Applications collaboratives : exemples, architectures et services

Tronche Christophe

LRI (CNRS URA 410)
Bât. 490, Faculté d'Orsay
91405 ORSAY CEDEX
Tél: + 33 01 69 15 66 22
E-mail: tronche@lri.fr
WWW: http://tronche.lri.fr:8000/

RÉSUMÉ. *Les applications collaboratives sont des applications par lesquelles plusieurs utilisateurs collaborent (ou interagissent) simultanément. Plusieurs types d'applications collaboratives existent, mais les plus répandues et les plus représentatives sont les applications temps-réels, grâce auxquelles plusieurs utilisateurs agissent en même temps sur les mêmes objets. Cet article présente les principaux aspects techniques de ces collecticiels "synchrones" et des problèmes architecturaux qu'ils soulèvent.*

ABSTRACT. *Collaborative applications are programs that allow several users to simultaneously cooperate (or interact) with each other. Many kinds of collaborative applications exist, but the real-time groupware systems are the more widespread and representative ones. With real-time applications, several users manipulate the same objects at the same time. This paper presents the most outstanding technical and architectural issues raised by these synchronous groupware systems.*

MOTS-CLÉS : *applications collaboratives, collecticiels, interface homme-machine, applications réparties.*

KEYWORDS: *collaborative applications, groupware, User interface, distributed applications.*

Les applications collaboratives sont des applications par lesquelles plusieurs utilisateurs collaborent (ou interagissent) simultanément. Cette définition, due à Ellis et al. [EGR91], laisse largement ouvert le champ de l'interprétation : après tout, le courrier électronique est le principal outil que la plupart d'entre nous utilisent pour travailler en groupe (mettre au point des réunions ou écrire des articles à plusieurs, par exemple). Doit-on pour autant classer le courrier électronique dans la catégorie des applications collaboratives ? C'est un pas que franchissent certains auteurs comme J. Grudin [Gru88]. Cependant, beaucoup d'autres préfèrent se restreindre à un ensemble d'applications bien définies dans lesquelles les utilisateurs travaillent avec un couplage assez fort, c'est-à-dire simultanément, au moins pendant une partie de la durée de vie de l'application.

Cet article présente les principaux aspects techniques de ces collecticiels "synchrones" et des problèmes architecturaux qu'ils soulèvent, et donne quelques pistes sur les développements futurs prévisibles. Dans une première partie, nous indiquerons quelles sont leurs principales caractéristiques ; un certain nombre d'exemples permettra ensuite de donner une idée de la variété des recherches dans ce domaine ; dans une troisième partie, nous examinerons les principaux problèmes des collecticiels : l'architecture et le verrouillage avant de terminer sur une discussion de l'adaptation des réseaux aux applications collaboratives.

1. Caractéristiques des collecticiels

Les applications collaboratives sont souvent désignées en anglais par le vocable *groupware*, un mot que Pierre Lévy propose de traduire par *collecticiel* [Lév90]. L'évolution de cette branche de l'interaction homme-machine a suivi depuis son émergence il y a une dizaine d'années, l'augmentation des performances des machines, mais surtout celle des réseaux.

Le collecticiel emprunte donc à de nombreuses disciplines ou sous-disciplines, on peut citer :

– le multimédia,

– le réseau,

– les systèmes distribués, et les applications réparties en général,

– les interfaces homme-machine,

– la sociologie, la psychologie cognitive, etc.

1.1. *Une brève taxonomie*

Classiquement, on distingue les différentes catégories de collecticiels selon les trois axes de l'*espace*, du *temps* et de la *taille du groupe* (figure 1) [EGR91, Flo91].

Si on cherche à discrétiser, on peut dire que la taille du groupe peut être "petite" (quelques personnes à une dizaine de personnes), ou "grande" (une dizaine à plusieurs

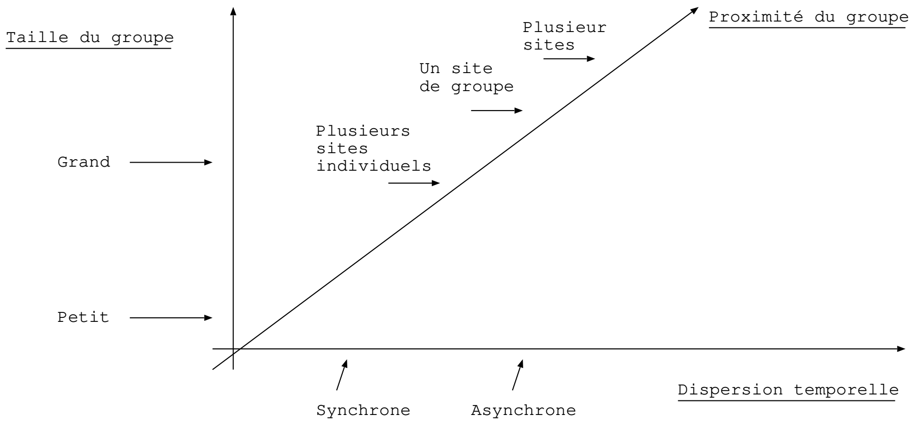

Figure 1. *Taxonomie*

milliers de personnes). La proximité peut être divisée en trois classes : “plusieurs sites individuels”, qui correspond à plusieurs utilisateurs, chacun derrière sa station de travail ; “un site de groupe”, qui correspond à une salle de conférence, et “plusieurs sites de groupes”, qui correspond à des salles de conférences et/ou des utilisateurs isolés dispersés dans le monde. Enfin, le *couplage* (c’est-à-dire le fait qu’une action d’un utilisateur soit rapidement perçue par les autres utilisateurs) peut être fort, relaxé ou faible. Au cours d’une même session, le couplage peut varier. Par exemple, dans le cas du télé-enseignement, on peut imaginer des périodes de cours où le couplage entre l’enseignant et les étudiants est fort, et des périodes de travaux dirigés où le couplage est fort au sein de petits groupes d’étudiants et faible entre les groupes.

1.2. *Le trèfle : production, coordination, communication*

Les recherches sur le collecticiel et les systèmes coopératifs traversent une période relativement dynamique. En France, elles ont donné naissance ces dernières années à un groupe de travail inter-PRC, le GT SCOOP [BLC95]. Le GT-SCOOP a utilisé avec succès pour décrire les différentes tâches liées à l’utilisation d’un collecticiel le trèfle imaginé par Ellis et al. [EW94] (figure 2). Selon ce modèle, les différentes fonctionnalités d’un collecticiel se décomposent en trois groupes, trois ”espaces” : la *production*, la *coordination* et la *communication*.

La production correspond à l’exécution effective de la tâche. Par exemple, dans le cas d’un éditeur de texte multi-utilisateur utilisé pour écrire un article à plusieurs, les fonctionnalités d’édition de texte font partie de l’espace de production. La coordination différencie une application mono-utilisateur d’une application multi-utilisateurs. Les mécanismes de gestion de la concurrence (que se passe-t’il si deux utilisateurs essaient de modifier simultanément le même mot du texte ?) appartiennent à cette seconde catégorie. Enfin, les praticiens du groupware, en s’appuyant sur la sociologie de la communication humaine (par exemple [Bat72]), ont mis en évidence la nécessité pour les utilisateurs impliqués dans une tâche de travail coopératif de disposer d’un canal de

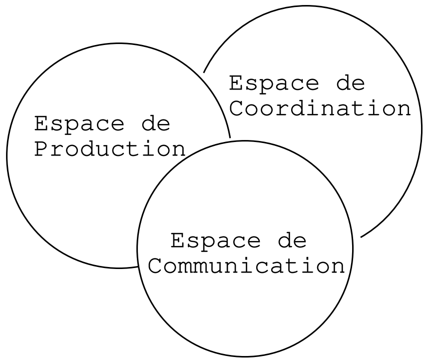

Figure 2. *Le trèfle du collecticiel*

communication qui leur permet de se coordonner [IK92]. En continuant avec l'éditeur partagé, un des auteurs pourrait par exemple avoir à demander à un autre "pourquoi as-tu fait cette modification ?". Aux premiers temps du groupware, la communication s'est faite le plus souvent par écrit soit par courrier électronique, soit à l'intérieur de l'application, comme dans Quilt [LFK88], un éditeur de texte permettant aux utilisateurs "d'annoter le texte dans la marge". Avec l'évolution de la technique, la communication a de plus en plus emprunté des canaux audio/vidéo analogiques, puis de la vidéoconférence numérique. Ces moyens de communication favorisent une interaction beaucoup plus directe et plus souple entre les utilisateurs.

2. Quelques exemple de collecticiels

Les chercheurs se sont d'abord focalisés sur des outils comme des agendas partagés et des outils de vote, les seuls que permettaient pratiquement les réseaux de l'époque et l'état de l'art en matière d'ingénierie des applications distribuées. Au fur et à mesure des progrès sont apparus des éditeurs multi-utilisateurs temps réels, avec lesquels chaque utilisateur voit peu ou prou ce que font les autres au fur et à mesure des modifications. Comme déjà signalé, l'expérimentation avec ces outils a fait apparaître les besoins de communication "hors-bande" entre les utilisateurs, c'est-à-dire la possibilité pour eux de communiquer hors de l'application, ce qui a conduit à des dispositifs de communication multimédia entre les participants. Ces mécanismes sont à leur tour devenus si intéressants que l'utilisation de l'ordinateur pour la communication est devenue une fin en soi, aboutissant aux "médiaspaces" et aux "salles de conférence virtuelles", dernier avatar du genre. Ces outils de conférence permettent de diffuser, outre l'image et le discours du conférencier, des diagrammes techniques, des vidéos, des sons. Mais elles offrent également des possibilités d'indexation des différentes parties d'une présentation [CH94], d'ajustement automatique de la scène, et bien sûr la possibilité pour les "spectateurs" de répondre, ce qui aboutit par exemple à des sessions de télé-enseignement assisté par ordinateur.

2.1. *Systèmes d'aide à la décision*

Ce furent les tout premiers outils collecticiels à apparaître. On trouve dans cette catégorie des outils de vote comme ceux de GroupSystems [NDV$^+$91], et des agendas multi-utilisateurs comme MPCAL et RTCAL (*Multi-Person CALendar* et *Real-Time CALendar*). MPCAL et RTCAL sont écrits en CLU et possèdent une structure centralisée. RTCAL notifie les utilisateurs concernés au fur et à mesure que des dates sont entrées.

LIZA [Gib89] est une boîte à outils permettant d'écrire du collecticiel (ayant une structure centralisée). Plusieurs applications collecticielles ont été développées avec LIZA afin de prouver son utilisabilité ; parmi eux: un outil de vote et et des outils montrant quels sont les utilisateurs en session et ce qu'ils font.

2.2. *Editeurs partagés*

Ce sont les systèmes les plus répandus. Ils correspondent à des programmes qui permettent à plusieurs utilisateurs d'éditer simultanément un même document. On peut les diviser principalement en éditeurs de textes et éditeurs graphiques. Les contraintes ne sont pas les mêmes, car s'il est possible de verrouiller une petite partie d'un texte, un simple trait peut couvrir (et donc forcer le verrouillage) de tous les éléments d'un dessin.

2.2.1. *Editeurs de texte*

InterNote [CBY89] est un programme d'annotation de documents hypermédias qui peut être considéré comme le niveau zéro du collecticiel. InterNote tourne sur le Macintosh. Les utilisateurs doivent se succéder devant la machine ou utiliser le programme à distance via AppleShare. Dans ce dernier cas, l'application ne pourra toujours être utilisée que par une seule personne. Néanmoins, l'application prend explicitement en compte la présence de plusieurs utilisateurs, par exemple en gardant la trace de l'auteur de chaque annotation. InterNote est une application centralisée.

VNS (*Virtual Notebook System*) [SCG89] est le niveau juste au-dessus. Il s'agit également d'un système de création de document hypermédia. Bien qu'il ne prenne pas plus en compte un accès multi-utilisateur qu'InterNote, VNS a été écrit sous Unix et utilise X Window. Il bénéficie donc "gratuitement" des possibilités d'accès réseau et multi-utilisateurs de Unix. Le système s'appuie sur une base de données multi-utilisateurs, une base étant stockée dans chacun des serveur de groupes de travail. Les serveurs communiquent à travers le réseau via un ordinateur particulier. L'application est donc centralisée.

Quilt [LFK88] est un système de création et d'annotation de documents (asynchrone) qui peut être considéré comme un véritable collecticiel : il prend en compte explicitement l'exploitation par plusieurs utilisateurs et dispose de mécanismes de notification entre les utilisateurs à plusieurs niveaux.

Suite [DC91] est également un véritable système collecticiel qui permet à plusieurs utilisateurs de remplir simultanément des formulaires. Suite est une application centralisée sous X.

CES (*Collaborative Editing System*) [GS87] est un autre éditeur partagé synchrone. Son principal intérêt est d'avoir été implémenté en Argus [LS83], un langage explicitement conçu pour écrire des programmes distribués. Cependant, Argus est basé sur l'hypothèse qu'il n'y a pas de contrainte temps-réel forte, ce qui n'est pas le cas pour les systèmes à manipulation directe.

L'éditeur GROVE a une certaine importance car il a été décrit dans [EGR91], un article séminal du domaine. Cet article est un des tout premiers à avoir mis en exergue les besoins spécifiques d'une interface multi-utilisateur. Il était par exemple proposé que, plutôt que les mots tapés par d'autres apparaissent brusquement, ils apparaissent d'abord dans une couleur imperceptible et "fonce" lentement pour que les autres utilisateurs puisse se préparer à l'apparition d'un nouveau texte.

L'éditeur multi-utilisateur GRIFFON [DVQ92] présente deux caractéristiques intéressantes : il s'appuie sur l'éditeur structuré GRIF et a été initialement construit audessus du langage distribué Guide. Pour GRIF, un texte est un objet structuré composé de sections, sous-sections, paragraphes, fragments, etc. Les utilisateurs peuvent verrouiller n'importe quel sous-arbre à leur convenance pour l'éditer sans être gêné par les autres utilisateurs.

2.2.2. *Editeurs graphiques*

Wscrawl [Wil95] est un éditeur domaine public orienté bitmap de ce genre. L'équipe du projet "GroupLab" de l'université de Calgary, emmenée notamment par Saul Greenberg, a également produit plusieurs éditeurs partagés temps-réels [GR95].

Nous avons réalisé dans notre équipe l'éditeur GroupDesign [KTBL93], un éditeur de dessin structuré multi-utilisateurs tournant simultanément sur Macintosh et stations Unix. GroupDesign est construit autour d'une architecture totalement répliquée: chaque utilisateur fait tourner une instance du programme qui connaît l'ensemble de l'état de l'application. Les modifications sont effectuées directement en local et envoyées aux autres instances. Un algorithme sophistiqué permet de gérer les conflits. L'aspect de rétroaction de groupe a été particulièrement étudié dans GroupDesign, par exemple, des zones de couleurs permettent à chaque utilisateur de savoir sur quelle partie les autres travaillent ; des animations permettent aux utilisateurs de ne pas être surpris par des modifications effectuées par d'autres utilisateurs, et des indications visuelles et sonores indiquent quel est le type d'activité de ces autres utilisateurs.

2.3. *Réalité virtuelle et jeux vidéos*

Les premiers jeux étaient plutôt statiques : RENDEZVOUS [Pat91] est un jeu de cartes à plusieurs utilisateurs, chacun ayant sa propre vue du jeu. RENDEZVOUS est une application centralisée. DBB (*Distributed Black Box*) décrit dans [DKR90] est un jeu dans lequel plusieurs joueurs doivent deviner la position de planètes dans un espace bidimensionnel en lançant dans cet espace des rayons qui y subissent la gravitation, et en observant la déviation. Chaque utilisateur ne voit qu'une partie du jeu et doit donc communiquer avec les autres (verbalement) pour obtenir les renseignements permettant au groupe d'aboutir.

Ces dernières années ont vu apparaître des jeux élaborés (du point de vue de l'ingénierie) dans lesquels plusieurs joueurs interagissent simultanément dans un milieu simulé. "Doom" a été un des premiers jeux de ce genre. Ils exploitent pleinement les réseaux locaux actuels (à fort débit et faible temps de latence) : à chaque étape de la simulation à temps discret, la position de chaque joueur et de tous les mobiles est émise vers toutes les autres stations. Bien que l'application soit bloquée entre chacune de ces "diffusions", ces jeux donnent une simulation de bonne qualité à environ 8 images par seconde, au prix d'une consommation importante de bande passante [doo94].

Les premières expériences de ces simulations d'environnement multi-utilisateurs ont été conduites dans le cadre des expériences de "réalité virtuelle". Un exemple est Rubber Rocks [CJK+92], bâti autour d'un moteur plus général décrit dans [LKL91]. Il s'agit d'un jeu qui se joue à deux par entrées multi-modales (gant de donnée, tracker, reconnaissance vocale, etc.) dans un monde simulé d'objets déformables qui rebondissent. Les événements en entrée (par exemple, le flot de données venant d'un gant de donnée, d'un tracker, d'une entrée vocale) sont traduits en événements de haut niveau ("prendre un objet") par des filtres. Un processus collecte tous les événements de haut niveau en entrée dans le système et pilote d'autres processus (simulation, interaction, rendu graphique). Les calculs liés à l'affichage sont répartis sur plusieurs stations pour des questions de performances, mais le système est fondamentalement centralisé.

2.4. *Vidéoconférence*

De tous les services qui supportent le travail collaboratif assisté par ordinateur, la vidéoconférence est sans aucun doute celui qui a le plus progressé ces trois dernières années. Il a donné lieu à l'émergence du concept de *médiaspace*. Un médiaspace est un réseau audio/vidéo contrôlé par ordinateur. Les premiers médiaspaces ont été créés à Bellcore [FKRR93], au Xerox PARC [BHI93] et à l'Université de Toronto dans le cadre du projet Telepresence [Bux91]. Nous avons nous-mêmes démarré notre propre médiaspace en 1993.

2.4.1. *Première génération : les médiaspaces analogiques*

Dans ces systèmes de première génération, la vidéo et le son sont analogiques. Chaque utilisateur dispose en plus de sa station de travail d'une caméra et d'un moniteur (figure 3), le tout formant un "nœud" du médiaspace. Un cross-bar audio/vidéo contrôlé par un ordinateur permet d'établir une liaison entre n'importe quelle paire d'utilisateurs. Le rôle de l'informatique dans ce cas est d'une part d'établir des liaisons pertinentes pour la tâche en cours (une notion qui varie énormément, par exemple le système *Cruiser* de Bellcore , peut établir des connexions au hasard), d'effectuer un certain nombre de contrôles d'accès (pas de connexion pendant une réunion) et possiblement de contrôler des caméras, dans le cas où le zoom ou l'orientation de la caméra peuvent être pilotés par ordinateur. Un service typique dans lequel l'ordinateur doit coordonner plusieurs dispositifs simultanément est le "glance", une connexion de quelques secondes vers un bureau pour savoir si son occupant est présent.

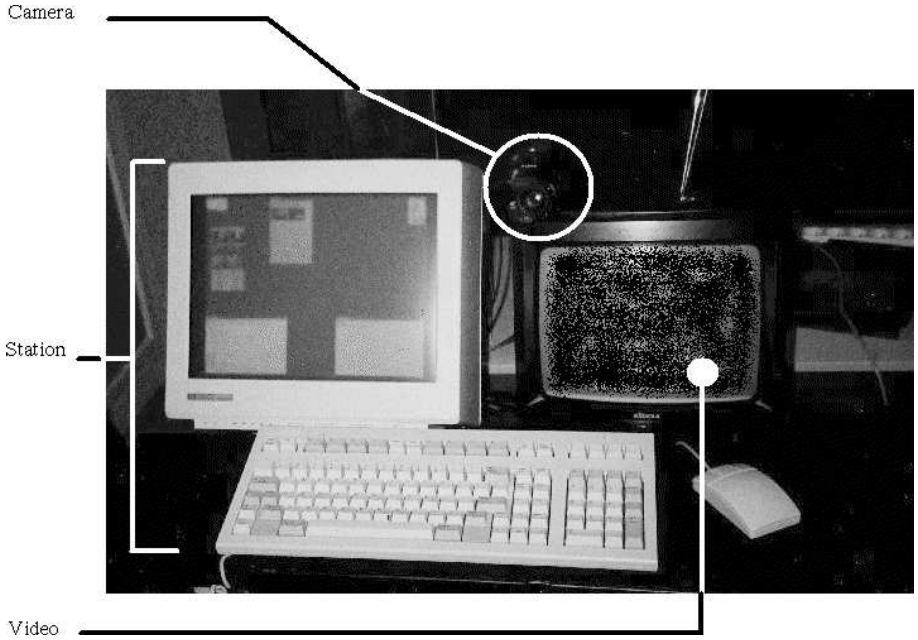

Figure 3. *Un nœud vidéo*

Ces systèmes ont rapidement fait la preuve de leur utilité... et de leurs limites. Les deux principales sont l'absence de transmission à grande distance (contrairement au courrier électronique), et l'impossibilité de se connecter simultanément à plusieurs personnes. Le premier problème peut être traité en utilisant des codecs et en transmettant le signal sur des liaisons fournissant un débit fixe, c'est-à-dire des liaisons RNIS à large bande. Cette solution a au moins deux inconvénients : le coût de la liaison et la rigidité du système, puisqu'on ne peut transmettre sur une liaison RNIS qu'une seule connexion, à comparer aux plus de cent personnes qui étaient desservies par le Cruiser.

Un écran de télévision n'affichant qu'une image et un nœud comportant rarement plus d'un écran vidéo, les seules liaisons possibles par des médiaspaces analogiques sont des liaisons de personne à personne, bien que des dispositifs électroniques permettent de multiplexer (analogiquement) jusqu'à quatre sources sur un seul écran. Voilà qui ne favorise guère les réunions !

2.4.2. *Deuxième génération : le tout-numérique*

L'intérêt pour les dispositifs analogiques disparaît avec l'arrivée de la génération actuelle des stations de travail équipées en standard de capacités de traitement vidéo, tels les stations Indys (et plus récemment O2) de Silicon Graphics, ou le PowerMac A/V d'Apple. Ces stations disposent, outre des convertisseurs analogique/numérique rapides, d'une puissance de calcul suffisante pour effectuer un certain nombre de traitement sur une image en temps réel, comme la compression du signal, ce qui en permet le transport en temps-réel au-dessus d'Internet. Internet devient alors une couche de

Figure 4. *Application de la vision artificielle à la vidéoconférence : la machine zoome sur les intervenants*

transport "banalisée". Ceci est effectué par des outils tels IVS [TH96] ou NV.

Le traitement numérique de l'information permet d'envisager des applications plus intéressantes que le simple transport. Par exemple, dans le système STREAMS de Bellcore [CH94], un exposé technique est enregistré. L'orateur peut choisir quelle est la partie à enregistrer parmi plusieurs flux (les transparents, l'assistance, l'orateur lui-même), pendant qu'un autre opérateur effectue le "cadrage" de l'orateur, en sélectionnant la portion pertinente de l'image.

La figure 4 illustre un travail d'application de la vision artificielle à la vidéoconférence en cours dans notre équipe : au centre se trouve l'image prise par la caméra. L'ordinateur repère la position des différents intervenant dans l'image et effectue un agrandissement de leur visage. Ceci préfigure le rôle futur de l'ordinateur, qui au cours des vidéoconférences, va devenir une sorte de "réalisateur automatique".

3. Architecture, Verrouillage

L'architecture des applications collaboratives a profondément évolué. Les premières applications étaient centralisées, un seul processus conservant à la fois l'état des données (par exemple des réservations de salles dans le cas d'un agenda partagé), mais aussi la vue que peut en avoir chaque utilisateur. Avec l'augmentation des besoins en ressources diverses, UC et mémoire sont apparues des architectures décentralisées, les processus communiquant en général par TCP/IP ou par AppleEvents (dans le cas des applications sur Mac Intosh). Puis, les applications étant de plus en plus exigeantes, en matière de performance réseaux notamment, elles tirent parti de nouveaux protocoles réseaux orientés vers le temps réel et d'extensions temps-réels de systèmes d'exploitation existant.

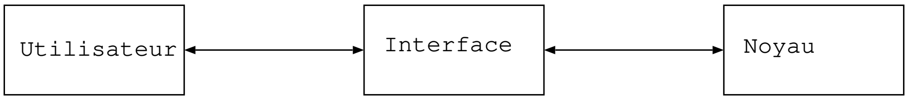

Figure 5. *Le modèle de Seeheim*

3.1. *Première étape : architectures centralisée*

Les premiers collecticiels reposaient sur une architecture *centralisée*. Dans ce type d'architecture, non seulement l'état global de l'application se trouve à un endroit bien défini (par exemple, au sein d'un seul processus), mais également la vue qu'a chaque utilisateur. Cette architecture a des inconvénients certains : une panne du contrôleur central, qui est un sérieux goulet d'étranglement, détruit toute la session. Les performances demandées à ce contrôleur sont telles qu'il est impensable d'avoir par exemple une communication vidéo numérique au-dessus d'une telle architecture. Néanmoins, sa simplicité lui vaut encore des partisans [PHR90, HBP$^+$93]. Dans ce type d'architecture, tous les évènements sont sérialisés. Ceci peut être vu comme un inconvénient, mais a l'avantage d'une grande simplicité au niveau de la structure de l'application. D'autre part, l'infrastructure nécessaire pour faire tourner l'application est réduite au strict minimum, ainsi dans [HBP$^+$93], il suffit que chaque utilisateur ait une station sous X.

3.2. *Architectures répliquées*

En général, les collecticiels temps-réel sont trop demandeurs en ressources machine pour qu'il soit raisonnable de les implémenter de façon centralisée. C'est certainement le cas si de la vidéoconférence est en jeu. Pour disposer des performances voulues, il est donc nécessaire de distribuer les applications. Si on utilise le modèle de Seeheim comme grille pour comprendre l'architecture d'une interface homme-machine, il existe deux composants qui peuvent être répliqués : le modèle et les vues.

3.2.1. *Le modèle de Seeheim*

Le *modèle de Seeheim* (figure 5) est fréquemment utilisé pour décrire les architectures d'applications interactives. Ce modèle stipule simplement que l'architecture idéalisée d'une application interactive se compose d'une part d'un utilisateur et de deux composants : le "noyau fonctionnel", qui regroupe des services spécifiques à l'application, et l'interface elle-même. Un système client-serveur donne une assez bonne idée des relations qu'entretiennent l'interface et le noyau fonctionnel au sens de Seeheim : l'interface envoie des ordres au noyau fonctionnel pour qu'il change son état, et reçoit des réponses. Le noyau fonctionnel est également censé notifier l'interface de ses changements d'état pour que l'interface puisse modifier la vue qu'elle présente de l'état du noyau fonctionnel. Si on prend l'exemple d'une application permettant de modifier une base de données, l'interface envoie des requêtes au serveur pour obtenir les informations à afficher, ou effectuer des modifications. L'affichage d'une barre de complétion pour indiquer quelle partie d'un fichier à été écrite (par exemple), et quelle partie reste à

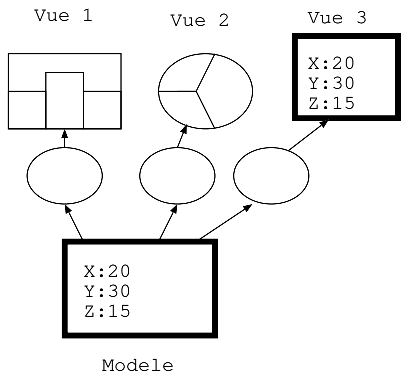

Figure 6. *MVC*

écrire nécessite que le noyau fonctionnel (ici le système de fichiers), notifie l'interface à intervalles réguliers pour que cette dernière puisse se mettre à jour.

Le modèle de Seeheim stipule en outre que l'utilisateur et le noyau fonctionnel ne communiquent pas directement, ils le font toujours au travers de l'interface. En particulier, l'utilisateur ne peut changer l'état du noyau fonctionnel par un procédé "magique" (non connaissable par l'interface).

Une incarnation du modèle de Seeheim est l'architecture *MVC* (Modèle- Vue- Contrôleur) de Smalltalk [KP88] (figure 6). Dans cette architecture, l'état abstrait de l'application est localisé dans une structure de donnée, ou un objet au sens de la programmation orientée-objet : le modèle. Différentes vues présentant les valeurs des données du modèle sous différentes formes peuvent être attachées à ce modèle. L'interaction entre les deux est réglée par un contrôleur, dont le rôle est de mettre à jour la vue lors d'un changement d'état du modèle (ce dernier notifie alors le contrôleur), ou de répercuter les actions de l'utilisateur (les modifications sur la vue) à l'intérieur du modèle.

Ce modèle est regardé comme vieillissant par la plus grande partie de la communauté de l'IHM, mais il a l'avantage de mettre nettement en évidence la séparation entre deux groupes de données bien définies, chacun susceptible d'être dupliqué dans l'architecture d'une application collecticielle : d'une part le modèle (les données), et d'autre part la vue qu'a chaque utilisateur de ces données. Il a été remarqué depuis longtemps que l'une des facilités, en quelque sorte, du collecticiel est qu'il existe une décomposition en différents processus "naturelle". Ceci conduit à une architecture dite "hybride". Dans cette architecture l'état (le contenu du noyau fonctionnel) est détenu par un processus qui le diffuse à autant d'autres processus que d'utilisateurs.

Le collecticiel a introduit la notion de *WYSIWIS* : "What You See Is What I See". Dans ce type d'application, tous les utilisateurs voient exactement la même chose, ce qui permet une collaboration fortement couplée. Mais ceci n'est pas toujours le plus commode, notamment parce qu'à priori, chaque utilisateur travaille sur "sa" portion de document, et préfère donc avoir une vision du document qui lui est propre. Implémenter ceci est commode avec une architecture hybride.

3.2.2. *Architecture purement répliquée*

Dans l'architecture hybride, lorsqu'un utilisateur effectue une modification, son processus vue envoie un message au processus modèle pour le notifier. Celui-ci doit alors le répercuter aux autres vues pour qu'elles se mettent éventuellement à jour. D'autre part, on est souvent amené, pour des raisons de performance, à diffuser à intervalle régulier l'état à chaque vue. Dans ce cas, l'intérêt du processus central, quoique conceptuellement satisfaisant, peut-être mis en question. Ceci conduit directement aux architectures *purement répliquées* [GRWB95]. Dans ce type d'architecture, le contrôleur central disparaît et son contenu est répliqué dans chaque vue. Cependant, cette architecture a ses propres inconvénients. Par exemple, dans le cas d'arrivée de "latecomers" (des utilisateurs qui arrivent après le début de la session), il suffit pour une architecture hybride de se connecter au contrôleur central pour obtenir l'état de l'application, alors qu'il faut procéder à une élection dans le cas de l'architecture hybride. Globalement, le surcroît de complexité de ces architectures ne peut être maîtrisé qu'en utilisant des outils logiciels qui gère de façon plus ou moins transparente la réplication ; mais ces bibliothèques ne pénètrent que lentement dans la communauté de l'IHM.

Un dernier problème avec l'architecture purement répliquée est qu'elle présuppose que tous les sites sont égaux, ce qui n'est pas forcément vrai. Par exemple, si un serveur vidéo doit être piloté, la machine qui en possède le contrôle est distinguée des autres. On règle le problème en ajoutant des serveurs supplémentaires pour chacune de ces ressources non partagées, ce qui ajoute encore à la complexité (et donc la fragilité) du système.

3.3. *Gestion des conflits*

Des conflits apparaissent inévitablement lorsque plusieurs personnes travaillent sur le même document. Par exemple, deux personnes peuvent vouloir déplacer un même objet dans deux directions différentes. Dans ce cas, il y a deux façons de régler le problème : laisser les utilisateurs le régler, ou implémenter dans l'application un mécanisme pour le régler. On parle dans le premier cas de protocole social, et dans le second de protocole matériel.

La première solution n'est pas forcément la plus mauvaise. Elle a au moins l'avantage d'être simple à implémenter : il n'y a rien à faire ! Il est noté également dans [DKR90] que ceci favorise le développement par les utilisateurs de leurs propres protocoles d'utilisation de l'outil collecticiel qui leur est confié. On peut également avertir l'utilisateur et le laisser régler le conflit. Par exemple, dans MPCAL (rappelons qu'il s'agit d'un système de gestion de calendrier), si des contraintes de rencontre sont incompatibles, les utilisateurs impliqués dans le conflit sont notifiés et doivent le résoudre. Dans le cas d'accès concurrents au même document, les utilisateurs sont avertis. Ils peuvent choisir d'ignorer l'avertissement, en sachant que le document sera alors modifié à leurs risques et périls [GS87].

On cherche cependant à laisser la machine régler le maximum de conflits. Pour leur résolution automatique, [EGR91] relève plusieurs méthodes possibles :

– verrouillage

– transactions

– les utilisateurs prennent la main chacun à leur tour ("floor")

– contrôleur centralisé

– détection de dépendances

– exécution réversible

– transformation des opérations

3.3.1. *Verrouillage*

Le problème du *verrouillage* se pose dans toutes les applications collecticielles, et particulièrement dans les éditeurs partagés. Etant donné que plusieurs utilisateurs peuvent accéder simultanément à un même document, il faut une méthode qui assure la consistence, c'est-à-dire que chaque utilisateur ait bien le même document. Ce problème de consistence se pose à deux niveaux que nous appellerons la consistence système et la consistence utilisateur. Afin de mettre la différence en lumière, imaginons que deux utilisateurs écrivent un texte à l'aide d'un éditeur ayant une structure répartie. Imaginons également que les deux utilisateurs, à la suite d'un manque de coordination, commencent ensemble à écrire l'introduction. Les caractères qu'ils tapent se mélangent pour former une bouillie incompréhensible. Si les deux utilisateurs voient la même bouillie, il y a *consistence système*. Mais un collecticiel digne de ce nom doit faire plus. Lorsque deux personnes utilisent une application pour écrire le même texte, elles s'attendent sans doute à ce que le texte résultant soit cohérent. Généralement, les éditeurs collecticiels divisent le texte en unités logiques (des fragments) auxquels sont appliqués des verrouillages [EGR91] : pour chaque fragment, un nombre illimité de personnes a le droit de lire si personne n'écrit, et si à un instant une personne écrit, le fragment ne peut être lu ni écrit par quelqu'un d'autre.

Ces verrouillages posent cependant des problèmes : ils prennent beaucoup de temps, il est difficile de savoir à quel niveau les faire ; faut-il verrouiller une lettre, un mot, un paragraphe ? Si on laisse le choix à l'utilisateur, il risque d'avoir beaucoup de décisions à prendre, ce qui amènera une charge cognitive qui devrait être évitée. Si on a des verrous trop fins (lettre par exemple), le temps des verrouillages et déverrouillages devient prohibitif, et on retombe sur les problèmes de "consistence-utilisateur" évoqué plus haut. Avec un grain trop fort, des utilisateurs risquent d'être inutilement bloqués parce que l'un d'entre eux accède à une portion du document. Choisir le bon niveau de verrouillage nécessite donc de connaître l'application.

Enfin un utilisateur qui ne rend pas le verrou (parce qu'il oublie de le faire par exemple) peut bloquer l'accès à une partie du document inutilement. Pour éviter ceci, le verrou peut être libéré après un certain temps, ou on peut donner aux autres utilisateurs une commande pour détruire le verrou [EGR91, GS87].

En ce qui concerne les éditeurs graphiques, pour lesquels on veut s'efforcer d'obtenir un aspect de manipulation directe, devoir mettre et retirer des verrous introduit un hiatus qu'il faut éviter (quoi que l'éditeur de graphe écrit avec LIZA fonctionne sur ce principe : chaque nœud du graphe a le statut public, partagé ou privé [Gib89]).

3.3.2. *Transactions*

C'est un modèle issu des bases de données. Une transaction est une unité de programme qui accède et met à jour (potentiellement) des variables. Au cours d'une transaction, toutes les variables concernées sont lues exactement une fois et sont écrites au plus une fois. On impose que si les données (qui représentent l'état de l'application) sont cohérentes avant la transaction, elles le sont encore après (mais peuvent ne pas l'être pendant). Le principal défaut de ce mécanisme est d'augmenter le temps de réponse de façon potentiellement inacceptable pour l'utilisateur, surtout dans le cas du collecticiel synchrone.

3.3.3. *"Floor control"*

Un moyen très simple pour éviter d'avoir à gérer les conflits est d'empêcher leur apparition. Avec le mécanisme des floors (mis en œuvre par exemple dans XTV) un seul utilisateur a la main. Si un autre utilisateur la veut, il doit la prendre ou la demander et attendre qu'on la lui donne. La main peut être retirée automatiquement à quelqu'un qui la détient depuis trop longtemps.

Le principal inconvénient de ce système est évidemment qu'à un instant donné, un seul utilisateur a la main. Cependant introduire plusieurs "mains" permet d'obtenir une certaine souplesse ([LA90] : une par application partagée ou une par fenêtre). On peut également autoriser plusieurs personnes à avoir la main. Le prix à payer pour tout ceci est évidemment un surcroît de complexité du logiciel, un temps de calcul plus élevé et une consommation plus importante de bande passante.

3.3.4. *Contrôleur centralisé*

Ceci correspond à l'architecture centralisée. Toutes les opérations sont effectuées séquentiellement par un unique processus. Ce modèle a deux avantages : on sait le mettre en œuvre, et c'est particulièrement commode avec X. Les inconvénients sont les inconvénients habituels des mécanismes centralisés : augmentation du temps de réponse, goulet d'étranglement, et manque de fiabilité, une panne du contrôleur central entraînant l'arrêt de toute la session.

3.3.5. *Détection de dépendances*

On remarque qu'en séquentialisant tout, on oblige de nombreuses opérations à s'exécuter l'une après l'autre alors qu'elles pourraient se dérouler en parallèle . Par exemple, pour un éditeur de dessin structuré, si deux personnes désirent déplacer deux objets distincts dans deux zones différentes, il n'y a aucune raison pour séquentialiser. En détectant les dépendances, il est possible d'effectuer le maximum d'opérations en parallèle et de ne séquentialiser que les opérations qui doivent l'être. Ceci requiert évidemment un logiciel d'autant plus complexe que les dépendances que l'on veut détecter sont à un niveau fin : il est facile de dire que bouger deux ellipses dans deux zones différentes ne posera pas de problème, en revanche, vérifier que leurs chemins ne se croiseront pas peut être très difficile et même impossible si les utilisateurs manipulent leurs objets en temps réel.

3.3.6. *Exécution réversible*

C'est une technique très élégante. On effectue immédiatement les opérations. Si une opération entre en conflit avec une des opérations précédentes, les dernières opérations sont défaites jusqu'au conflit, qui est résolu, puis on refait toutes les opérations qui ont été défaites. La mise en œuvre de cette technique est complexe. Il faut être capable de défaire toute opération. Une pile est nécessaire et il peut être difficile, voire impossible, de borner sa taille.

D'autre part cette technique pose des problèmes en termes de facteurs humain: supposons qu'il y ait un télépointeur (un curseur manipulé à distance par un utilisateur pour désigner des positions à l'écran de l'utilisateur local), l'utilisateur distant peut désigner un objet qui n'est pas encore apparu à l'écran.

GroupDesign utilise l'exécution réversible avec détection des dépendances pour minimiser la partie des opérations à défaire. Alain Karsenty a donné une condition suffisante pour qu'il soit possible de borner la taille de la pile des opérations à défaire [KBL93], et GroupDesign a été conçu pour respecter cette condition.

3.3.7. *Transformation des opérations*

Cette technique, qui est une extension des deux précédentes, est la plus sophistiquée. Lorsque deux opérations sont conflictuelles, on s'efforce de les composer pour obtenir le résultat voulu. Cette technique est relativement difficile à mettre en œuvre, et ne peut s'appuyer que sur une connaissance de la sémantique des objets manipulés.

3.4. *Couplage*

Les premiers systèmes multi-utilisateurs étaient strictement WYSIWIS (What You See Is What I See). Mais un couplage plus faible peut être nécessaire [DC91]. Par exemple, si un utilisateur remplit un formulaire électronique qui est observé par un autre utilisateur, ce dernier ne veut probablement pas être troublé par les fautes de frappe du premier : un champ ne devrait être transmis que lorsqu'il est rempli. Le couplage peut nécessiter un contrôle assez fin. Supposons un enseignant qui suit sur un écran le code écrit par quelques étudiants, au fur et à mesure qu'ils l'écrivent. L'enseignant pourra vouloir être averti de toutes les modifications apportées par des étudiants débutants, susceptibles de faire beaucoup d'erreurs ; pour des étudiants plus avancés, il ne veut être notifié que des modifications affectant la syntaxe, et pour des étudiants encore plus avancés, capables de comprendre seuls leurs erreurs de syntaxe, il ne voudra être notifié que des modifications susceptibles de changer la sémantique. Tout ceci permet à l'utilisateur de concentrer son attention sur les points cruciaux et de le décharger des détails sans importance (ce qui est évidemment toujours l'objectif en matière d'interface utilisateur). Dans le système Suite de [DC91], plusieurs personnes peuvent visualiser, remplir et modifier simultanément des formulaires (correspondant par exemple à un budget familial). Il est possible de configurer l'instant où les autres utilisateurs sont notifiés d'une modification : automatiquement, quand un champ est validé, sur demande de l'utilisateur, etc.

Dans Quilt [LFK88], plusieurs personnes peuvent travailler simultanément sur un article. Lorsqu'elles ont fini, le document est envoyé à un relecteur (sur demande des auteurs). Le relecteur peut annoter le document à l'attention des auteurs, ou pour son usage privé. Lorsqu'il a terminé, il le signife au système qui enverra aux auteurs les commentaires qui leur sont destinés. Il serait absurde d'envoyer ces commentaires avant que le relecteur ait fini, alors qu'il peut encore décider, par exemple, que l'un d'entre eux est superflu.

Dans le cadre des systèmes à manipulation directe, la situation est légèrement différente. La manipulation directe induit des contraintes temps-réel qui peuvent être importantes. Si on prend le cas d'un simulateur d'avion de chasse avec au minimum deux personnes, il est clair qu'en phase de combat toute action doit être instantanément perçue par toutes les personnes engagées dans la simulation. Pour un éditeur graphique, la situation est moins critique.

Dans tous les cas, l'important est que l'utilisateur ne puisse pas faire la distinction entre une action transmise instantanément et une action qui arrive avec un certain retard à l'interface d'un utilisateur. Dans le cas d'un simulateur de combat, un délai dans les transmissions mettra rapidement le système dans un état incohérent. Dans le cas d'un éditeur partagé (par exemple), il est possible de simuler une propagation instantanée de l'information [BLK92]. Avec une telle méthode cependant, toutes les vues de l'état du système ne sont pas dans un état cohérent à chaque instant. Il est même possible qu'au cours d'une session, deux vues ne soient jamais dans le même état (bien qu'il y ait convergence de toutes les vues vers le même état en temps fini si le système est au repos).

En conclusion, la question de savoir si la notification doit être sous le contrôle de l'utilisateur ou être effectuée automatiquement par le système ne peut être réglée sans une connaissance de l'application.

4. Réseau

Le temps des gros serveurs pilotant des dizaines de consoles avec un système d'exploitation "temps partagé" est loin derrière nous. Donc, on l'aura compris, une application collecticielle nécessite pour son fonctionnement un réseau. Une application collecticielle en activité est un ensemble de processus communicant.

4.1. *Le problème*

Outre la diffusion vers plusieurs personnes simultanément, la spécificité du collecticiel en matière de réseau est la coexistence de deux types de flux ayant des caractéristiques opposées : d'une part les flux multimédias, qui résident dans l'espace de communication, et de l'autre les données concernant la manipulation du document, qui sont dans l'espace de production. Les premières génèrent d'importantes quantités de données, mais la perte d'une partie d'entre elles (d'une image, par exemple) n'a pas d'impact déterminant sur l'exécution de la tâche. Les secondes génèrent un flux plus faible (dans le cas d'un éditeur de texte, par exemple, chaque utilisateur ne génère pas plus de quelques caractères à la seconde, alors que dans le même temps, une communication vidéo génère plusieurs dizaines de kilobits). En revanche la perte d'un seul bit (un caractère dans un mot), peut avoir un impact important.

4.2. *Quels protocole pour le collecticiel ?*

Dans le cas d'une application "locale", c'est-à-dire dont tous les éléments réseaux sont administrés par une seule entité, il est possible d'utiliser des protocoles "fermés" : Ethernet, IPX/SPX ou AppleTalk. En revanche si le système est "ouvert" (plusieurs architectures différentes sont susceptibles d'avoir à cohabiter), il faut utiliser les protocoles les plus répandus. Trois possibilités ont été largement exploitées pour le collecticiel : TCP, UDP et ATM. Les deux principaux critères de choix sont le nombre d'utilisateurs supposé pour l'application, et la nature des données transportées.

– TCP. Ce protocole a été critiqué comme étant trop lourd pour des transactions (requête/réponse), qui sont typiques du calcul distribué [Che88, DHN95]. On lui reproche également d'être "bloquant" : le débit peut ralentir brusquement, par exemple sur un début de congestion du réseau. Néanmoins, ses avantages sont certains : transport fiable des messages, possibilité de transporter des messages de n'importe quelle taille, utile par exemple pour la vidéo, efficacité globale au travers d'algorithmes tels que la détection du MTU le long des chemins et des algorithmes de contrôle de congestion sophistiqués [Jac88], et enfin la possibilité de garder la trace des utilisateurs accessibles au travers du "keep-alive". TCP n'est réaliste que pour un petit nombre d'utilisateurs (une dizaine au plus).

– UDP. Ce service est mieux adapté à la diffusion que TCP, comme le prouve l'existence du Mbone [Eri94]. UDP souffre cependant des inconvénients corrigés par TCP : pas de contrôle de flot, pas de correction des erreurs. La perte de paquet qui est tolérable dans certaines conditions pour la vidéo, ne l'est pas pour les flux données, dans ce cas ils doivent être corrigés par des algorithmes de type *leaky bucket* ou *forward error correction*. En dépit de ces inconvénient, UDP est la base de la plupart des programmes de transport de multimédia au-dessus d'Internet, et restera sans doute incontournable.

– ATM. Les réseaux ATM [Par94], censés fournir une qualité de service fixée à l'avance, semblent être le réseau idéal pour le travail collaboratif. Si la couche physique correspondante est rapide et fiable (fibre optique), ils offrent un débit suffisant pour le multimédia, et sont crédités de plus de la capacité de permettre le mélange de flux ayant des caractéristiques différentes. Cependant, pour le moment plusieurs inconvénients ont limité leur pénétration dans le monde du collecticiel : le coût, le manque d'interopérabilité entre les équipements provenants de différents constructeurs, le manque d'API permettant à une application de spécifier la qualité de service voulue, l'absence de "performances ATM" sur des réseaux qui contiennent des tronçons non-ATM, et l'absence d'interface ATM sur les réseaux fournis par les opérateurs de télécommunication.

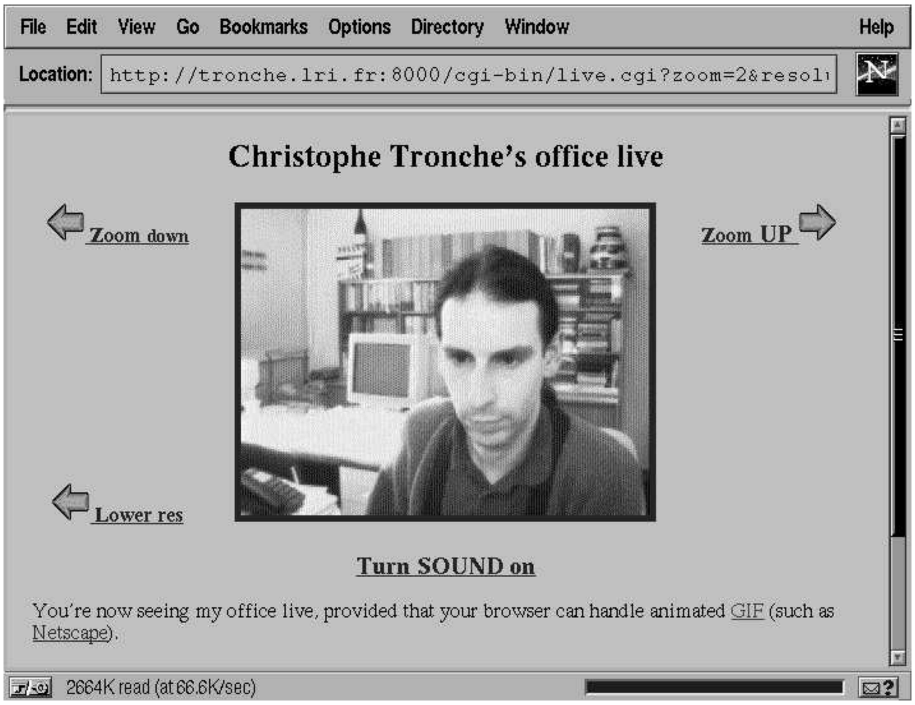

Figure 7. *Vidéo-conférence dans le WWW sans composant additionnel*

4.3. *Le problème de la compression des flux multimédia*

Tous les flux n'ont pas la même sensibilité aux distorsions introduites par le réseau (perte et permutation de paquets), et toutes les données n'ont pas la même importance à l'intérieur d'un même flux. Ici, la compression nécessaire pour utiliser efficacement la bande passante interfère avec la perte d'information possible avec certains protocoles. Comme le signale Partridge [Par94], les discussions sur la perte de paquets deviennent souvent émotionnelle: "Si on peut supprimer ce bit, mon codec ne l'aurait jamais envoyé".

Dans le cas par exemple du codage MPEG vidéo [Schft], certains blocs de données servent de support pour le codage d'un grand nombre d'images successives, alors que d'autres ne codent qu'une portion d'une image, la perte d'un bit dans les différents types de bloc n'a pas le même impact. Une des solutions très utilisées pour ce problème est l'encodage avec priorité [ABEL96] : on ajoute de la redondance pour pouvoir reconstituer le signal en cas de pertes de paquets (ou de cellules dans le cas d'un réseau ATM), mais on en ajoute davantage sur les cellules "stratégiques". Cependant, ces méthodes sont coûteuses en temps de calcul, et la compression du signal vidéo l'est également.

L'expérience que nous avons dans notre équipe nous conduit à penser que des algorithmes sophistiqués de compression ne se justifie pas dans le cas de la vidéoconférence, car ils consomment des ressources en calcul qui diminuent le nombre de trames par seconde, le seul indicateur véritablement pertinent. Par exemple, nous avons comparé sur une station Indy de SGI deux codecs transmettant de la vidéo, l'un en utilisant un encodeur MPEG domaine public (mpeg_encode 1.3), l'autre avec un encodeur GIF "maison". Dans le premier cas, en utilisant tout le temps UC disponible sur la machine, on obtient environ 2.5 images par seconde ; dans le second 16 images/seconde en consommant seulement 20 % de l'UC disponible. Le taux de compression plus faible du GIF conduit à un débit d'environ 80 Kilo-octets à la seconde pour des images 320 x 200, ce qui est important mais passe sans problème sur un réseau local. Certes, l'augmentation de la vitesse des machines rendra la compression de plus en plus rapide, mais comme le note Tanenbaum [Tan96], le débit disponible sur les réseaux augmente plus vite que la vitesse de l'UC des ordinateurs. Dans ces conditions, l'intérêt d'algorithmes sophistiqués de compression vidéo pour le transport temps-réel peut être remis en cause (la compression reste cependant intéressante pour le stockage ou le transport de données qui ne sont pas générées au vol).

5. Conclusion : La tendance du collecticiel

Les tendances qui s'esquissent pour le collecticiel suivront celles des réseaux d'une part, avec l'introduction de la mobilité, et d'autre part l'introduction "d'agents intelligents", c'est-à-dire que tous les utilisateurs collaborant autour d'une tâche ne sont pas des personnes, mais certains sont des programmes

Le changement le plus important dans l'architecture des collecticiels dans les années à venir sera sans doute le développement de composants collaboratifs "plug-and-play" soudés ensemble par le World-Wide-Web [CKTV96]. La plupart des composants sont déjà disponibles, comme la vidéoconférence (figure 7), mais l'utilisation du World-Wide-Web comme cadre permettra d'exploiter pour des applications de travail collaboratif d'autres outils tels que les "agents intelligents" [Nor94], l'ordinateur d'outil facilitant la participation de plusieurs personnes deviendrait alors participant lui-même !

Références

[ABEL96] A. Albanese, J. Bloemer, J. Edmonds, and M. Luby. Priority encoding transmission. *IEEE Transactions on Information Theory*, 42(6):1737–1744, november 1996. preliminary version in 35th IEEE FOCS, 1994.

[ACM89] ACM SIGCHI. *Hypertext'89*, November 1989. Proc. of Hypertext'89, November 5–8, Pittsburgh, Pennsylvania.

[Bat72] Gregory Bateson. *Steps to an Ecology of Mind.* Chandler, 1972.

[BHI93] Sara A. Bly, Steve R. Harrison, and Susan Irwin. Mediaspaces: Bringing people together in a video, audio and computing environment. *Comm. ACM*, (36):28–47, January 1993.

[BLC95] Michel Beaudouin-Lafon and Joëlle Coutaz. Scoop / rapport de recherche 1994-1995. Technical report, SCOOP working group, November 1995.

[BLK92] Michel Beaudouin-Lafon and Alain Karsenty. Transparency and awareness in a real-time groupware system. In *UIST'92*. ACM, ACM Press, November 1992. Proceedings of the ACM Symposium on User Interface Software and Technology.

[Bux91] William Buxton. Telepresence: Integrating shared task and personnal spaces. In Hendriks [Hen91], pages 27–36.

[CBY89] Timothy Catlin, Paulette Bush, and Nicole Yankelovich. Internote: Extending a hypermedia framework to support annotative collaboration. In *Hypertext'89* [ACM89], pages 365–378. Proc. of Hypertext'89, November 5–8, Pittsburgh, Pennsylvania.

[CH94] Gil Cruz and Ralph D. Hill. Capturing and playing multimedia events with STREAMS. In *Proc. ACM Multimedia 94*, pages 193–200, 1994.

[Che88] David R. Cheriton. The V distributed system. *Comm. ACM*, 31(3):314–333, March 1988.

[CJK+92] Christopher Codella, Reza Jalili, Lawrence Koved, Bryan J. Lewis, Daniel T. Ling, James S. Lipscomb, David A. Rabenhorst, Chu P. Wang, Alan Norton, and Greg Turk. Interactive simulation in a multi-person virtual world. In Penny Bauersfeld, John Bennett, and Gene Lynch, editors, *CHI'92 Conference Proceedings*, pages 329–334. ACM SIGCHI and SIGGRAPH, May 1992.

[CKTV96] Paolo Ciancarini, Andreas Knoche, Robert Tolksdorf, and Fabio Vitali. Pagespace: An architecture to coordinate distributed applications on the web. In *Fifth International World Wide Web Conference*, May 1996.

[DC91] Prasun Dewan and Rajiv Choudary. Flexible user interface coupling in a collaborative system. In Robertson [Rob91], pages 41–48.

[DHN95] Mark Debbage, Mark B. Hill, and Denis A. Nicole. An interface to a reliable packet delivery service for parallel systems. *Trans. Parallel Distrib. Sys.*, 6(4):400–411, April 1995.

[DKR90] C. De Koven and T. Radhakrishnan. An experiment in distributed group problem solving. In Simon J. Gibbs and A.A. Verrijn-Stuart, editors, *Multi-User Interfaces and Applications*, pages 61–76. IFIP WG 8.4, North-Holland, September 1990.

[doo94] *Data Communications*, page 34, May 1994.

[DVQ92] Dominique Decouchant, Irene Vatton, and Vincent Quint. L'édition coopérative de documents avec griffon. In *Actes du Colloque IHM'92*, Paris, 1992.

[EGR91] Clarence A. Ellis, Simon J. Gibbs, and Gail L. Rein. Groupware: some issues and experiences. *Comm. ACM*, 34(1):38–58, January 1991.

[Eri94] Hans Eriksson. Mbone: The multicast backbone. *Comm. ACM*, 37:54–60, 1994.

[EW94] Clarence A. Ellis and J. Wainer. A conceptual model of groupware. In *ACM CSCW 94 conference on Computer Supported Cooperative Work*, pages 79–88, 1994.

[FKRR93] Robert S. Fish, Robert E. Kraut, Robert W. Root, and Ronald E. Rice. Video as a technology for informal communication. *Comm. ACM*, (36):48–61, January 1993.

[Flo91] Gert H. Florijn. Groupware research at SERC. In Hendriks [Hen91], pages 77–87.

[Gib89] Simon J. Gibbs. LIZA: An extensible groupware toolkit. In *CHI'89 proceedings*, pages 29–35. ACM, May 1989. Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, (Austin, Texas, April 30 – May 4).

[GR95] Saul Greenberg and Mark Roseman. *Groupware Real-Time Drawing*, chapter 8: GroupKit: A groupware toolkit for building real-time conferencing applications. Greenberg, Saul and Hayne, Stephen and Rada, Roy, 1995.

[Gru88] Jonathan Grudin. Why CSCW applications fail: Problems in the design and evaluation of organizational interfaces. In *Proc. Of CSCW'88*, pages 85–93, 1988. Proc. of the ACM Conference on Computer-Supported Cooperative Work.

[GRWB95] Saul Greenberg, Mark Roseman, Dave Webster, and Ralph Bohnet. *Groupware Real-Time Drawing*, chapter 3: Human and technical factors of distributed group drawing tools. Greenberg, Saul and Hayne, Stephen and Rada, Roy, 1995.

[GS87] Irene Greif and Sunil Sarin. Data sharing in group work. *ACM Transactions on Office Information Systems*, 5(2):187–211, April 1987.

[HBP+93] Ralph D. Hill, Tom Brinck, John F. Patterson, Steven L. Rohall, and Wayne T. Wilner. The rendezvous language and architecture. *Comm. ACM*, 36(1):62–67, January 1993.

[Hen91] P.R.H. Hendriks, editor. *The potential of team and organizational computing*. Software Engineering Research Centre (SERC), SERC, 1991. Proceedings of Groupware 1991, Utrecht, The Netherlands.

[IK92] Inoshiro Ishii and M. Kobayashi. Clearboard: a seamless medium for shared drawing and conversation with eye contact. In *ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 525–532, May 1992.

[Jac88] Van Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM'88 Symposium on Communications Architectures and Protocols*, volume 18, pages 314–329, 1988. Part of ACM SIGCOMM Computer Communications Review.

[KBL93] Alain Karsenty and Michel Beaudouin-Lafon. An algorithm for distributed groupware applications. In *Proc. of the 13th International Conference on Distributed Computing Systems ICDCS'93*, 1993.

[KP88] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view-controller user interface paradigm. *Journal of Object-Oriented Programming*, 1(3):26–49, August/September 1988.

[KTBL93] Alain Karsenty, Christophe Tronche, and Michel Beaudouin-Lafon. GroupDesign: Shared editing in a heterogeneous environment. *Usenix Journal of Computing Systems*, 6(2):167–192, 1993.

[LA90] J. Chris Lauwers and Lantz Keith A. Collaboration awareness in support of collaboration transparency: Requirements for the next generation of shared window systems. In J. Carasco Chew and J. Whiteside, editors, *Human Factor in Computing Systems*, pages 303–311. ACM, April 1990.

[LFK88] Mary D. P. Leland, Robert S. Fish, and Robert E. Kraut. Collaborative document production using Quilt. pages 206–215, 1988.

[LKL91] Bryan J. Lewis, Lawrence Koved, and Daniel T. Ling. Dialogue structures for virtual worlds. In Robertson [Rob91], pages 131–136.

[LS83] Barbara Liskov and Robert Scheifler. Guardians and actions: Linguistic support for robust, distributed, programs. *ACM Transactions on Programming Languages and Systems*, 5(3):381–404, July 1983.

[Lév90] Pierre Lévy. *Les technologies de l'intelligence*. Ed. La découverte, 1990.

[NDV+91] J.F. Nunamaker, A.R. Dennis, J.S. Valacich, D.R. Vogel, and J.F. George. Electronic meeting systems to support group work. *Comm. ACM*, 34(7):40–61, july 1991.

[Nor94] Donald A. Norman. How might people interact with agents. *Comm. ACM*, 37:68–71, July 1994.

[Par94] Craig Partridge. *Gigabit Networking*. Number ISBN 0-201-56333-9. Addison-Wesley Publishing Company, 1994.

[Pat91] John F. Patterson. 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.

[PHR90] John F. Patterson, Ralph D. Hill, and S. L. Rohall. Rendezvous: An architecture for synchronous multi-user applications. In *Proceedings of the third conference on Computer Supported Cooperative Work (CSCW'90), Los Angeles, CA*. ACM SIGCHI, ACM Press, New York, October 1990.

[Rob91] Scott P. Robertson, editor. *Reaching through technologies*. ACM, May 1991. Proc. of the ACM conference on human factors in computing systems, New Orleans, Louisiana, April 27–May 2, 1991.

[SCG89] Frank M. III Shipman, R. Jesse Chaney, and G. Anthony Gorry. Distributed hypertext for collaborative research: The virtual notebook system. In *Hypertext'89* [ACM89], pages 129–135. Proc. of Hypertext'89, November 5–8, Pittsburgh, Pennsylvania.

[Schft] Heiner (main referee) Schomaker. Generic coding of moving pictures and associated audio, recommendation h.262. Technical Report ISO/IEC 13818-2, iso, November 1993 (draft).

[Tan96] Andrew Tanenbaum. *Computer Networks (3rd ed.)*. Prentice-Hall, 1996.

[TH96] Thierry Turletti and Christian Huitema. ideoconferencing in the internet. *ACM / IEEE Transactions on Networking*, pages 340–351, June 1996.

[Wil95] Brian Wilson. *Groupware Real-Time Drawing*, chapter 7: Wscrawl 2.0: A Shared Whiteboard Based on X-Window, pages 129–141. Greenberg, Saul and Hayne, Stephen and Rada, Roy, 1995.