COCACOM : Environnement de Développement d'Applications Coopérantes et Multimédia

J-M. Inglebert, L. Courtrai, S. Sadou
Laboratoire LAMSI
Université de Haute Bretagne

Abstract

This papers présent the COCACOM library : a set of class to distribute of multimédia applications design.The COCACOM class provide a gênerai platform of development and **exploitation** ensuring the distribution and communication between the set of applications **in distributed** architecture. This library contains five class to cover the following topics: parallelism,multimedia,graphic interface,reflex and coopératif work. Two COCACOM applications are presented in this paper: a visionar of burst document and a vote coopératif System.

Introduction

La croissance du nombre des machines multimédia, du fait de la diminution de leur prix, combinée à l'utilisation de plus en plus importante de l'informatique répartie, via les réseaux internationaux, a créé un nouveau domaine de développement : celui des *applications coopérantes multimédia.* La conception et surtout la réalisation de ce type d'application s'effectue soit en développant un support logiciel "maison" soit par l'interfaçage de composants logiciels préexistants tels que :

- Un outil de création, de répartition et de communication de tâches sur un réseau de machines du style *PVM* [GBD93], taches au sens des systèmes d'exploitation. Le logiciel doit être construit aux dessus des protocoles de communication standard tel que *XDR.*
- Un outil de communication entre entités programmables évoluées comme la notion de groupe définie dans *ISIS* [Bir92]. Ce' type de communication constitue un support obligatoire pour la réalisation d'applications coopérantes.
- Un outil de traitement des entrées-sorties multimédia Les données traitées : texte, son et image statique ou dynamique (ex. vidéo) sont délicates à stocker, transporter et synchroniser.
- Un outil de construction d'interface graphique utilisateur. Dans ce domaine les solutions existantes sont nombreuses, on peut citer : les bibliothèques de Widgets comme OSF-Motif ou Athena pour l'environnement unix et le SDK de Microsoft pour l'environnement Dos.

L'interconnexion de ces logiciels spécialisés produit un support peu efficace et souvent difficile à programmer. En effet, l'ensemble devient trop proche des architectures logicielles supports pour

proposer une réelle abstraction de programmation. Les solutions nécessitent un réexamen global du problème en spécifiant les besoins pour ce type d'application. Les qualités du logiciel recherché doivent être celles prônées par le génie logiciel [Mey88] telle que modularité, réutilisabilité, portabilité et abstraction de programmation. L'approche <<Objet Actif» va dans ce sens, en encapsulant dans une même entité des données, du code ainsi qu'une activité. C'est ce que présente cet article en décrivant la .bibliothèque **COCACOM** :

- Classe d'Objets pour la Construction d'Applications Coopérantes Multimédia » sur architecture distribuée. L'outil se veut générique afin de permettre des instanciations spécifiques pour différents types d'application.

La première section présente les différentes classes de l'environnement COCACOM. Nous décrivons les différents concepts et les classes qui l-eur correspondent. Ensuite, nous présentons une implémentation de la bibliothèque de classes sur une architecture distribuée. Dans la seconde partie nous décrivons un exemple d'utilisation un outil de visualisation de documents éclatés. Dans la troisième partie, nous montrons comment, à partir des classes COCACOM, la conception d'application peut engendrer un second jeu de classes dédié (à une classe d'application). Nous décrivons un tel jeu de classes pour un système de vote coopératif.

1 La bibliothèque COCACOM

Nous présentons dans cette section l'outil COCACOM (Classes d'Objets pour la Conception d'Applications Coopérantes Multimédia). Il fournit un ensemble de classes d'objets spécialisés. A l'exécution, un programme COCACOM est matérialisé par une collection d'objets actifs et passifs communiquants entre eux.

1.1 Les **classes COCACOM d'objets actifs et passifs**

La bibliothèque comprend cinq classes de base qui représentent chacune un thème différent parallélisme, multimédia, interface, notion d'événement, travail coopératif. Nous présentons maintenant les différentes classes ; l'interface de programmation complète des classes est décrite dans [CIS94].

Les **tâches**

Nous introduisons les objets *task* comme le grain de décomposition d'une application en entités autonomes qui seront à l'exécution réparties sur les noeuds de réseau. Une composante de l'application est définie par la spécification d'une tâche. Ces objets actifs [CRGM92] contrôlent les objets passifs de la composante de l'application. La coopération des tâches est fondée sur la communication asynchrone par envois de messages entre elles. L'interface de programmation est identique pour les communications locales ou distantes.

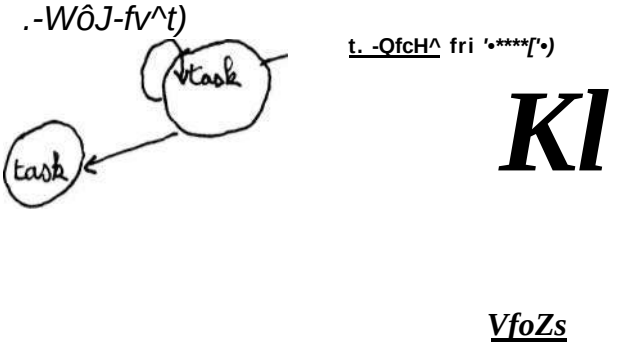

Fig. 1 - Interface de programmation d'un objet *tâche*

Les tâches sont créées dynamiquement par la primitive :

t=new Task("type", site, args...)

Le programmeur doit préciser le nom d'un composant logiciel (représentant la tâche) et spécifie éventuellement le site sur lequel il désire sa localisation (si l'argument n'est pas spécifié, le système

prend en charge cette localisation).
Une tâche peut par envoi de message communiquer le nom d'une autre tâche à une troisième, celle ci pourra alors coopérer avec la nouvelle tâche. On verra par la suite comment les groupes établissent un autre moyen de coopérer.
Les primitives *send* et *waitFrom* permettent d'envoyer et de lire des messages entre les tâches. La communication est transparente quelle que soit la localisation de la tâche destinataire *t.*
Chaque tâche possède un environnement local géré dynamiquement par l'activité de la tâche. Il contient l'ensemble des autres objets de l'application.

Les objets *group*

Les objets *group* sont des entités regroupant un ensemble d'objets. Ils fournissent un nouveau système persistent de nommage dans l'application. Ils survivent à la fin des composantes de l'application et permettant ainsi une communication en mode <<non connecté» des tâches. Un objet adhère à un ou plusieurs groupes et devient accessible des autres objets ayant une référence à ce groupe. Les groupes sont donc le support pour la coopération des tâches.

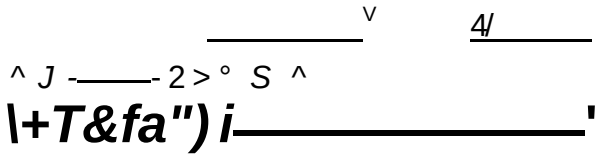

Fig. 2 - **Interface de programmation d'un *groupe***

Un groupe est nommé par un identificateur qui doit être connu par l'ensemble de l'application.
Voici les fonctions principales de la classe ***group*** :

La primitive *g=new Group("identificateur")* c'ée un groupe logique de nom "identificateur" si celui-ci n'existe pas. La primitive *g.join(self)* demande l'adhésion de la tâche courante au groupe de nom logique *g.*
- La communication des tâches des groupes utilise la communication entre les tâches.
- La primitive *g.send(char *)* envoie un message *m* vers les tâches du groupe de g La fonction *g.apply()* applique une méthode sur l'ensemble des tâches du groupe.
- La classe propose un ensemble de méthodes de synchronisation des tâches appartenant à un même groupe.

Les objets *média* et interface graphique

Les objets *média* contiennent les données de l'application. La classe est le support' de plusieurs sous-classes spécialisées ; les objets *texte,* les objets *son* et les objets *images.* Ces classes contiennent les méthodes permettant de les manipuler ces objets média. Des objets *organiseurs* agencent logiquement les différents objets *média* entre eux.
Nous introduisons les objets *média* comme entités de programmation des applications multimédia. Un objet *média* encapsule l'information visuelle, textuelle, sonore ou l'organisation de l'application.
La classe *média* est une classe abstraite ; cinq sous-classes affinent celle-ci et permettent la création d'instances.

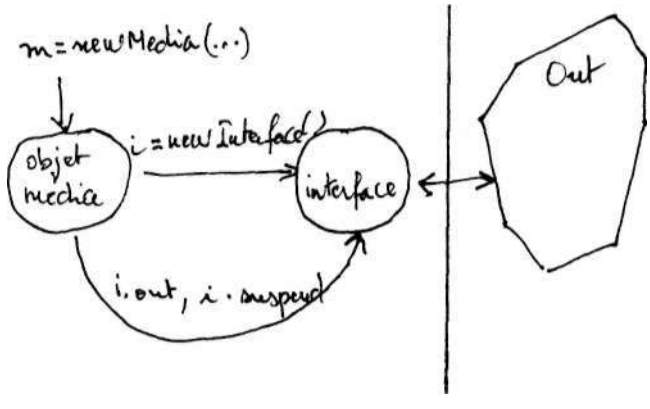

Fig. 3 - **Programmation d'un objet *média* et de son *interface***

La primitive *t=new Media()* permet de créer dynamiquement une instance de type Type (Type étant parmi (Texte, Son, Image, Organiseur)).
Les objets *interface* se chargent de la liaison entre l'application et le monde extérieur (l'utilisateur ou l'architecture). Cette classe doit être implémentée pour chaque architecture logicielle cible. L'objet *interface* connecte un objet média avec un objet de sortie dépendant de l'architecture logicielle et/ou matérielle visée.

Cette dissociation de l'objet média de son interface vers l'extérieur permet de rendre portable l'application sur la plupart des architectures logicielles en <<réimplémentant>> la classe *interface.* Cette indépendance vis à vis de l'extérieur permet d'adapter les entrée-sorties en fonction des composants multimédia de la machine.

Les objets *reflex*
Les objets *reflex* rendent réactifs les objets passifs. Ils associent, à un objet passif, <<un bout de code>> qui sera exécuté, soit directement sur la demande de la tâche, soit indirectement à l'apparition d'un événement venant de l'interface de l'objet.
La primitive *new Reflex(char *actions)* crée un objet associé à une suite d'actions. Chaque action est décrite sous la forme d'un macro-langage et permet trois types d'actions :
- externes : Ce sont les actions externes créant une nouvelle tâche dans l'application. Ces tâches filles sont connectées avec leur tâche mère permettant ainsi une coopération ultérieure.
- internes : Ce sont les actions qui modifient l'environnement local de la tâche. Elle sont spécifiées et implémentées par le programmeur de l'application.

Les objets *reflex* sont souvent associés aux objets *média.* L'objet média n'est plus simplement contrôlé par la tâche. : il devient réactif au monde extérieur. La méthode de la classe *Interface associate(r,e)* connecte un objet *reflex rà* un événement extérieur e qui déclenchera l'action du reflex. Evénement est détecté par l'interface de l'objet, connectée à l'extérieur. L'environnement prédéfinit un ensemble d'événements principalement liés aux entrées-sorties.

1.2 Réalisation
Une implémentation des classes COCACOM, sous la forme d'une librairie de classes C++ , a été réalisée sur le système d'exploitation Unix. Elle à été développée sur un réseau local de stations de travail Sparc SUN et Rs6000 IBM en intégrant des stations distantes via le réseau Internet.
La gestion des tâches (création, répartition) et le système de communication utilisent la version 3 de PVM (Parallel Virtual Machine) ainsi que les <<sockets>> pour la gestion de la coopération «multi-utilisateur». A l'exécution chaque tâche est un processus UNIX. Le système de nommage des tâches est construit au dessus de celui de PVM. La déclaration et la définition d'une tâche s'expriment dans un fichier séparé (composant logiciel) et le nom du composant fournit le nom symbolique de la tâche.
Les groupes sont gérés par un serveur multiprogrammé. Les communications sont construites au dessus des *sockets.*
Les objets *média* sont développés en C++. La classe *Interface* est principalement développée en Motif et X (pour l'intégration des images). La classe *Interface* cache la gestion des événements dans le code de ses instances.

2 Exemple d'utilisation : Htext

Htext est un outil de visualisation de document hypertexte construit à partir des classes COCACOM. Les documents visualisés par **Htext** sont spécifiés dans un fichier suivant la syntaxe HTML des serveurs WEB [BLCGP92]. Cette syntaxe est au standard SGML.
Htext charge un document, reconstruit une version graphique de ce dernier, puis l'affiche. De plus il visualise les liens hypertexte qui permettent à l'utilisateur d'ouvrir d'autres documents.

2.1 Les objets COCACOM utilisés pour la construction de Htext

Dans **Htext,** comme dans tout autre visionneur de documents, on identifie des éléments visibles ou audibles (fenêtres, textes, sons, images, boutons et menus) et des éléments non visibles.
La figure 6 montre l'arborescence des objets média constituants **Htext.**

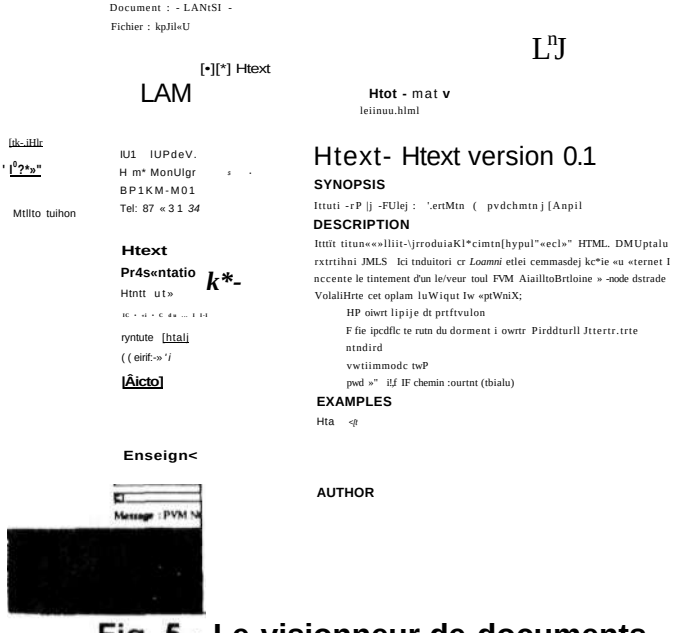

Fig. 5 - Le visionneur de documents *Htex.*

Les éléments identifiés dans **Htext** sont regroupés en deux groupes :
- Les éléments propres au visionneur **Htext;**
- Les éléments décrivants le document visualisé.

La réalisation, en terme d'objets issus des classes COCACOM, pour le premier groupe est la suivante :

- La fenêtre visualisant un document est réalisée par un objet de type *Task.*
- Les boutons ou menus du visionneur sont construits à partir d'objets *Media, Interface* et *reflex.* Par exemple, le bouton permettant de quitter l'outil est associé à *reflex* d'envoi de message de terminaison aux autres tâches générées par l'outil.
- Le convertisseur est un élément non visible (traitements) qui construit des objets *Media* à partir d'un fichier HTML représentant un document. Ce convertisseur est réalisé à partir d'un objet de type *Task.*

La réalisation, en terme d'objets issus des classes COCACOM, pour le second groupe est la suivante :
- Les entités (données) du document sont traduites directement en terme d'objets *Média* (par le convertisseur).
- La structuration du document est prise en charge par des objets *Managment.* Chaque objet *Managment* gère une liste d'objets *Media;* ceux-ci pouvant être des objets *Managment.*
- Les liens hypertexte sont réalisés par les objets *Reflex* attachés à des objets *Media* de type *command.*

2.2 Lien entre Htext et la syntaxe du document

La construction du visionneur est indépendante de la syntaxe du document. Il suffit en effet de modifier l'objet convertisseur pour accepter d'autres syntaxes. Un premier prototype de **Htext** visualisait des documents hypertexte utilisant une syntaxe définie localement.

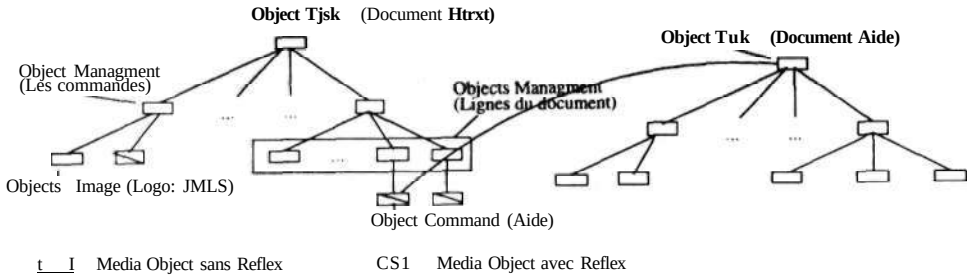

Fig. 6 - Objets constituant *Htext*

Conclusion

Nous avons présenté une bibliothèque de classes pour la construction d'applications coopérantes ainsi qu'une implémentation sur une architecture distribuée. Nous avons ensuite montré un niveau d'abstraction qui par instanciation d'un jeu classes permet de spécifier le support logiciel pour une classe d'application.
Nos travaux portent actuellement sur la couche support et la couche abstraction :
- Mise en place d'abstractions de synchronisation des objets actifs et passifs de l'environnement [GC92],
- Extension des services et des protocoles de coopération entièrement compatible avec les outils actuels
- Etablissement d'un langage de conception d'applications multimédia coopératives capable de s'adapter (s'instancier) à des environnements physiques différents
- Mise en place d'un générateur d'applications incluant une interface de conception et la production du code des applications

Bibliographie

[Ame87] P. America. POOL-T : A Parallel Object-Oriented Language, in Object-Oriented YONEZAWA, The MIT Press, 1987.

[Bir92] Kenneth-P. Birman. The process group approach to reliable distributed. Technical Report TR-91 -1216, Cornell University, Department of Computer Science, March 1992.

[BLCGP92] Tim Berners-Lee, Robert Cailliau, Jean-Francois Groff, and Bemd Pollerman. World-Wide Web: The Information Univers Technical report, Cern, Internet Draft, 1992.

[CIS94] Luc Courtrai, Jean-Michel Inglebert, and Salah Sadou. Les classes COCACOM : Manuel de référence. Technical report, LAMSI, Vannes, Juillet 1994.

[CRGM92] Luc Courtrai, Jean-Francois Roos, Jean-Marc Geib, and Jean-Francois M e haut. Communicating A ctive C omponents: an E nvironment for Concurrent Applications on Parallel Machines. EUROMICRO'92, Proceedings of EUROMICRO Hardware and Software Design Automation, published in Microprocessing and Microprogramming, Paris (F) , 35(1-5):47—54, September 1992.

[GBD 93] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. PVM 3 users's guide and référence manuel . Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, Enginnering physic and Mathematics Division

Mathematical Science Section. May 1993

[GC92] Jean Marc Geib and Luc Courtrai. A bstractions for S ynchronization to I nherit S ynchronization C onstraints In Proceedings of ECOOP'92 Workshop on Object Based Concurrency and Reuse Utrecht Netherlands, June 1992.

[Mey88] B Meyer Object-Oriented Software Construction Prentice-Hall, 1988.

[Str86] B. Stroustrup. The C++ Programmmg Language Addison-Wesley Publishing Company, 1986.

[YBS86] A Yonezawa, J. P. Briot, and E. Shibayama. Object-Oriented Concurrent Programming in ABCL/1 OOPSALA[1] 86 Proc of the ACM Conf. on Object-Oriented Programming Systems, Languages and Applications, Spécial Issue of SIGPLAN Notices,21(11)258-268,November 1986.