communications inter-processus de ces objets. Nous proposons également une approche de réservation de ressources pour la garantie de qualités de service.

Les agents de média continus

Les agents de média continus du modèle émulé sont réalisés en logiciel; ce sont des processus créés par les noeuds du MM-VMD.

Un objet CMA est construit sur la base d'autres processus subordonnés assurant les services des ports. Ils se chargent également du protocole de contrôle avec le noeud hôte et de la garantie des qualités de service.

L'agent de média continus traduit les commandes du noeud par des opérations sur les processus serveurs des ports et sur des affectations de variables d'état globales. Les processus serveurs de ports rapportent les exceptions à l'agent par l'intermédiaire de signaux et de variables d'état globales.

Le contrôle d'admission

Nous avons vu que l'information sous forme de média continus exigeait un synchronisme entre les traitements des éléments consécutifs les constituant. Avant la mise en service d'un noeud nous effectuons un étalonnage permettant de déterminer sa limite de charge admissible. A chaque agent de média continus abrité par un noeud est associée une proportion de sa limite admissible de charge. Le contrôle d'admission de nouveaux ports sur les agents de média continus est par la suite régie par leur proportion de charge admissible. Notons que la limite de charge d'un agent comprend aussi bien les fonctions de contrôle que les services des ports.

Nous définissons la notion de *quantum de traitement* qui est la durée maximale nécessaire au service d'un port. La limite de charge maximale des agents est donnée en unités de quantum de traitement. Pour chaque agent de média continus, une unité de quantum de traitement est réservée aux traitements de contrôle. La différence de cette unité du

total de charge admissible donne le nombre maximal de ports qui peuvent être servis par l'agent.

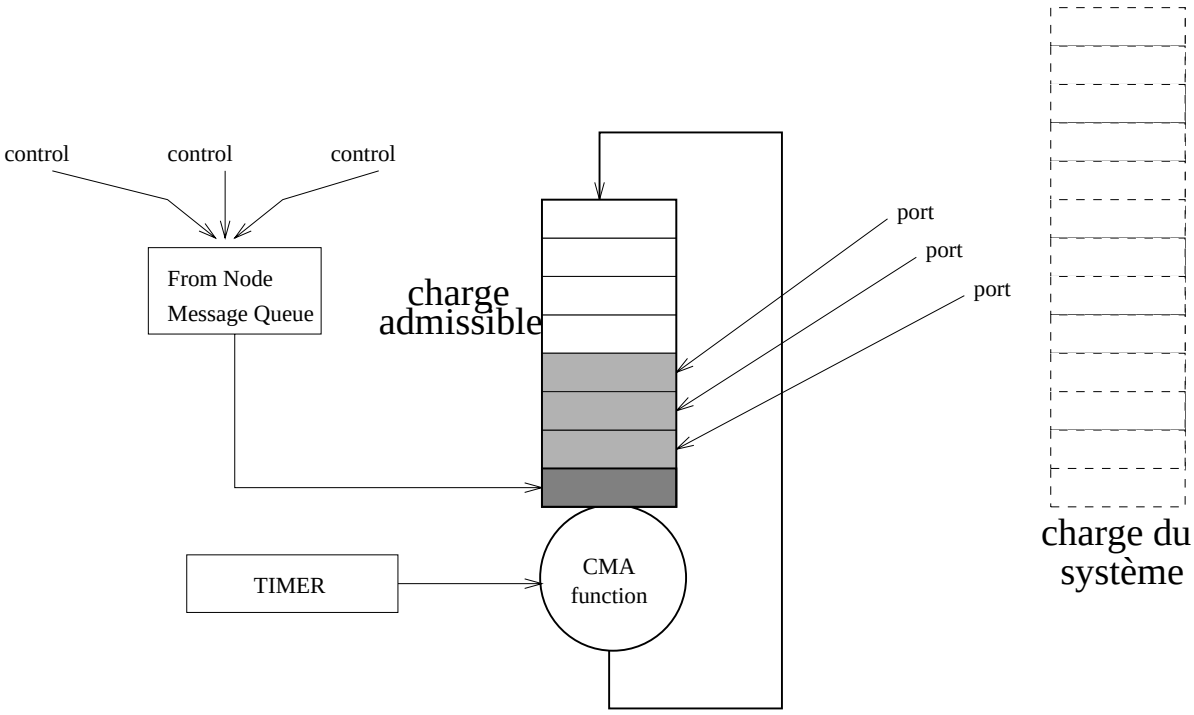


Figure 5.5 : Estimation des ressources et contrôle d'admission

La figure 5.5 illustre cet algorithme de réservation de ressources régissant le contrôle d'admission. Sur cette figure, la puissance totale du système abritant l'agent de média continus est estimée à 13 unités de quantum de traitement. Huit de ces unités de quantum de traitement sont réservées à l'agent de média continus, qui en réserve une pour les traitements de contrôle. Il en reste 7 qui peuvent être utilisées pour le service des ports acceptés. Dans l'exemple, trois ports ont été acceptés et l'agent peut encore en accepter quatre autres.

Les objets CMConnector et CMPort

Dans le cas d'agents appartenant au même noeud, le connecteur est réalisé par un mécanisme de communication inter-processus. Le mécanisme retenu ici est celui de la mémoire partagée estimé être le plus rapide des mécanismes de communication inter-processus. Lorsqu'un noeud crée un connecteur entre deux agents (comme les agents récepteur et puits des noeuds de restitution de la figure 5.1), il appelle la primitive système permettant la construction de la structure de mémoire partagée. L'identificateur

de structure de mémoire partagée retourné est alors transmis en paramètre des messages *ConnectPorts-Request* postés aux agents concernés. Lesdits agents s'attachent à cette mémoire partagée et y écrivent ou lisent les données des ports en sortie ou en entrée respectivement.

Dans le cas d'agents situés sur des noeuds distincts (comme l'agent source du journal et les agents récepteurs des noeuds de restitution de la figure 5.1), la primitive de service *PartialInitMMJournalRead-Request* du protocole inter-noeud contient en paramètre les identificateurs des ports source et destination de la session. Chaque noeud concerné recevant cette primitive de service réserve un point d'accès aux services de transport en émission si le noeud abrite le port source de la session, et en réception s'il abrite un port destination de la session. Il passe alors l'identificateur du point d'accès aux services de transport approprié dans son message de commande *ConnectPorts-Request* posté à l'agent de média continus. Les opérations de lecture ou d'écriture des agents se feront sur des points d'accès aux services de transport temps-réel.

5.5.3 Le cas de l'environnement IPC du système V

i Nous venons de décrire une implémentation du journal multimédia basée essentiellement sur les processus, des mécanismes de communication inter-processus et des points d'accès aux services de transport. Nombreux sont les systèmes d'exploitation multi-tâche qui supportent ces mécanismes de base.

Forts des acquis techniques d'une expérience antérieure d'implémentation d'une plate-forme d'acquisition et de distribution d'images brutes dans un environnement MS Windows (plate-forme décrire en annexe 8.4), nous avons réalisé une implémentation des agents source et récepteur de l'objet MMJournal, ainsi que les services d'initialisation, de lecture, de pause et d'arrêt. Une implémentation utilisant les processus et les mécanismes IPC du système V a été réalisée.

L'implémentation comporte quelques milliers de lignes de codes C pour les aspects source et récepteur.

La figure 5.6 schématise l'architecture logicielle en termes de processus et communication inter-processus. Le soucis majeur ayant d'avoir un premier prototype de l'objet MMJournal, il y manque le mécanisme de contrôle d'objets CMA par les noeuds pour être intégralement compatible au modèle MM-VMD, de même que l'interface application est

remplacée par une simple interface par laquelle un utilisateur peut rentrer les commandes d'initialisation de la session, de lecture, de pause et d'arrêt.

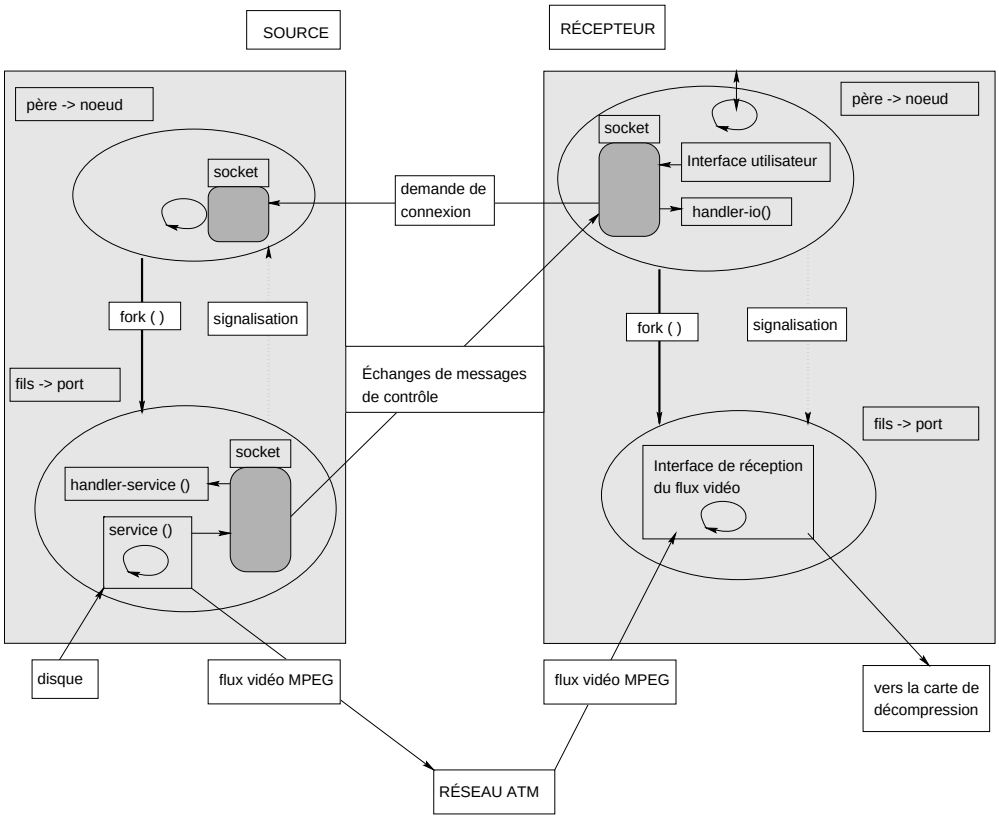


Figure 5.6 : Architecture logicielle d'un noeud abritant une source et d'un noeud abritant un récepteur

Le noeud abritant la source boucle en permanence à l'écoute de demandes de connexions issues de noeuds abritant des récepteurs. La création d'un port sur la source se traduit par la création d'un processus fils par le processus noeud. Ce processus fils établit une connexion de transport rapide avec l'adresse et le numéro de port utilisés par le récepteur vers lequel le flot MPEG sera émis. Il hérite également du noeud la **socket** créée lors de l'acceptation de la demande de connexion. Le processus port utilisera cette socket pour échanger directement des messages de contrôle avec le noeud abritant le récepteur auquel il est connecté, sans plus passer par son noeud local comme dans le modèle MM-VMD. Ce qui n'est pas très génant dans la mesure où nous recherchons dans cette première version du modèle de l'objet MMJournal les performances en termes de nombre de sessions et pas encore en termes de temps de réponse aux commandes. Le contrôle de la partie source de la session se fait directement via une connexion TCP.

Le noeud abritant le récepteur est directement mis en communication avec le processus

port de la source qui servira le flot. il crée localement un processus fils qui se met en attente de réception des données du flot MPEG. Le processus récepteur est entièrement esclave du noeud qui l'abrite et ne peut que être créé (par fork()) ou détruit (par des signaux).

Les premiers essais d'évaluation de performance se sont faits sur une plate-forme intermédiaire utilisant une infrastructure Ethernet - TCP/IP en lieu et place de l'infrastructure de communication ATM.

Le noeud abritant l'objet MMJournal est une station HP 9000 715/50Mhz sur le système HP-UX offrant un environnement IPC du système V. Nos mesures présentent une saturation du réseau au bout de 6 sessions de lecture d'entrées MPEG-I.

D'autre part, le système HP-UX n'est pas préemptif, donc l'algorithme de contrôle d'admission que nous avons proposé pourrait ne pas être assez robuste pour prévenir la dégradation du service des ports due à des surcharges imprévues. C'est ici que naît l'intérêt des systèmes d'exploitation temps-réel préemptifs. Ils ont pour particularité d'offrir des horloges temps-réel et des processus légers (*threads*) à priorité qui peuvent être préemptés si un autre processus léger de priorité supérieure venait à être actif. Des exemples de systèmes temps-réel préemptifs sont Windows NT [5] (non POSIX 1003.1c et ayant l'avantage d'être également multiprocesseur), ou tout autre système compatible POSIX 1003.1b ou 1003.1c.

5.5.4 L'intégration noyau : une optimisation possible

Le prototype d'implémentation que nous venons de présenter réalise les objets relatifs aux média continus sous forme de processus utilisateur. Cette approche induit de nombreuses copies inutiles de données entre les tampons des espaces d'adressage utilisateur et noyau. C'est notamment le cas lorsqu'un processus d'agent source extrait des données du disque en passant par les tampons du noyau, les recopie dans son espace d'adressage utilisateur, et les émet ensuite sur le réseau en passant une fois de plus par des tampons de l'espace d'adressage du noyau. De telles copies de données peuvent être éliminées en intégrant les objets agents de média continus et les connecteurs dans le noyau système. Cette technique a été d'ailleurs utilisée par Kevin et Pasquale [13] dans l'intégration de *splice*, une sorte de chemin de données optimisé entre les périphériques graphiques, disque dur et réseau.

La figure 5.7 illustre le schéma d'intégration des objets relatifs aux média continus

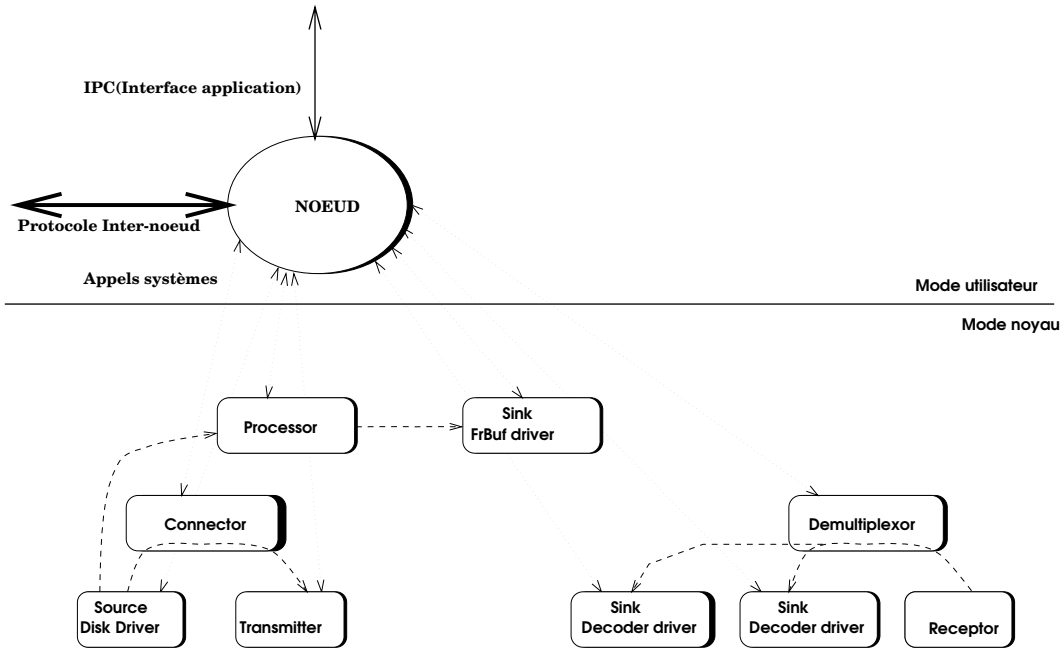


Figure 5.7 : Intégration des objets relatifs aux média continus au noyau système

dans le noyau que nous proposons. Le processus noeud reste sous forme de processus utilisateur. Il en est de même pour l'interface application qui reste sous forme de file de messages entre le processus noeud et le processus application. Par contre l'interface de contrôle entre le processus noeud et les agents de média continus devient un jeux de primitives d'appel système. Les agents de média continus se comportent maintenant comme des sur-couches supplémentaires de pilotes de périphériques dont le contrôle peut être effectué à distance par l'intermédiaire des noeuds du modèle MM-VMD.

5.5.5 Conclusion

Après une description du modèle de contrôle du modèle d'émulation, nous avons présenté l'implémentation des objets relatifs aux média continus. Le cas d'une réalisation dans l'environnement IPC du Système V offert par HP-UX 9.0 nous a permis ensuite d'avoir une première estimation des performances, et nous avons donné le schéma d'une proposition d'intégration au noyau système. Le paragraphe suivant traite de l'intégration du modèle MM-VMD sur une carte d'interface ATM à intelligence locale dans le but d'alléger les traitements effectués par le processeur de la machine hôte.

5.6 Intégration à une carte d'interface de communication

Dans le cadre du projet RACINES il a été conçu un système de communication sur réseaux locaux ATM, et en particulier, une carte d'interface de communication à haut niveau de fonctionnalité pour les systèmes d'automatisme distribués. En plus des communications temps-réel propres à ces systèmes, cette carte devrait permettre l'intégration de média continus. Ayant choisi le modèle du MM-VMD comme la voie d'intégration des média continus aux systèmes d'automatisme, il est intéressant de montrer quelques aspects de l'implémentation du modèle MM-VMD sur cette carte à haut niveau de fonctionnalité. Nous attaquerons cette question par une brève description préalable de la carte. Nous donnerons ensuite l'architecture logicielle du modèle MM-VMD sur celle-ci.

5.6.1 Description de la carte

La carte est conçue pour interfacer des équipements aussi variés que des caméras et moniteurs vidéo, des microphones, des haut-parleurs, des robots, des commandes numériques, des automates programmables et des calculateurs. En bref, elle est conçue dans l'optique d'intégration efficace de l'information sous forme de média continus dans les systèmes d'automatisme distribués. Elle isole entièrement le processeur de la machine hôte de tout trafic de média continus par des chemins directs (DMA ou bus ATM local) entre périphériques et infrastructures de communication. Elle comporte en effet un commutateur ATM embarqué ayant quatre ports d'entrée et quatre ports de sortie à 155,52 Mbit/s chacun. Un couple de ports entrée-sortie sert à la communication avec l'ordinateur hôte, un autre couple à la communication avec une autre carte de même type connectée au même ordinateur hôte, et les deux couples de ports restant servent à la communication avec l'extérieur via des interfaces physiques optiques [2]. Ce double accès vers l'extérieur permet de construire des topologies variées, la plus simple étant une topologie à double anneau. Nous nous contenterons de la topologie double anneau dans l'intégration du modèle MM-VMD puisqu'elle est la topologie de moindre coût permettant à toutes les stations de la configuration de communiquer. Rappelons que les noeuds du modèle MM-VMD ont besoin d'une telle possibilité de communication pour le protocole inter-noeud. La figure 5.8 illustre une vue d'ensemble de l'architecture matérielle de la carte d'interface de communication.

La carte supporte deux cartes filles ; l'une pour l'interface physique au réseau ATM et

[2] Voir [30] pour les détails des choix de conception de cette carte.

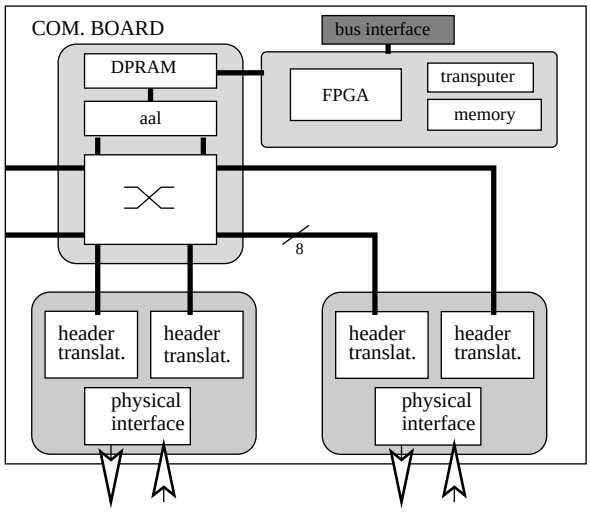


Figure 5.8 : Une vue générale de la carte de communication

l'autre pour le processeur embarqué. La couche d'adaptation à ATM de type 5 (AAL 5), type de fonctionnalité la plus légère, est réalisée en matériel par un circuit spécialisé. La communication entre le composant AAL5 et le processeur embarqué s'effectue au moyen d'une mémoire partagée. Une mémoire à double accès a été retenue afin d'optimiser les transferts sur le bus de la carte. La seconde carte fille contient l'"intelligence" du système. Le processeur retenu est un Transputer connu pour son intégration facile aux cartes d'interface de périphériques. Le composant programmable (FPGA) réalise essentiellement la conversion entre le bus du Transputer et le bus de la carte utilisée en DMA. Le Transputer est en charge de la gestion du réseau et de la signalisation. De plus, il réalise un protocole de transport léger (*sscop*) au dessus de l'AAL5 pour la communication de signalisation. Il s'agit du mode assuré nécessaire à la signalisation ainsi qu'aux transferts avec acquittements.

5.6.2 Intégration du MM-VMD à la carte

Le schéma d'implémentation est identique à celui retenu dans l'environnement des processus et communications inter-processus à la différence que le noeud et les objets subordonnés relatifs aux média continus sont déportés sur le transputer de l'interface de communication (figure 5.9) qui interface également des périphériques producteurs et consommateurs de média continus. Le mode assuré minimal *sscop* au dessus de l'AAL5 permet la transmission des messages du protocole inter-noeud. Le protocole inter-noeud peut être considéré à juste titre comme un protocole de signalisation car il permet de

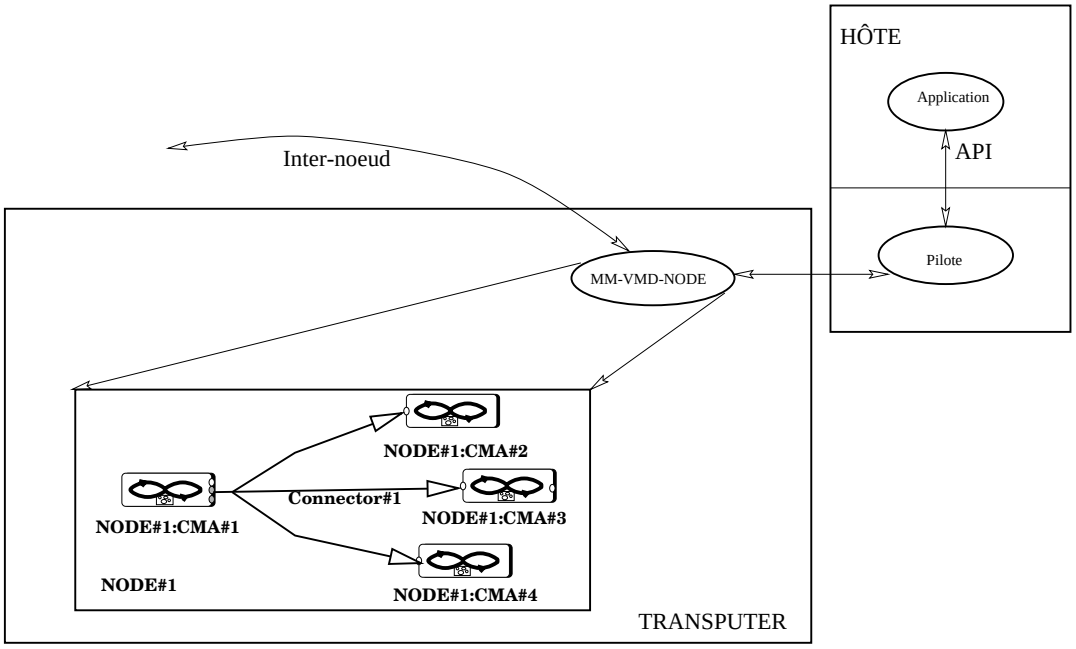


Figure 5.9 : Intégration du modèle MM-VMD à l'interface de communication

configurer et de contrôler les agents et connecteurs relatifs aux média continus. Les flots de média continus entre noeuds sont transmis sur les connexions AAL5 offrant les fonctionnalités minimales nécessaires à ceux-ci. Cette communication est largement favorisée par la topologie à double anneau tolérante à une panne de station que permet le commutateur embarqué de la carte d'interface de communication. De plus, les ressources du transputer sont gérées par un exécutif temps-réel offrant un traitement d'interruptions matérielles flexible, un mécanisme de passage de messages efficace, une horloge temps-réel, une gestion du parallélisme, et des mécanismes de sémaphore et de communication inter-processus. Ces propriétés de l'exécutif temps-réel permettent un comportement "plus contrôlable" des objets relatifs aux média continus, notamment des agents de média continus. Le processeur hôte se trouve ainsi libéré de tous les traitements de gestion et de réalisation des objets relatifs aux média continus. Par contre l'interface application entre les programmes d'application et le modèle MM-VMD se fera à travers un pilote de périphérique intégré au système hôte.

5.7 Conclusion

Ce chapitre nous a permis de présenter un prototype d'implémentation du modèle MM-VMD basé sur l'exemple de lecture des composantes média continus d'un journal multimédia. Le prototype est basé sur les processus et les mécanismes de communication inter-processus. Une réalisation dans l'espace utilisateur du système UNIX a été faite et sert dans un système de distribution et de visualisation de films au format MPEG. Ledit prototype peut aussi bien être réalisé sur d'autres systèmes temps-réel préemptifs comme Windows NT ou tout système temps-réel compatible POSIX 1003.1b ou 1003.1c. Il peut également être intégré à une interface de communication ATM comportant une intelligence locale.

Chapitre 6

Conclusion

Conclusion

L'introduction du multimédia dans les applications distribuées temps-réel nous a conduits à une analyse de MMS mettant en évidence ce qu'il lui manque pour satisfaire les exigences des flots de média continus.

Nous avons proposé une définition d'objets, dans le modèle objets OMT, et des services de gestion de ces derniers dans un environnement MMS sous forme de VMD multimédia réparti : le **MM-VMD**.

Les nouveaux objets de ce MM-VMD sont : l'agent de média continus (**CMA**) effectuant des traitements périodiques sur les éléments d'un médium continu (**CMedia**), le port (**CMPort**) qui est la frontière de l'agent de média continus et l'environnement du MM-VMD, le connecteur (**CMConnector**) qui connecte deux ports d'agents de média continus d'un MM-VMD, la session (**CMSession**) qui représente une chaîne complète de génération-restitution de média continus dans un MM-VMD, le journal multimédia (**MMJournal**) permettant la journalisation de séquences finies de média continus, et l'événement d'origine média continus qui donne une abstraction des événements résultant d'algorithmes de traitement de signal sur les média continus.

Un prototype d'implémentation du journal multimédia dans un environnement de processus et de communications inter-processus a été proposé. Il nous a permis de réaliser certains des objets relatifs aux média continus, de pouvoir les configurer et les contrôler au moyen de nouveaux services MMS dans un environnement UNIX système V (quelques milliers de lignes de C). Les limites de performance observées sont surtout dues à la non disponibilité des cartes d'interface ATM.

D'autre part, conscients du problème de copies multiples entre espaces d'adressage des systèmes d'exploitation monolithiques, nous présentons un schéma d'intégration du modèle au noyau système. De même que nous décrivons l'intégration du modèle MM-VMD à une carte d'interface de communication ATM à intelligence locale dans le but d'alléger le processeur de la machine hôte. Cette carte, actuellement mise sur le marché par ITMI-APTOR, notre partenaire de recherche, est montée sur une plate-forme expérimentale locale.

MMS est la seule norme ouverte de messagerie industrielle et nous croyons que l'intégration des média continus (audio, vidéo et animation) aux systèmes d'automatisme distribués sera facilitée par l'extension de son modèle VMD vers le modèle réparti MM-

VMD. Notre approche a touché quelques aspects de cette extension, allant de la définition des objets à celle des services de gestion, avec une expérimentation d'implémentation d'un journal multimédia.

Ce qu'il nous semble interessant à entreprendre à la suite de ce travail serait une vérification formelle du modèle et de son protocole inter-noeud, de même qu'une implémentation concrète sur la carte d'interface de communication à intelligence locale. Ce qui permettrait à cette proposition d'extension de MMS d'être crédible aux yeux des utilisateurs et des fabricants de produits.

Chapitre 7

Bibliographie

Références

[1] A. Campbell, G. Coulson, F. Garcia, D. Hutchinson, and H. Leopold. Integrated Quality of Service for Multimedia Comunications. In *IEEE INFOCOM'93*, pages 732–739, 1993.

[2] C. J. Parris, G. Ventre, and H. Zhang. Graceful Adaptation of Guaranteed Performance Service Connections. Technical Report TR-93-011, The Intertional Computer Science Institute, Berkeley, CA, 1993.

[3] C. Topolcic. *Experimental Internet Stream Protocol, Version 2 (ST-II)*. Internet RFC 1190, 1990.

[4] John Crowcroft. Inter-Process Communications Paradigms for Distributed Multimedia Conference Control. Technical report, University College London, January 1994. To appear in Journal of Internetworking Research and Experience.

[5] Helen Custer. *Inside Windows NT*. Microsoft PRESS, 1992.

[6] R. Herrwich D. Anderson and C. Schaefer. SRP: A Resource reservation protocol for guaranteed-performance communications in the Internet. Technical report, The International Computer Science Institute, Berkeley, CA, 1991.

[7] D. Ferrari, A. Banerjea, and H. Zhang. Graceful Adaptation of Guaranteed Performance Service Connections. Technical Report TR-92-072, The Intertional Computer Science Institute, Berkeley, Berkeley, CA, 1993.

[8] Dakoury Y. and Elloy J.P. A new Multi-server Concept for the MMS Environment. In *Proc. of the 9th IFAC workshop on Distributed Computer Control Systems*, Tokyo, Japan, 1989.

[9] David L. Tennenhouse, Joel Adam, David Carver, henry Houth, Michael Ismert, Christopher Lindblad, Bill Stasior, David Wetherall, David Batcher and Theresa

Chang. A Software-Oriented Approach to the Design of Media Processing Environments. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, pages 435–444, Boston, Massachusetts, 1994.

[10] Martin de Prycker. *Asynchronous Transfer Mode*. Ellis Horwood, 1991.

[11] Adrian Nye et al. *Xlib Programming Manual - The Definitive Guides to the X Window System, Third Edition*, volume 1. O'Relley and Associates, Inc., 1991.

[12] L. Delgrossi et al. Media Scaling for Audiovisual Communication with the Heidelberg transport System. In P. Venkat Rangan, editor, *Proc. of the ACM Multimedia 93*, pages 99–104, Anaheim, 1993. ACM Press.

[13] Kevin Fall and Joseph Pasquale. Improving Continuous-Media Playback Performance with In-Kernel Data Paths. In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, California, 1994.

[14] T. Fisher. Real-Time Scheduling Support in Ultrix-4.2 for Multimedia Communication. In P. Venkat Rangan, editor, *Network and Operating System support for Audio and Video*, pages 321–327, Berlin Heidelberg, 1993. Springer Verlag.

[15] G. Coulson and G. S. Blair. Micro-Kernel Support for Continuous media in Distributed Systems. *To appear in Computer Networks and ISDN Systems*.

[16] G. Coulson, G.S. Blair, N. Davies, N. Williams. Extensions to ANSA for Multimedia Computing. *Computer Networks and ISDN Systems*, 25:305–323, 1992.

[17] G. Gopal, G. Herman, and M. P. Vecchi. The Touring Machine Project: Toward a Public Network Platform for Multimedia Applications . In *Proceedings of the 8th International Conference on Software Engineering for Telecommunication Systems and Services*, Florence, ITaly, March 1992.

[18] G. Le Lann and N. Rivierre. Real-Time Communications over Broadcast Networks: The CSMA-DCR and the DOD-CSMA-CD Protocols. Technical Report , INRIA, Roquencourt, 1993.

[19] S. J. Gibbs and D. C. Tsichritzis. *Multimedia Programming: Objects, Environments and Frameworks*. Addison-Wesley Publishing Company, 1995.

[20] H. Tokuda, T. Nakajima and P. Rao. Real-Time Mach: Towards a Predictable real-Time System. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1990.

[21] Andrew Herbert. An ANSA overview. *IEEE Network*, 8(1), January/February 1994.

[22] R. G. Herrtwich and L. Delgrossi. Beyond ST-II: Fulfilling the Requirements of Multimedia Communication. In P. Venkat Rangan, editor, *Network and Operating System support for Audio and Video*, pages 321–327, Berlin Heidelberg, 1993. Springer Verlag.

[23] ISO/IEC. *ISO/IEC 9506-5, Industrial Automation Systems - Manufacturing Message Specification - Part 5: MMS Companion Standard for PLC (Draft)*.

[24] ISO/IEC. *ISO/IEC 9506-6, Industrial Automation Systems - Manufacturing Message Specification - Part 6: MMS Companion Standard for Process Industry (Draft)* .

[25] ISO/IEC. *ISO/IEC 9506-1, Industrial Automation Systems - Manufacturing Message Specification - Part 1: Service Definition.* 1990.

[26] ISO/IEC. *ISO/IEC 9506-2, Industrial Automation Systems - Manufacturing Message Specification - Part 2: Protocol Specification*. 1990.

[27] ISO/IEC. *ISO/IEC 9506-3, Industrial Automation Systems - Manufacturing Message Specification - Part 3: MMS Companion Standard for Robotics*. 1991.

[28] ISO/IEC. *ISO/IEC 9506-4, Industrial Automation Systems - Manufacturing Message Specification - Part 4: MMS Companion Standard for Numerical Control*. 1991.

[29] ISO/IEC JTC1/SC29/WG11. *Coding of Moving Pictures and associated audio*. International Organization for Standardization, 1993.

[30] J.-F. Guillaud, Max R. Pokam and Gérard Michel. An ATM-based Multimedia Integrated Manufacturing System. In *Proc. of the 4rth IEEE International Symposium on High Performance Distributed Computing*, Pentagon City, August 1995.

[31] J. Nieh, J. G. Hanko, J. D. Norcutt, and G. A. Wall. SVR4UNIX Scheduler Unacceptable for Multimedia Applications. In *Proceedings of the Workshop on Operating Systems for Audio and Video*, Lancaster, 1993.

[32] J. P. Parsons Jr. Digital video architecture for OS/2 2.1. In A. Rodriguez, M. Chen, and J. Maitain, editor, *Proceedings of IS&SPIE Symposium on Electronic Imaging: Science & Technology, workshop on High-Speed Networking and Multimedia Computing*, pages 19–30, San Jose, February 1994.

[33] Raj Jain. *FDDI Handbook : High-Speed Networking Using Fiber and Other Media*. Addison-Wesley Publishing Company, 1994.

[34] K. Nahrstedt and J. M. Smith. Application-Driven Approach to Networked Multimedia Systems. In *Proceedings of the 18th Conference on Local Computer Networks*, 1993.

[35] K. Nahrstedt and J. Smith. Experimental Study of Issues in End-to-End QoS. Technical report, University of Penylvania, 1994.

[36] L. Delgrossi, R. G. Herrtwich, and F. O. Hoffmann. An Implementation of ST-II for the Heidelberg Transport System. In *IEEE Globecom'92*, Orlando, 1992.

[37] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new Resource Reservation Protocol. *IEEE Network Magazine*, September 1993.

[38] Max R. Pokam and Gérard Michel. Guaranteeing Spatial Coherence in Real-time Multicasting. In *Proc. of the 1995 IEEE International Symposium on Information Theory*, Whistler, B.C. Canada, September 1995.

[39] P. Castori. On-line Access to the MMS Abstract Syntax. Technical Report http://litwww.epfl.ch/~mms/mms_abstract_syntax.html, Swiss Federal Institute of Technology, March 1995.

[40] P. G. S. Florissi. QuAL: Quality Assurance Language. Technical Report CUCS-007-94, Columbia University, New York, February 1994.

[41] Craig Partridge. GIGABIT NETWORKING, October 1993. ISBN, 0-201-56333-9; LC Call Number, TK5105.5P3745 1993.

[42] Max R. Pokam and Gérard Michel. Extended MMS for Multimedia Applications on an ATM based Network. In *1rst COST 237 Project Workshop*, Col-de-Porte, Grenoble, France, March 1993.

[43] Max R. Pokam and Gérard Michel. A Multimedia Application Programming Framework. In *4rth International Conference on Database Systems for Advanced Applications*, Singapore, April 1995.

[44] R. B. Dannenberg, D. B. Anderson, T. Neuendorffer, D. Rubine and J. Zelenka. Performance Measurements of the Multimedia Testbed on Mach 3.0. Experience Writing Real-Time Device Drivers, Servers and Applications. Technical Report #CMU-CS-93-205, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1993.

[45] R. G. Herrtwig. The Role of Performance, Scheduling, and Resource Reservation in Multimedia Systems. In A. Karshmer and J. Nehmer, editor, *Operating Systems of the 90s and Beyond*, pages 361–367. Springer Verlag, 1991.

[46] R. Keller and W. Effelsberg. MCAM: An Application Layer Protocol for Movie Control, Acces and Management. In P. Venkat Rangan, editor, *Proc. of the ACM Multimedia 93*, pages 21–30, Anaheim, 1993. ACM Press.

[47] R. Mackiewicz. An Overview to the Manufacturing Message Specification. Technical Report http://litwww.epfl.ch/m̃ms/mms_IntroSISCO, SISCO, 1994.

[48] Robert F. Mines, Jerrold A. Friesen, Christine L. Yang. DAVE: A Plug and Play Model for Distributed Multimedia Application Development. In , editor, *Proceedings of the 2nd ACM international Conference on Multimedia*, pages 59–66, San Francisco, California, 1994.

[49] J. Rumbaugh et al. *Object-Oriented Modeling and Design*. Prentice-Hall International Inc., 1991.

[50] S. Boecking, S. Damaskos, L. Delgrossi, A. Fartmann, R. Hammeschmidt, G. Hoelzing, I. Milouscheva, and J. Svandvoss. The BERKOM multimedia transport system. In A. Rodriguez, M. Chen, and J. Maitain, editor, *Proceedings of IS&SPIE Symposium on Electronic Imaging: Science & Technology, workshop on High-Speed Networking and Multimedia Computing*, pages 256–265, San Jose, February 1994.

[51] S. Damaskos and A. Gavras. Connection oriented protocols over ATM: a case study. In A. A. Rodriguez, M. Chen, and J. Maitan, editor, *Proc. of IS&T/SPIE Sysposium on Electronic Imaging: Science & Technology, workshop on High-speed Networking and Multimedia Computing*, pages 266–278, San Jose, 1994.

[52] Isil Sebuktekin. *Goodness Definition and Goodness Measure for High Speed Transport Protocols for Lightweight Networking Applications*. PhD thesis, Lehigh University, May 1992.

[53] S. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. In *Proceedings, 1994 SIGCOMM Conference*, pages 47–57, London, UK, August 31st - September 2nd 1994.

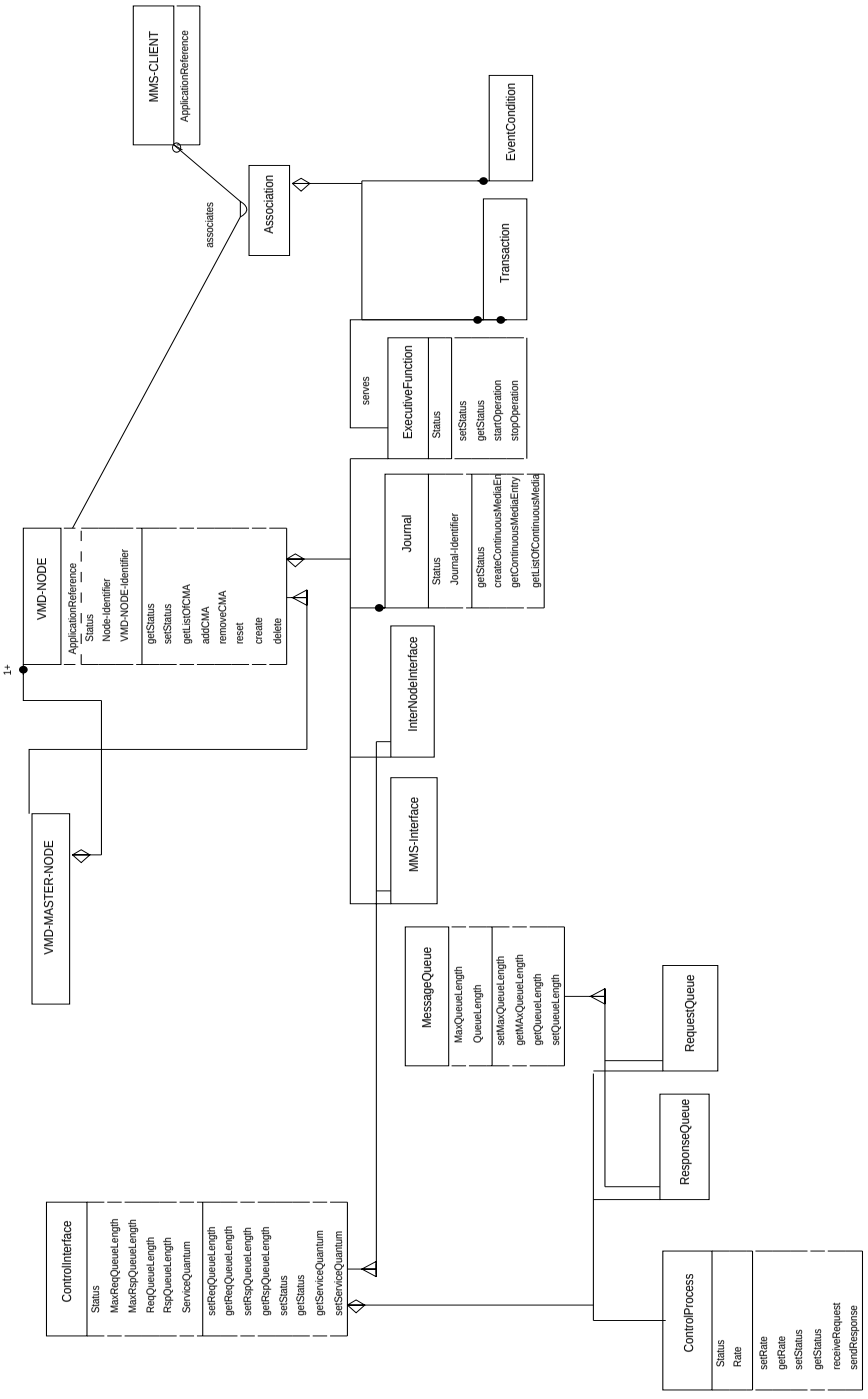
[54] T. Nakajima, T. Kitayama and H. Tokuda. Experiments with realtime Servers in Real-Time Mach. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, 1993.

[55] A. Valenzano, C. Demartini, and L. Ciminiera. *MAP and TOP Communications*. 1992.

Chapitre 8

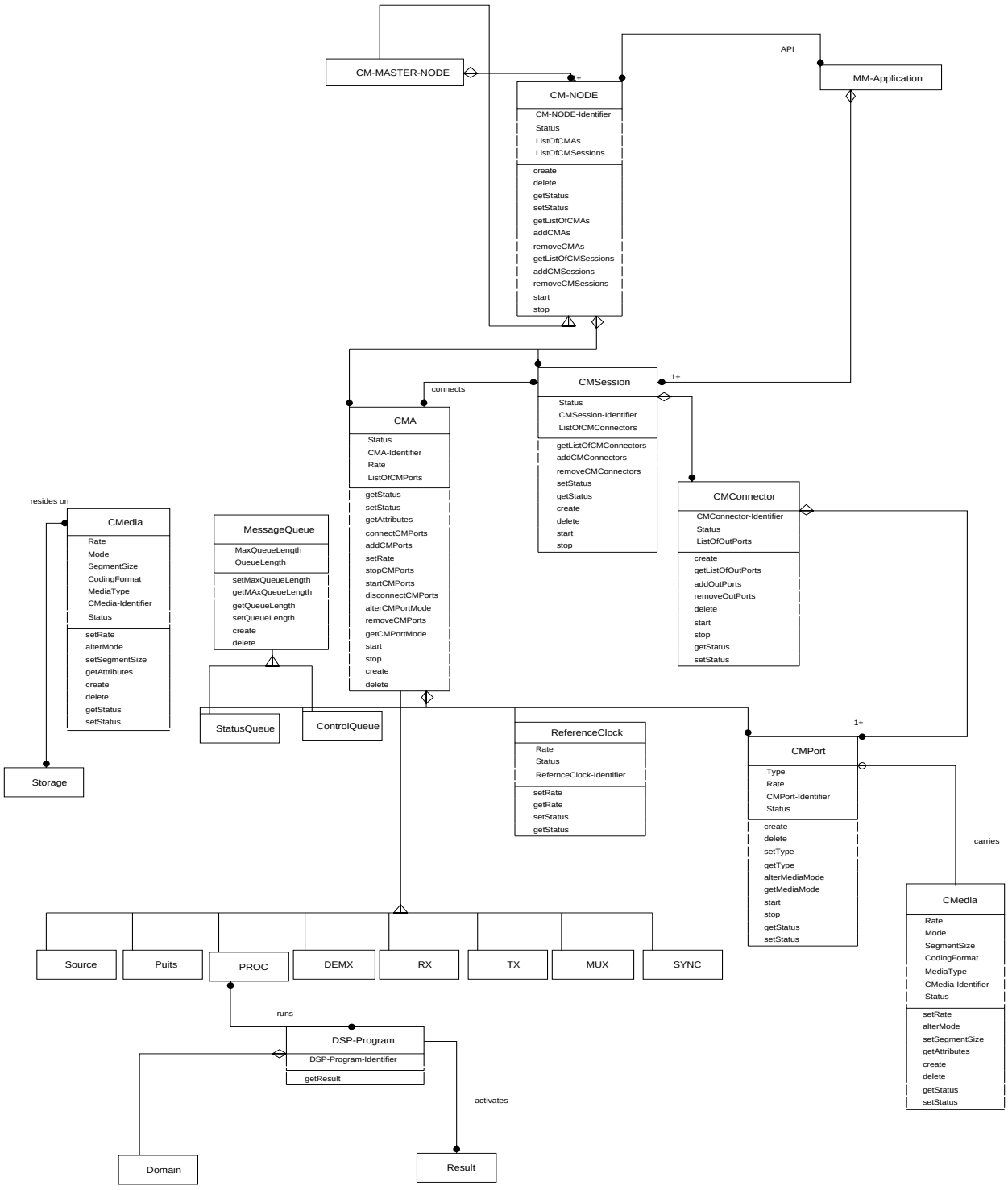
Annexes

8.1 Le modèle objets OMT du VMD réparti

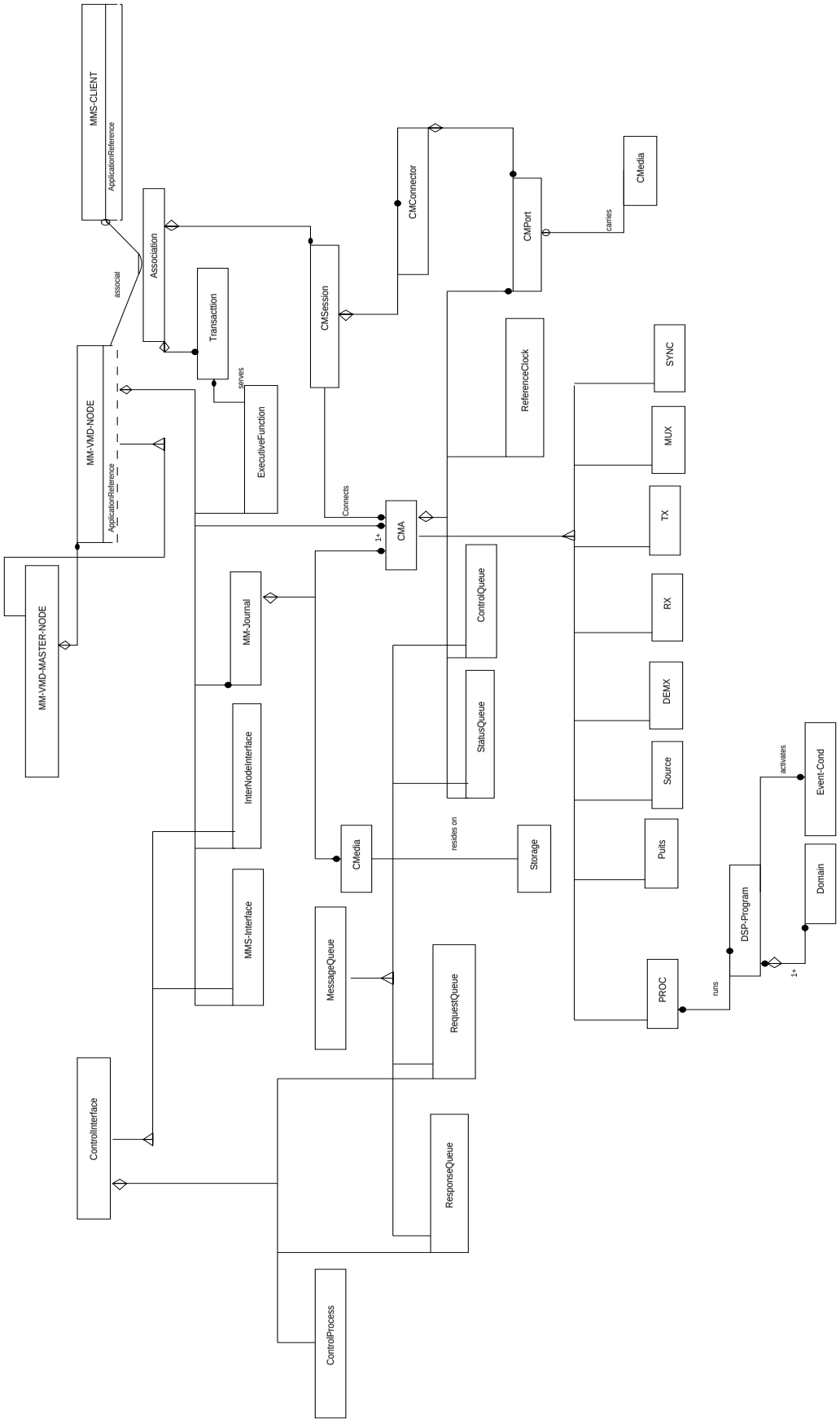


[1]Chaque noeud VMD (*VMD-NODE* possède une interface de communication MMS supportant la communication avec les clients MMS, et une interface de commun ication inter-noeud assurant la communication entre noeuds pour la gestion ou le contrôle d'objets répartis. Un noeud maître (*VMD-MASTER-NODE*) assure la gestion des noeuds qui lui sont subordonnés.

8.2 Le modèle objets OMT de communication de média continus



8.3 Le modèle objets OMT du MM-VMD



8.4 Plate-forme de distribution d'images

8.4.1 Description de la plate-forme

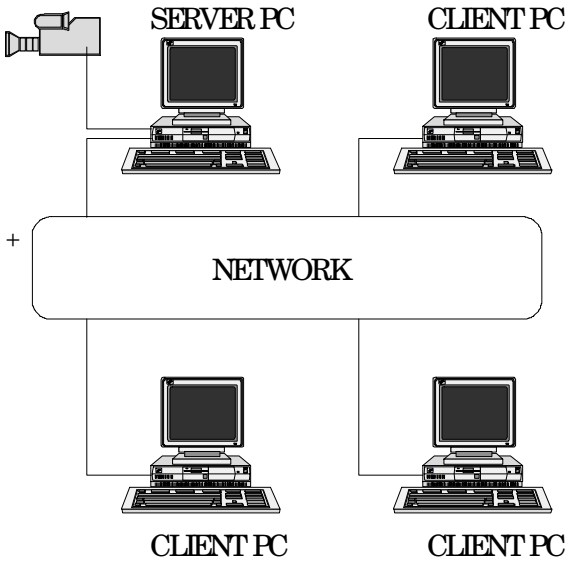


Figure 8.1 : Plate-forme de distribution d'images

Nous avons réalisé une plate-forme de distribution d'images entre ordinateurs personnels connectés par un réseau. La source d'images est une caméra vidéo connectée à l'un des PCs et l'infrastructure de communication est basée sur TCP/IP et un réseau Ethernet. La diffusion offerte par la communication permet d'émettre des lignes d'images numérisées vers des PCs récepteurs.

L'architecture du système est donnée sur la figure 8.1. un client peut se connecter au serveur et initier la transmission des images si ce dernier ne le fait pas encore. Les images reçues sont alors affichées sur l'écran du PC récepteur. Un client peut également demander la suspension d'émission d'images s'il est le dernier client en cours de service. D'autres paramètres tels que le taux de transfert des images peuvent être contrôlés par les clients.

Une caméra vidéo est connectée au PC émetteur par l'intermédiaire d'une carte d'acquisition grand public. Des fonctions de contrôle de la carte d'acquisition sont disponibles dans un kit de développement.

La communication est supportée dans la plate-forme par la pile TCP/IP WinSock de l'environnement MS Windows.

Le transfert des données de contrôle est fait au moyen le protocole TCP, et le transfert de données d'images au moyen de UDP.

8.4.2 Mesure de performances

Les images transmises sont de résolutions 160x120. Une image comporte alors 24 (Bit/Pixel) 160 120 Pixels = 460800 bit.

L'application comporte quatre procédures principales :

1. copie des données du tampon de la carte d'acquisition (CD),
2. conversion de format YUV-RGB (CF),
3. transfert émetteur-récepteur (TF),
4. et affichage sur écran (AE).

Les mesures faites pour une suite de 100 images consécutives donnent :

```
CD = 6 s --> 16,7 images/s --> 7,7 Mbit/s
CF = 11 s --> 9,1 images/s --> 4,2 Mbit/s
TF = 194 s --> 0,52 images/s --> 238 Kbit/s
AE = 3 s --> 33,3 images/s --> 15 Mbit/s
```

Le point de saturation de cette chaîne acquisition-transfert-affichage d'images est bien la transmission. Il est impossible en effet de transférer plus de 0.47 image/s, ce qui correspond à une image toutes les 2 secondes environ. L'implémentation de WinSock utilisée en est la principale raison.