# GroupDesign: un éditeur partagé dans un environnement hétérogène

Alain Karsenty et Christophe Tronche

Laboratoire de Recherche en Informatique - CNRS URA 410
Bâtiment 490 - Université de Paris-Sud
91 405 Orsay Cedex
tél : 69 41 65 91 - fax : 69 41 65 86
e-mail : ak@lri.fr, tronche@lri.fr

**Résumé** : GroupDesign est une application de dessin structuré qui a la particularité d'être un collecticiel hétérogène. En effet, GroupDesign permet à plusieurs utilisateurs d'éditer simultanément le même document dans un environnement hétérogène (Macintosh/stations Unix). Cet article présente les résultats de cette expérience. Concernant l'interface de l'application, nous décrivons les fonctionnalités de GroupDesign propres au collecticiel : Echo graphique et audio, Localisation, Identification, Téléconférence, Age et Historique. Concernant l'implantation du système, nous examinons les difficultés intrinsèques à l'aspect hétérogène et nous montrons comment nous avons transformé une application mono-utilisateur en collecticiel hétérogène.

**Mots clés** : Groupware, Collecticiel, Éditeurs Multi-Utilisateurs, Systèmes Coopératifs.

**Abstract** : GroupDesign is a real-time multi-user drawing tool that runs in a heterogeneous environment. Indeed, GroupDesign allows a large number of users to edit simultaneously the same document in a heterogeneous environment (Macintosh/Unix workstations). This article draws the conclusions from our experience. First, we present a number of functionalities that we feel necessary in any real-time groupware system : Graphic & Audio Echo, Localization, Identification, Age, and History. Then, we describe the implementation of the system, focusing on the specific problems raised by the heterogeneous aspect.

**Keywords** : Groupware, Group editor, Collaborative system, Computer-Supported Cooperative Work.

# Introduction

La vision traditionnelle d'un logiciel interactif est d'associer une application à un utilisateur travaillant sur une machine. Les bases de donnés ou les systèmes multi-utilisateurs comme Unix, conservent cette vision qui consiste à donner à l'utilisateur l'impression qu'il est seul sur sa machine, bien que d'autres utilisateurs puissent accéder aux mêmes données. Le collecticiel, domaine des applications de groupe, remet en question cette vision puisqu'une application collecticielle prend en compte plusieurs utilisateurs et vise à rendre chaque utilisateur conscient des actions des autres utilisateurs plutôt que de les isoler les un des autres. Un premier type d'applications collecticielles n'ont pas d'équivalent mono-utilisateur : ce sont les messageries électroniques, les systèmes de coordination, les conférences par ordinateur,etc. Par ailleurs, il existe des collecticiels qui sont des extensions d'applications existantes. C'est le cas des éditeurs partagés, qui permettent à plusieurs utilisateurs d'éditer un même document depuis leurs stations de travail respectives. Par exemple, la rédaction d'un article par plusieurs personnes se fait souvent par échange de fichiers alors qu'il est plus simple de travailler simultanément sur le même document.

C'est ce dernier domaine des applications collecticielles en tant qu'extension d'application mono-utilisateurs auquel nous nous sommes intéressé. Cette étude s'est réalisée à travers le développement d'un prototype d'éditeur graphique, GroupDesign, permettant d'éditer simultanément un dessin sur stations Macintosh et Unix. Ceci nous a permit d'évaluer la difficulté de transition d'un logiciel mono-utilisateur au collecticiel et à révélé deux problèmes : l'insuffisance de l'interface des systèmes mono-utilisateurs et l'absence d'architecture logicielle adaptée à nos besoins. Au niveau de l'interface, nous avons résolu le problème par un ensemble de fonctionnalités qui viennent s'adjoindre aux fonctions mono-utilisateurs et qui permettent de soutenir l'activité du groupe afin que l'utilisateur ne se sente pas isolé. Du point de vue de l'architecture, nous avons developpé une architecture purement répliquée hétérogène, qui donne à notre système la souplesse nécessaire a un environnement hétérogène et la réactivité suffisante pour une interface à manipulation directe.

Nous décrivons en premier lieu GroupDesign et les fonctionnalités développées propres au collecticiel. Nous présentons ensuite l'architecture répliquée hétérogène et détaillons l'implantation du système. Puis, nous terminons cet article par une discussion et une section sur les travaux dans le même domaine.

# L'application GroupDesign

GroupDesign est une application de dessin structuré multi-utilisateurs similaire à MacDraw, implantée à la fois sur un réseau de Macintosh connectés par LocalTalk et sur stations de travail Unix. GroupDesign a été conçu dans le but d'être utilisé comme outil de dessin pour la réalisation de projets impliquant de nombreux utilisateurs et répartis sur une longue période plutôt que comme support durant des réunions de travail. Le mode d'interaction des utilisateurs peut donc varier au cours de la session d'un mode synchrone, où les utilisateurs modifient la même page du document, à un mode asynchrone, dans lequel chaque utilisateur travaille séquentiellement ou bien parallèlement sur des parties du document distinctes. Hormis les fonctions de dessin classiques, nous avons donc fourni un ensemble de fonctions collecticielles qui couvre les différents modes de travail des utilisateurs. Les fonctions synchrones permettent aux utilisateurs de comprendre l'action simultanée du groupe sur le diagramme : Echo graphique et audio, mode Téléconférence, Localisation. Les autres fonctions sont utiles pour comprendre l'évolution du document, donc dans un mode de travail asynchrone : Historique, Age du document, Identification. La plupart des fonctions utilisent la couleur comme moyen d'identification des utilisateurs. Un menu liste les noms des utilisateurs participant à la session avec une couleur unique associée à chacun d'eux. Les paragraphes suivants décrivent plus précisément les fonctions collecticielles que nous avons développé.

### *Écho*

Nous utilisons le mot *écho* pour décrire la représentation des actions d'un utilisateur sur les interfaces des autres utilisateurs du groupe. Si par exemple un utilisateur crée un rectangle, cette opération est propagée à tous les utilisateurs qui reçoivent alors un écho de l'opération de création.

L'écho doit être à la fois compréhensible et non distrayant. C'est un compromis à atteindre qui varie suivant la nature des applications. Dans un système de téléconférence, où l'interaction entre utilisateurs est particulièrement forte, l'écho sera plus important que pour un éditeur de texte multi-utilisateurs qui doit permettre à chaque utilisateur de travailler sur un paragraphe sans être constamment distrait par les modifications effectuées par les autres utilisateurs.

GroupDesign permet de construire des diagrammes complexes impliquant éventuellement un nombre d'utilisateurs important. Il est donc probable que l'interaction entre utilisateurs ne doit pas être trop forte afin de leur laisser une certaine indépendance. Ainsi le type d'écho que nous avons développé (graphique et audio) vise à atteindre ce compromis où un utilisateur est à même de suivre les actions du groupe sans être déconcentré par les fonctions d'écho.

***Écho graphique***

En mode normal, l'application utilise une interface WYSIWIS ("What You See Is What I See") [17] non stricte. Le terme WYSIWIS signifie que chaque utilisateur a la même vue du document. Dans notre cas, WISIWIS non-strict signifie que le diagramme est le même pour tous les utilisateurs mais que chaque utilisateur garde un contrôle indépendant sur ses fenêtres.

Lorsque plusieurs utilisateurs ont une vue commune du diagramme, les données modifiées par un utilisateur ont un écho graphique sur l'écran des autres utilisateurs, sous forme d'une animation. Si par exemple un utilisateur redimensionne un rectangle, l'écho du redimensionnement sur les autres écrans consiste en une animation qui modifie progressivement la taille du rectangle. Si l'on se contentait d'afficher le rectangle avec sa nouvelle taille, l'effet de sautillement visuel serait désagréable pour l'utilisateur et rendrait difficile le suivi des opérations du groupe. Au contraire, l'animation provoque un effet visuel naturel sans être distrayant. Son inconvénient est d'être coûteuse en temps de calcul et de dessin, un problème déjà étudié et résolu par Card et al. [8].

La plupart des opérations graphiques donnent lieu à deux échos correspondant au début et à la fin de l'opération. Le début est signalé par l'affichage d'une icône représentant l'opération en cours sur l'objet modifié, et dont la couleur est celle de l'utilisateur qui effectue l'opération. La fin de l'opération est représentée par une animation montrant le changement d'état. Considérons par exemple un utilisateur déplaçant un rectangle. Cette opération se fait en trois phases, avec des effets différents pour l'utilisateur effectuant l'opération et pour le groupe :

Phase 1 - *utilisateur*: clique sur l'objet.
*groupe*: dessin d'une icône représentant l'opération de déplacement de la couleur de l'utilisateur, sur l'objet déplacé (figure 1, phase 1).
Phase 2 - *utilisateur*: déplace la souris, le feed-back l'aide à placer l'objet.
*groupe*: rien de visible ; l'objet est verrouillé et ne peut être déplacé.
Phase 3 - *utilisateur*: relâche le bouton de la souris, ce qui a pour effet de dessiner l'objet final.
*groupe*: l'icône est effacée et l'objet, au moyen d'une animation graphique, se déplace jusqu'à la position finale (figure 1, phase 2).

Figure 1 - Echo graphique

Le premier écho est particulièrement important puisqu'il limite les risques de conflits. En effet, cet écho avertit les utilisateurs de l'identité de l'utilisateur et du type de l'opération en cours, et verrouille l'objet pour les opérations de même type. Les conflits ne peuvent donc se produire que lorsque deux utilisateurs agissent sur le même objet exactement en même temps.

***Écho audio***

L'echo audio vient combler une lacune propre à l'écho graphique : lorsqu'une opération n'est pas visible dans le cadre de la fenêtre, aucun écho n'est fourni à l'utilisateur. L'écho audio consiste à représenter chaque opération graphique par un son approprié, le volume indiquant la distance par rapport à l'endroit ou l'utilisateur se situe dans le diagramme. Le son permet d'avoir rapidement une idée globale de l'activité du groupe, comme le montre le système Arkola [12]. Il évite par exemple l'effet distrayant de voir soudain un objet glisser dans le cadre d'une fenêtre, puisque l'écho audio anticipe l'écho graphique.

***Identification et localisation***

L'identification permet de retrouver les noms des utilisateurs qui ont créé ou modifié le diagramme. Ce mode fonctionne selon deux options : chaque objet a la couleur de l'utilisateur qui l'a créé, ou bien du dernier utilisateur qui l'a modifié. La couleur est un moyen particulièrement clair d'identifier les utilisateurs. Le diagramme se lit comme une carte ou se dessine suivant les couleurs des zones où chaque participant a travaillé.

Dans le mode WYSIWIS non strict, l'utilisateur n'a pas de moyen pour retrouver la zone du diagramme où un autre utilisateur travaille. Le mode localisation permet de visualiser le contenu de la fenêtre frontale des participants à la session. Chaque fenêtre est représentée par un rectangle de la couleur de l'utilisateur. De plus, sélectionner un utilisateur dans le menu, centre la fenêtre sur la zone visionnée par celui-ci. Les utilisateurs peuvent ainsi se "téléporter" d'une zone utilisateur à une autre. Ce mode permet aux utilisateurs de se synchroniser tout en laissant une certaine

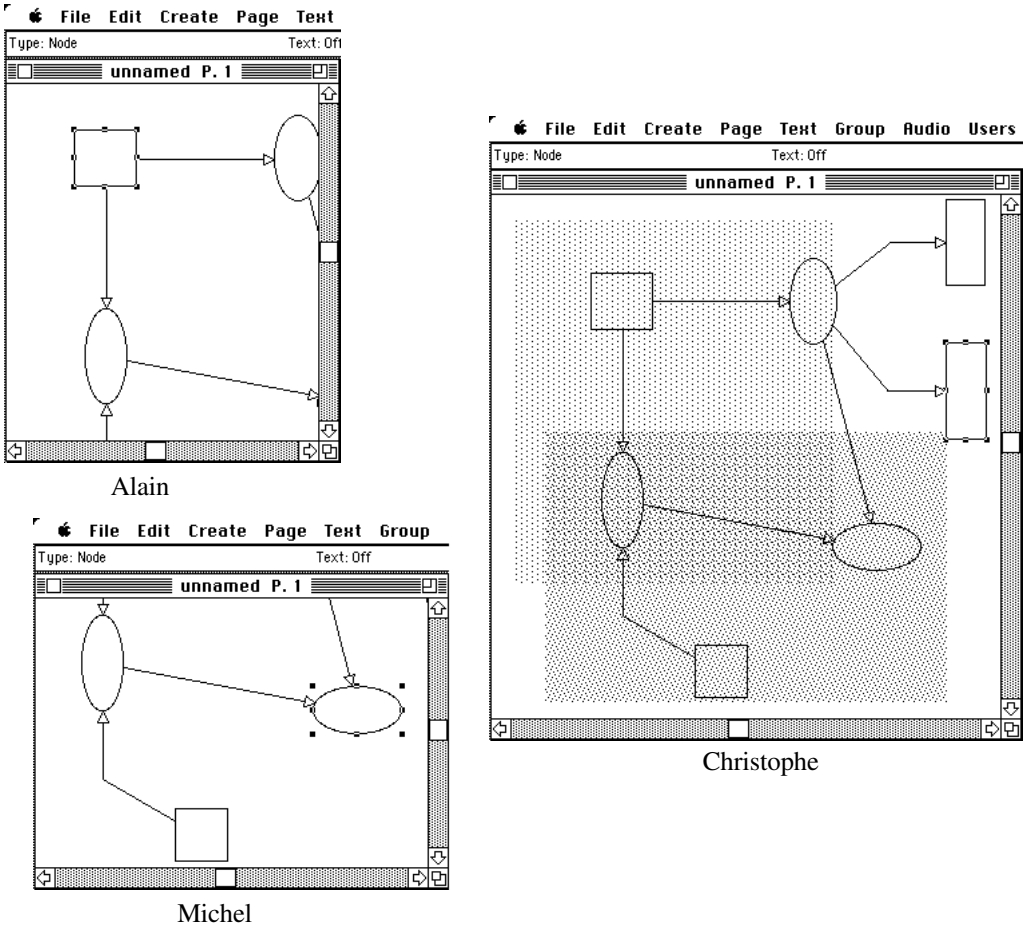


Figure 2 - Mode localisation

indépendance, ce qui serait difficile à réaliser avec une interface strictement WYSIWIS. La figure 2 est un exemple de session collecticielle comprenant 3 utilisateurs. L'utilisateur Christophe est en mode localisation, ce qui lui permet de visualiser la vue de Alain et Michel.

***Téléconférence***

Dans le contexte d'une téléconférence ou le synchronisme des vues est important, le mode localisation n'est pas suffisant. Dans le mode téléconférence, toutes les opérations de l'utilisateur sur les fenêtres sont transmises aux autres participants de la session. Dans ce cas, l'interface est strictement WYSIWIS. L'addition d'un moyen de communication audio ou vidéo est dans ce cas souhaitable afin d'accompagner d'un commentaire les opérations sur le diagramme.

***Age du document et historique***

Le mode "Age du document" permet de retrouver les parties du diagramme le plus récemment modifiées. Les objets sont affichés d'une couleur variant du rouge au bleu selon leur date de dernière modification. Dans ce mode, un utilisateur peut facilement repérer les parties où les utilisateurs sont le plus actif.

L'historique permet de rejouer les dernières opérations effectuées par le groupe. Une boite de dialogue affiche pour chaque opération rejouée le nom de l'utilisateur et le type de l'opération (déplacement, création,...). L'historique est proche de la fonction d'age puisqu'en visionnant les dernières opérations on retrouve l'age du document mais il donne de plus la nature exacte des dernières opérations. De plus l'age du document est un mode permanent, alors que l'historique est une commande : on ne peut modifier le diagramme pendant que l'on parcourt l'historique.

***Édition privée***

Dans le mode édition privée, un utilisateur peut effectuer un ensemble d'opérations qui sont rendues visibles sur les autres écrans sur demande de l'utilisateur. Par exemple, un utilisateur novice dans l'utilisation du logiciel et qui ne veut pas que ses hésitations soient visibles par le groupe peut utiliser ce mode pour transmettre en un seul bloc le dessin final.

Cependant, lorsque l'utilisateur est dans le mode édition privée, il s'isole du groupe qui n'est plus conscient de ses modifications. Ainsi les membres du groupe risquent de modifier les objets que l'utilisateur en mode édition privée a déjà modifié, ce qui augmente les risques de conflits. En cas de conflit, la résolution se fait au désavantage de l'utilisateur en mode édition privée afin de gêner le moins d'utilisateurs possible. Ces conflits peuvent cependant être limités : un utilisateur en mode

édition privée est signalé au groupe, qui peut alors éviter de travailler dans la zone de cet utilisateur (en utilisant le mode localisation).

# Architecture

Les architectures traditionnelles des applications collecticielles sont le plus souvent centralisées ou partiellement répliquées. En effet, la gestion de la concurrence entre différents utilisateurs peut être réalisée simplement si un unique processus effectue les opérations, le risque de conflit étant ainsi supprimé. Or ce type d'architecture ne nous semble pas adapté aux applications collecticielles qui nécessitent une réponse rapide de l'interface. C'est pourquoi nous avons conçu une nouvelle architecture que nous qualifions de purement répliquée et hétérogène.

## Purement répliquée

Une architecture purement répliquée signifie que chaque utilisateur a sa propre application connectée à travers le réseau aux autres applications de la même session. Chaque modification du diagramme est propagée à travers le réseau afin de mettre à jour les autres applications. Dans l'environnement Macintosh nous avons exactement un processus par utilisateur, alors que sous Unix un certain nombre de processus intermédiaires, décrits dans les paragraphes suivants, gèrent le transport des données.

Le choix d'une architecture répliquée plutôt que centralisée est justifié par les raisons suivantes :

- réponse de l'interface : une architecture centralisée implique une réponse plus lente de l'interface aux actions de l'utilisateur alors que l'architecture purement répliquée permet une réponse optimale de l'interface puisque les actions des utilisateurs peuvent être traitées immédiatement, sur sa propre machine.
- adaptée à un environnement hétérogène : une architecture centralisée hétérogène impliquerait une réponse encore plus lente de l'interface puisque les messages devraient traverser plusieurs types de réseaux.
- autonomie : chaque application est totalement autonome et le système est ainsi plus robuste.
- "collectivisation" d'applications existantes : une telle architecture permet, comme nous l'avons fait pour GroupDesign, de rendre collecticielles des applications mono-utilisateurs existantes.

Cependant, une architecture répliquée nécessite un contrôle réparti de la concurrence, et une gestion des conflits qui découlent de cette concurrence. Pour gérer les conflits qui résultent de la modification simultanée d'un même objet par plusieurs utilisateurs, nous avons implanté un algorithme basé sur la sémantique de l'application [13].

## Hétérogène

GroupDesign est une application hétérogène, c'est à dire capable de s'exécuter simultanément dans des environnements différents : Unix et Macintosh dans notre cas. Ecrire une application purement répliquée hétérogène est en principe relativement facile : il suffit d'écrire les exemplaires (nous appelons ainsi les "répliques" dans un environnement hétérogène), dans le nouvel environnement et de les faire communiquer avec l'application existante au travers d'un réseau. Des difficultés de réalisation se posent néanmoins.

### *Inadaptation des environnements actuels*

Cette inadaptation prend deux formes : d'une part les systèmes et les réseaux existants sont aussi inadaptés à l'écriture de collecticiel hétérogène qu'à l'écriture de collecticiel homogène (cf par exemple [16]) ; notamment, les systèmes et les réseaux courants ne peuvent réagir en temps borné. D'autre part les boites à outils existantes sont aussi rares que difficiles à exploiter pour écrire du collecticiel (le domaine étant trop récent). Nous sommes donc contraints de développer nos propres outils.

### *Transport*

L'architecture purement répliquée impose de trouver un compromis entre le nombre de liens traversés par un message et le nombre de messages qui vont circuler dans le réseau. En effet, augmenter le nombre de liens allonge le temps de transmission, ce qui diminue le couplage entre les utilisateurs, mais permet une plus grande efficacité dans le transport des messages [18]. Passer au collecticiel hétérogène met l'accent sur les problèmes de performances du réseau : un message passe par plusieurs routeurs intermédiaires et la vitesse dans un chemin du réseau est la vitesse du lien le plus lent.

***Nommage***

Tout réseau homogène possède un mécanisme qui permet de nommer n'importe quelle machine du réseau. Dans un réseau hétérogène, on doit pouvoir nommer des machines sur des réseaux pour lesquelles les conventions de nommage ne sont pas les mêmes. La connexion de réseaux pose des problèmes supplémentaires. Si deux machines, chacune sur un sous-réseau, portent le même nom, une collision de noms se produit lorsque les deux sous-réseaux sont connectés.

***Formats***

Les applications s'échangent des informations de formes diverses : texte, image, son, etc. sous des formats eux-même très divers. Certains formats sont très spécifiques à un matériel, un système d'exploitation, voire une application. Travailler dans un environnement hétérogène impose la conversion de formats de représentation de données en d'autres formats.

# Implantation

GroupDesign a été réalisé aussi bien sur Macintosh que sur stations Unix à partir d'une interface de programmation basée sur une application de dessin structuré existante.

Sur Macintosh, nous avons utilisé MetaDesign, une application mono-utilisateur existante. MetaDesign est prévu pour servir de noyau à des applications de dessin structuré au travers d'une interface de programmation, Design/OA [14]. Sous Unix, nous avons utilisé le Widget Graphe, un widget Motif qui permet de gérer des graphes (noeuds reliés par des liens) de façon similaire à Design/OA.

MetaDesign et le Widget Graphe fonctionnent selon le même principe : d'une part, des fonctions de programmation permettent de réaliser par programme tout ce que l'utilisateur peut faire de manière interactive ; d'autre part, des filtres ou "callbacks" permettent d'intercepter les actions de l'utilisateur afin de renseigner les modules d'extension.

Le tableau ci-dessous résume l'état actuel de notre expérience. L'hétérogénéité est maximale, puisque toutes les couches sont différentes (et incompatibles).

| | **Unix** | **Macintosh** |
|---|---|---|
| **Machine** | Station Unix | Apple Macintosh |
| **Réseau** | IP | AppleTalk |
| **Protocole** | UDP | AppleEvents |
| **Système d'exploitation** | Unix | Apple Macintosh |
| **Système graphique** | X11 | Quickdraw |
| **Interface de programmation** | Widget Graph | Design O/A |

Afin de résoudre les problèmes de transport évoqués dans la section précédente, il faut disposer d'au moins une machine qui soit capable de "parler" le langage des deux mondes AppleTalk et TCP/IP. Ceci est obtenu en permettant à un Macintosh d'utiliser IP pour discourir avec le monde UNIX, les liaisons de Macintosh à Macintosh se faisant par AppleTalk. La liaison est obtenue physiquement par un câble Ethernet. La bibliothèque MacTCP permet de programmer le Macintosh en exploitant IP.

En ce qui concerne le nommage, étant donné que nous avons fait le choix d'avoir les Macintoshs comme machine "bilingue", il faut une adresse IP sur le Macintosh où envoyer des messages, c'est à dire une adresse physique, un protocole (UDP) et un numéro de port. Un serveur de noms situé à une adresse connue permet aux applications de se localiser les unes les autres.

La réalisation de GroupDesign s'est faite relativement aisément, notamment grâce au moteur réactif décrit ci-dessous. Ceci démontre que l'architecture purement répliquée permet de transformer une application mono-utilisateur en application multi-utilisateurs sans trop de difficulté, ce qui nous semble un point important.

## Le moteur réactif

Notre architecture étant purement répliquée, une session donnée est constituée d'au moins autant de processus qu'il y a d'utilisateurs en session (un processus par application interactive). Il est utile de voir une telle application comme un ensemble d'entités communicantes. Les différents exemplaires de l'interface (qui tournent chacun sur une machine) sont tous de telles entités : leur fonctionnement consiste essentiellement à s'échanger des messages, à effectuer une requête graphique / audio, ou à traiter un événement venant de l'utilisateur. Des entités supplémentaires sont nécessaires en environnement hétérogène, par exemple des serveurs de noms. Toutes ces entités ne sont connues de l'extérieur que par les messages auxquels elles savent répondre ;

l'abstraction correspondante pour les processus dispersés à travers un réseau est celle d'*acteur*. Un acteur reçoit des messages et y répond en envoyant d'autres messages.

Nous implantons cette abstraction grâce à une bibliothèque de classes C++ : le *moteur réactif*. Le moteur réactif fournit des primitives de haut niveau pour la description des communications et le nommage des acteurs distants. D'autre part, il suffit de lui fournir les noms et les corps des méthodes auquel un acteur doit être capable de répondre. Le moteur réactif prend en charge l'attente des messages, leur décodage, l'appel du corps de méthode correspondant, et le renvoi éventuel des résultats.

Une dernière possibilité du moteur réactif est d'écouter plusieurs sources d'événements, toutes les sources n'utilisant pas nécessairement le même type. Ceci est souvent utile dans le cadre du collecticiel hétérogène ; par exemple un exemplaire d'une interface sous X Window peut être à la fois en attente d'événements locaux venant du serveur X et d'événements distants arrivant d'autres acteurs au travers du réseau.

Un acteur écrit avec le moteur réactif peut posséder un nom, qui est une paire constituée par le nom de l'application et le nom de l'utilisateur. Si un acteur possède un nom, il est enregistré automatiquement à sa création auprès du serveur de noms. Les autres acteurs peuvent alors lui envoyer des messages (en utilisant le nom symbolique). Le moteur réactif traduit automatiquement les noms des acteurs distants en adresses réseau. Un message est découpé en champs possédant chacun un nom, et contenant une chaîne de caractères.

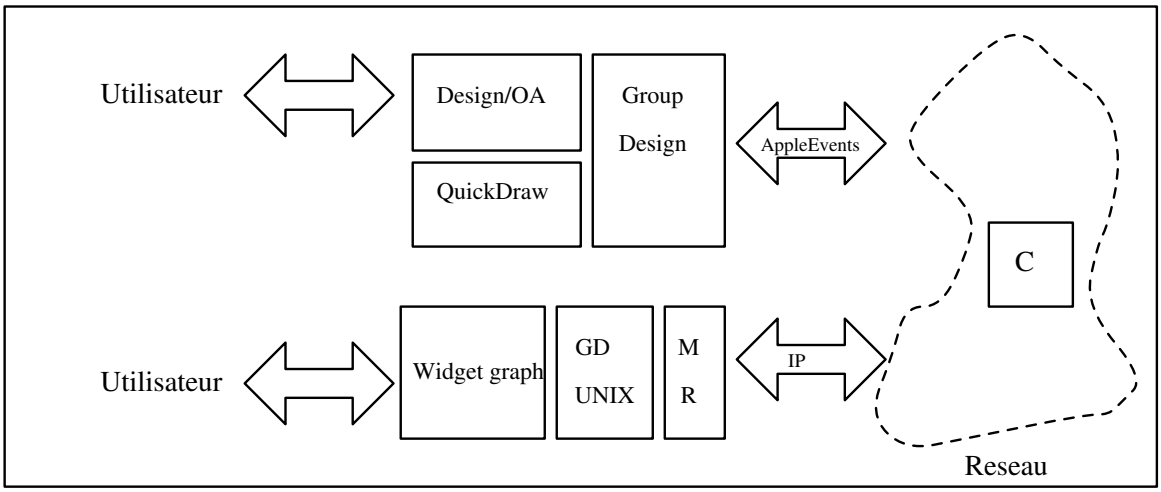


Figure 3 - Session hétérogène

## L'application finale

La figure 3 représente l'application en fonctionnement, avec deux machines en session : un Macintosh et une station Unix. D'un côté, chaque application communique avec le réseau. Pour des raisons historiques, GroupDesign utilise directement les AppleEvents, et sa version Unix (GD UNIX), utilise le moteur réactif (MR).

La communication avec l'utilisateur se fait sur le Macintosh avec Design O/A d'une part et directement avec les primitives graphiques de Quickdraw, la boite à outils graphique du Macintosh. Sous Unix, nous avons utilisé comme interface le widget Graphe [2]. Toutes les machines en session bénéficient alors des possibilités des deux plateformes, par exemple toutes peuvent utiliser le placement du widget Graphe.

La boite notée "C" sur le schéma est un boîtier logiciel effectuant la conversion des messages dans les deux sens. Il est important de ne pas saturer ce type de boîtier avec des messages pour ne pas dégrader les performances, alors que l'architecture répartie est potentiellement grosse consommatrice de bande passante. Le problème de l'efficacité du transport est d'ailleurs un problème important du collecticiel synchrone. Nous l'avons traité en utilisant des réseaux d'acteurs reconfigurables (construits au-dessus du moteur réactif), qui peuvent s'adapter à la topologie de la session et effectuer le transport de façon relativement économe. Cette solution est détaillée dans [18].

## Discussion

La discussion suivante présente les leçons tirées de cette expérience en ce qui concerne le protocole inter-application et le passage d'une application mono-utilisateur au collecticiel hétérogène.

***Protocole inter-applications***

Nous entendons par protocole inter-application la nature des messages qui vont être transmis d'un exemplaire à l'autre. En effet, la correspondance entre une opération que l'utilisateur effectue et le message qui est transmis aux applications des autres utilisateurs n'est pas triviale. Dans les cas de commandes courtes, par exemple les raccourcis clavier, la correspondance est la plupart du temps directe. Cependant les opérations peuvent aussi être longues lorsqu'elles sont effectuées à partir d'un dialogue ou bien par manipulation directe. Prenons, l'exemple d'un utilisateur de GroupDesign qui déplace un rectangle. L'opération de déplacement d'objet se décompose en l'envoi de deux messages qui sont d'abord le verrouillage de l'objet lorsque l'utilisateur clique sur le rectangle, puis le message de déplacement lorsque l'utilisateur relâche la souris. Ainsi, la correspondance opérations/messages doit être bien choisie afin de répondre aux besoins du groupe utilisant le collecticiel. En particulier, nous avons vu précédemment que l'envoi de deux messages pour les commandes longues permettait de verrouiller partiellement les objets et de réduire les conflits.

De plus, dans un environnement hétérogène, si la correspondance entre opérations et messages est bien choisie, il n'est pas nécessaire que les applications aient toutes les mêmes opérations. Il suffit que chaque commande soit décomposée en sous-ensembles de plus bas-niveaux que toutes les applications connaissent. Par exemple, le Widget Graphe implante un algorithme de placement de graphe que GroupDesign sur Macintosh ne connait pas. Le message qui est envoyé aux autres exemplaires est celui de repositionnement pour chaque objet et non pas un message global de repositionement automatique que GroupDesign n'a pas implanté.

***Du mono-utilisateur au collecticiel hétérogène***

GroupDesign est un exemple d'application collecticielle hétérogène qui a été implantée à partir d'une application mono-utilisateur existante sous Unix (WidgetGraph) et sous Macintosh (Design/OA). Ces interfaces de programmation permettent de facilement modéliser une application comme une boite émettrice et réceptrice. L'émission d'événement est aisée grâce aux filtres de Design/OA ou aux "callbacks" de la toolkit Motif. L'aspect réception est assuré par une librairie de fonctions spécifique à l'application. Ainsi, avec l'utilisation du moteur réactif précédemment décrit, la collectivisation d'une application est simplifiée.

Cependant, les interfaces de programmation ne prennent pas en compte les aspects coopératifs des applications. Ainsi, les fonctions de GroupDesign propres au collecticiel que nous avons décrites précédemment n'ont pas pu être implantées uniquement à partir des interfaces de programmation. Les fonctions collecticielles qui ont présenté des difficultés sont surtout les fonctions d'écho graphique. L'animation, par exemple, a été implantée à l'aide de fonctions graphiques de bas niveau alors qu'il aurait été agréable de pouvoir faire une opération d'animation simplement au travers de l'appel d'une fonction de l'application (WidgetGraph ou Design/OA). En particulier, l'inconvénient de la solution actuelle est que durant une animation, le système ne permet pas à l'utilisateur d'interagir avec l'application ce qui peut-être contraignant lorsque l'animation devient trop longue. Un autre exemple de problème rencontré est celui de l'affichage de l'icône de verrouillage sur un objet, qui est aussi implanté à partir de la librairie graphique (Quickdraw ou Xlib), ce qui nous oblige à gérer le rafraîchissement de l'écran en concurrence de l'application.

Ces problèmes indiquent que les systèmes graphiques comme Quickdraw ou Xlib ne sont pas adaptés aux besoins des applications collecticielles, ce qui oblige à des compromis ou des solutions inutilement complexes. Cependant, il existe des solutions à ces problèmes, au moins à l'état de prototype. Ainsi, le système Whizz [9] permet d'integrer des animations dans les interfaces, et la boite à outils Xtv [3] permet de gérer plusieurs plans indépendants dans chaque fenêtre.

## Travaux dans le même domaine

Des travaux ont déjà été menés dans le domaine des applications collaboratives de dessin non structuré de type MacPaint [10,15]. Le dessin non structuré soulève des problèmes de nature différente du dessin structuré. Il n'est par exemple pas nécessaire de gérer les conflits puisque ceux-ci n'ont pratiquement aucune chance de se produire (deux utilisateurs ont peu de chance de modifier le même pixel du dessin). Ces applications sont souvent utilisées dans le cadre d'une réunion où le nombre d'utilisateurs, parfois situés dans la même salle, est restreint [7].

Un certain nombre de produits ont vu le jour dans le commerce [10,5]. Aspects [5] mérite d'être mentionné, puisqu'il permet l'édition de dessins structurés. Cependant le mécanisme de gestion des conflits qui consiste à verrouiller l'objet sur lequel travaille un utilisateur ralentit le temps de réponse de l'interface. De plus la structure du diagramme est plus simple que dans notre système, d'où des fonctionnalités collecticielles moins nombreuses.

Dans le domaine des éditeurs de textes, le système Grove [11] se rapproche de notre travail. Cependant, les fonctionnalités collecticielles sont moins développées. De plus Grove s'intéresse à des petits groupes d'utilisateurs.

Un certain nombre de boite à outils permettant d'écrire des applications reparties dans le domaine hétérogène [6,1], mais jusqu'ici aucun développement satisfaisant n'a été accompli en matière d'interface homme machine en milieu hétérogène.

## Conclusion

Nous avons présenté un outil de dessin collecticiel hétérogène. Notre expérience prouve qu'il est possible de construire une application collecticielle a partir d'une application mono-utilisateur, même en milieu hétérogène. De plus, les points délicats que nous avons soulignés dans la discussion apportent des informations utiles pour le développement de futurs outils afin de faciliter le développement du collecticiel.

Nous pensons continuer ce projet dans trois directions principales :

- Le système que nous avons présenté est en cours de développement, et les différentes fonctionnalités que nous avons présentées devront être évaluées avec des utilisateurs réels. Il apparaît d'ores et déjà qu'une interface multi-utilisateurs nécessite des techniques d'interaction spécifiques, notamment pour l'écho (animation, son, etc.).
- L'application sur un éditeur de texte des techniques appliqués à GroupDesign nous permettrait de valider notre architecture et les fonctionnalités collecticielles que nous avons définies dans une autre application.
- L'entrée d'utilisateurs en cours de séances collecticielles et la sauvegarde d'un document partagé doivent être étudiées afin de rendre le cours d'une session collecticielle au moins aussi simple que pour une application mono-utilisateur.

## Remerciements

Ce travail est en partie supporté par le PRC Programmation. Nous remercions Apple France qui nous a fourni l'équipement pour développer GroupDesign et MetaSoftware pour avoir mis à notre disposition le logiciel Design/OA à partir duquel est construit GroupDesign.

## Bibliographie

[1] Barbacci, M., R., DoubleDay, D., L., Weinstock, C., B., and Lichota, R., W., Durra: An integrated approach to software specification, modeling and rapid prototyping. Technical report CMU/SEI-91-TR-21, September 1991.

[2] Beaudouin-Lafon, M., The graph widget -- user's manual. Technical report, LRI, Université de Paris-Sud, France, June 1991. version 1.5.

[3] Beaudouin-Lafon M., Berteaud Y., Chatty S., Creating direct Manipulation Applications with Xtv. In *Proc. European X Window System Conference (EX)* , November 90.

[4] Beaudouin-Lafon, M., Karsenty, A., Transparency and Awareness in a Real-Time Groupware System. In *Proc. ACM Symposium on User Interface Software and Technology UIST'92* (Monterey, CA, November 1992).

[5] Biel, V., Groupware Grows Up. In MacUser, June 1991, pp. 207-211.

[6] Birman, K., Cooper, R., Joseph, T., Marzullo, K., Makpangou, M., Kane, K, Schmuck, F., and Wood, M., The ISIS System Manual. Technical report, The ISIS Project, Cornell University. September 1990.

[7] Buxton, W., Moran, T., Europarc's Integrated Interactive Intermedia Facility (IIIF): Early Experiences. In *IFIP* 1990, pp. 11-34.

[8] Card, S. K., Mackinlay, J. D., Robertson, G. G., The Information Visualiser, an Information Workspace. In *Proc. of the Conference on Computer Human Interaction* (CHI) '91 (New Orleans, Louisana, April 1991), pp. 181-188.

[9] Chatty, S., Integrating Animation with user Interfaces. In Engineering for Human Computer Interaction, *Proc. of IFIP WG2.7 Working Conference ,* Carson, J. and Unger, C. Editor

[10] Coleman, Dale, Timbuktu vs. Carbon Copy Mac: Close Race. In *MacWeek* (September 11, 1990), pp. 181-188.

[11] Ellis, C.A., Gibbs, S.J., and Rein, G.L., Groupware Some Issues and Experiences. In *Communication of the ACM*, January 1991, Vol. 34, No 1, pp. 39-58

[12] Gaver, W. W., Smith, R. B., O'Shea, T., Effective Sounds in Complex Systems: The Arkola Simulation. In *Proc. of the Conference on Computer Human Interaction* CHI '91 (New Orleans, Louisana, April 1991), pp. 85-90.

[13] Karsenty, A., Beaudouin-Lafon, M., An Algorithm for Distributed Groupware Applications. In *Proc. of the 13th International Conference on Distributed Computing Systems* ICDCS'93 (Pittsburgh, Pennsylvania, May 1993), à paraitre.

[14] Meta Software Corporation, *Design/OA Manual*, 150 CambridgePark Drive, Cambridge, MA, March 1989.

[15] Minneman, Scott L., and Sara A. Bly, Managing a Trois: a Study of a Multi-User Drawing Tool in Distributed Design Work. In *Proc. of the Conference on Computer Human Interaction* (CHI) '91 (New Orleans, Louisana, April 1991), pp. 217-224.

[16] Patterson, J., F., Comparing the programming demands of single-user and multi-user applications.In *Proceedings of the fourth Symposium on User Interface Software and Technology*, pages 87--94. ACM SIGCHI, ACM Press, November 1991.

[17] Stefik, M., Bobrow, D.G., Foster, G., Lanning, S., and Tartar, D. WYSIWIS revised: Early Experiences with Multiuser Interfaces. In *ACM Transactions on Office Information Systems*, Vol. 5, No 2, April 1987, pp. 147-186.

[18] Tronche, Ch., Transport des données dans le collecticiel hétérogène. In *Proc. Quatrièmes journées sur l'Ingénierie des Interfaces Homme-Machine* (IHM), 1992.