THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE FLANDRES ARTOIS

pour l'obtention du titre de

DOCTEUR

en Productique : Automatique et Informatique Industrielle

par

Pierre VILERS

ETUDE D'UN SYSTEME DE TELECONFERENCE TEMPS-REEL APPLICATION A UN JEU DE ROLES EN GESTION DE PRODUCTION

Soutenue le 21 Février 1992 devant la commission d'examen :

Madame, Messieurs

J.M. TOULOTTE	*Président*	*Professeur à Lille I*
P. PREVOT	*Rapporteur*	*Professeur à INSA Lyon*
M. MERIAUX	*Rapporteur*	*Professeur à Lille I*
A. DERYCKE	*Directeur*	*Professeur à Lille I*
B. CANTEGRIT	*Co-Directrice*	*Maître de Conférence à Lille I*
J.M. GEIB	*Examinateur*	*Maître de Conférence à Lille I*
A. SORTON	*Invité*	*Professeur IUT Poitiers*

Je tiens à remercier Monsieur le Professeur Jean-Marc Toulotte qui me fait l'honneur de présider le jury de cette thèse et qui m'a régulièrement conseillé au cours de ce travail.

Je remercie les Professeurs Michel Meriaux et Patrick Prevot d'avoir accepté d'être rapporteurs de cette thèse et d'avoir consacré du temps à son examen.

J'exprime ma profonde gratitude au Professeur Alain Derycke, mon Directeur de Recherche, pour m'avoir accueilli dans son équipe. Je lui suis très reconnaissant pour ces trois années de recherche pendant lesquelles il m'a toujours fait bénéficier de ses conseils et de son expérience.

Je remercie Brigitte Cantegrit pour le soutien qu'elle m'a toujours manifesté.

Je remercie Jean-Marc Geib pour avoir bien voulu être examinateur de cette thèse.

Je remercie Alain Sorton d'avoir accepté de faire partie de ce jury et pour m'avoir conseillé pour la réalisation de la partie applicative du système.

Je tiens également à adresser tous mes remerciements à Claude Viéville pour sa gentillesse et l'aide qu'il m'a toujours apportée tout au long de ce projet.

J'exprime aussi mes remerciements à toute l'équipe NOCE : Hervé Cabre, Danièle Clément, Pascal Croisy, Jean-Noël Gers et Frédéric Hoogstoel pour leur collaboration et l'ambiance chaleureuse qu'ils ont su créer. Je voudrais également remercier Gérard Martin et Marie-Hélène Bricout pour les différents échanges que nous avons pu avoir.

Je remercie Isabelle Logez pour sa disponibilité, sa gentillesse et le travail qu'elle a fourni pour réaliser ce document.

SOMMAIRE

INTRODUCTION GENERALE

Le travail que nous présentons dans ce mémoire est l'aboutissement d'une recherche dans le domaine des C.S.C.W. (Computer Supported Cooperativ Work) menée au sein du Laboratoire Trigone en collaboration avec le Laboratoire d'Automatique de Lille I.

Il s'inscrit dans le cadre d'un programme inter-laboratoires sur la communication. Ce programme de recherche a comme axe principal, le travail coopératif supporté par ordinateur dans sa forme la plus générale (salle de réunion spécialisée, télé-consultation, télé-expertise et télé-tutorat).

En effet, le CUEEP (Centre Université-Education d'Economie Permanente) pratique dans ses centres, depuis des années, un enseignement véritablement "ouvert" qui permet d'offrir aux apprenants un maximun de facilités dans leurs études. Les paramètres sur lesquels joue cette "ouverture" sont les suivants : la durée, le rythme et l'espace. Il faut cependant garantir que cette nouvelle génération de systèmes ouverts puisse satisfaire un large choix de stratégies d'apprentissage (télé-enseignement et auto-formation). Dans la génération précédente des systèmes d'enseignements ouverts, l'accent avait été mis essentiellement sur la conception de supports pédagogiques multimédia qui pouvaient se substituer à l'enseignant. L'interactivité étant relativement faible, la participation des apprenants à des périodes de regroupement permettait de combler cette lacune.

L'introduction des nouvelles technologies éducatives, en particulier, celles basées sur l'informatique peuvent améliorer de façon spectaculaire l'interactivité de l'apprenant et du média (vidéodisque,CD/ROM et les tuteurs intélligents) mais deviennent encore plus efficaces lorsqu'elles offrent des facillités de communication/conversation. Cela permet aux apprenants de communiquer entres eux et d'accéder aux tuteurs. Ces différentes formes de coopération sont à notre sens très importantes surtout dans un contexte d'individualisation accrue. Effectivement des recherches ont démontré que des apprenants qui travaillent en groupe réalisent mieux les tâches assignées que ceux travaillant seuls [GARI 80].

Notre équipe de recherche s'intéresse donc particulièrement aux nouveaux outils pour la communication éducative qui mettent en jeu deux catégories de systèmes qui

différent en fonction du type de communications supporté : les communications synchrones (ou en temps réel) et les communications asynchrones (ou en temps différé).

Le projet qui nous a été confié est relatif à l'étude et au développement d'une plate-forme expérimentale spécialisée dans le support de systèmes de conférence temps-réel multimédia. Elle constitue une première étape pour la conception et l'expérimentation d'applications coopératives.

Le premier chapitre est consacré à une présentation générale des systèmes informatiques qui permettent le travail coopératif. Nous faisons dans un premier temps, une taxinomie des différents systèmes qui existent mais nous nous interesserons, par la suite, uniquement aux systèmes de conférence temps réel. Nous dégagerons et détaillons alors, les principaux concepts qui les caractérisent.

Dans le second chapitre, nous présenterons l'architecture que nous avons retenue pour notre plate-forme de conférence temps-réel. Nous commencerons tout d'abord par expliquer les modifications que nous avons apporté à l'environnement de développement choisi (Smalltalk-80) pour permettre une extension distribuée nécessaire au partage d'objets entre images Smalltalk différentes. Nous justifirons nos choix sur le modèle d'architecture adopté et nous développerons ensuite les mécanismes sous-jacents qui sont à la base de l'implémentation du système de gestion de la conférence et des applications coopératives.

Le troisième chapitre donnera un exemple d'application coopérative que nous avons développée. Il s'agit d'une simulation distribuée de la gestion de production d'un atelier de fabrication qui permet d'apprécier les avantages et les inconvénients de plusieurs méthodes de planification (méthodes MRP et Kanban). Nous montrerons comment une application peut être analysée pour être ensuite insérée dans notre système de conférence. Cette simulation est la transposition d'un jeu à vocation pédagogique appelé "Jeu du Kanban" utilisé dans des enseignements de type licence à l'université de Poitiers.

Dans le dernier chapitre, nous nous proposons de faire la critique de notre système en donnant les domaines de recherche qu'il faut approfondir. Cela nous permettra de faire le point avec les plus récents travaux dans le domaine du travail coopératif supporté par des ordinateurs.

I - PRESENTATION GENERALE

1.1 - Introduction

Dans notre société et en particulier dans les entreprises, la **Communication** entre individus est devenue un des défis à relever. En effet, le modèle Taylorien d'Organisation Scientifique du Travail qui tend à décomposer en tâches individuelles un projet global est de plus en plus remis en cause et délaissé au profit du travail de groupe.

Il est toujours indéniable que pour réaliser certaines tâches, des personnes doivent exécuter des micro-tâches individuellement. Cependant ces dernières seront ré-intégrées dans des projets plus vastes car la majorité des tâches à remplir nécessite souvent une participation collective.

Nous retrouvons par exemple cela dans des processus de prise de décisions qui nécessitent l'avis de plusieurs personnes, dans la réalisation d'applications informatiques où il est actuellement rare vu leurs ampleurs qu'une seule personne soit chargée de toutes les phases du projet, dans le management participatif où chacun est tenu de collaborer quelque soit son rang, etc.. .

Pour parvenir à une communication efficace, il faut que des **structures de communications** soient mises en place de façon adéquate. C'est à dire, qu'elles soient adaptées aux activités engendrées par le travail à effectuer et qu'elles favorisent également la synergie du groupe.

Les systèmes et les environnements informatiques autorisent déjà, depuis de nombreuses années, des échanges d'informations entre des individus. Cependant, ces systèmes ne permettent pas des communications directes entre les utilisateurs. Ces derniers ont simplement accès à des environnements partagés qui sont généralement associés à des bases de données conçues sous une forme d'architecture appelée Client/Serveur. Certains de ces systèmes nous sont familiers comme par exemple les systèmes de gestion bancaires qui permettent à des banquiers situés sur des positions géographiques différentes (départementales ou nationales) d'obtenir des informations. Les systèmes de réservation de places à la S.N.C.F. et les systèmes de partage de fichiers sur réseaux locaux font également parties de cette catégorie.

Il est important de signaler que sur ces systèmes, il n'existe pas d'outils offerts aux utilisateurs pour coordonner leurs différentes tâches.

Nous voyons à travers ces exemples et notre expérience quotidienne que cela n'est pas suffisant pour réaliser un travail réellement coopératif. Pour combler ces lacunes, une nouvelle discipline de recherche s'est développée depuis quelques années. Il s'agit du **travail coopératif supporté par ordinateur** (Computer Supported Cooperative Work - **CSCW**). Les systèmes CSCW sont conçus pour permettre des communications d'hommes à hommes par l'intermédiaire d'ordinateurs connectés en réseaux.

Ainsi, le CSCW se situe comme un sous-ensemble des systèmes informatiques.

Figure 1.1. *Différents niveaux des systèmes de communications informatiques.*

Ces systèmes englobent un large éventail d'outils qui diffèrent dans la nature des interactions et dans la localisation des participants. En effet, les interactions autorisées peuvent être en temps réel (synchrones) ou en temps différé (asynchrones). Quant aux participants, ils peuvent se trouver dans une même salle (il s'agit alors d'outils spécialisés dans le support de réunion conçus pour un type d'activité bien précis), dans un même établissement (le Group-Network), ou à des positions géographiques très éloignées. Il est à noter que pour les deux derniers cas, la distinction réside essentiellement dans le type de réseau employé (Réseaux locaux ou Réseaux publics distants).

Le schéma suivant caractérise les différents cas de figure que l'on rencontre.

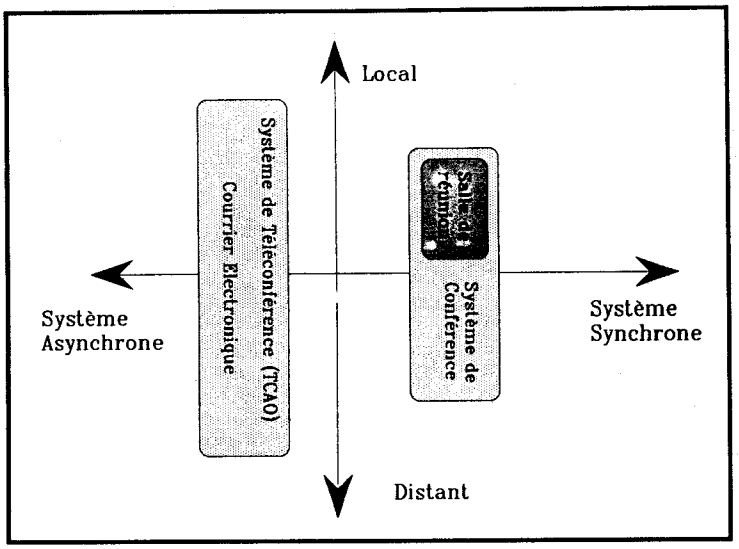


Figure 1.2. *Différentes formes de CSCW*

Depuis quelques années, différents prototypes caractérisant les quatre cas du schéma précédent ont été expérimentés. On ne peut pas dire qu'ils aient résolu l'ensemble des problèmes mais ils ont permis de cerner les difficultés liées à une utilisation collective. Ils ont, par là même, montré que pour évoluer, cette discipline avait besoin de compétences dans les spécialités suivantes : interface homme/machine, psychologie, sociologie, science cognitive, informatique.

A l'intérieur de cet ensemble du C.S.C.W. une distinction est faite pour une catégorie de systèmes. Il s'agit des **Groupwares** ou des **Collecticiels** (traduit par Pierre Levy) [BEAU 91]. Ces systèmes ont comme point commun d'être conçus pour des projets dans lesquels toutes les personnes impliquées ont une tâche commune à accomplir. Ils doivent également offrir des mécanismes de coordination entre les participants.

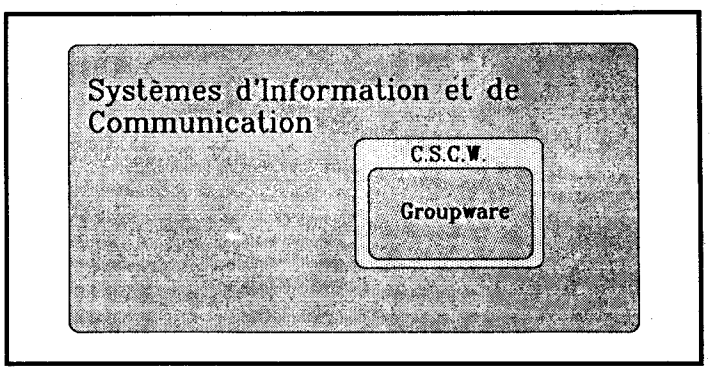


Figure 1.3. *Différents niveaux des systèmes informatiques.*

La définition du Groupware donné par Ellis [ELGI 91] est la suivante : "*Computer-based systems that support groups of people engaged in a common task and that provide an interface to a shared environment*". Deux des aspects très importants du Groupware sont l'environnement partagé et la tâche commune. Un groupware peut être vu comme un logiciel où l'utilisateur serait remplacé par un groupe. Donc, les systèmes informatiques multi-utilisateurs qui fonctionnent suivant un mode appelé "temps partagé" sont à exclure, car ils ne permettent que le travail d'utilisateurs indépendamment les uns des autres. En effet, les terminaux utilisés sont connectés à des machines virtuelles isolées.

Les systèmes pour lesquels le Groupware semble être particulièrement adapté sont les suivants:

. *Les systèmes support d'aide à la décision.*

Lorsque l'on parle des systèmes d'aide à la décision, on les associe très souvent à des systèmes experts qui classiquement sont mono-utilisateurs.

Pour le Groupware, nous entendons par système d'aide à la décision, des outils qui permettent la coordination et la collaboration de plusieurs personnes qui ne se trouvent pas au même endroit mais qui doivent coopérer afin de prendre une décision. Le terme anglo-saxon qui est souvent employé est GDSS (Group Decision Support System). Comme chacun des acteurs détient généralement une partie de l'information et/ou du savoir, il devra la partager avec le reste du groupe pour parvenir à une solution.

Ces systèmes englobent les salles de réunions informatisées dans lesquelles les participants sont rassemblés dans une même pièce mais devant une station de travail.

En effet, bien que le marché de la micro-informatique se soit considérablement développé ces dernières années, l'utilisation de l'ordinateur dans les réunions reste anecdotique car elle se trouve généralement limitée à la présentation de données ou à une démonstration.

Jusqu'à maintenant, les participants ont toujours tendance à privilégier : tableaux noirs, rétroprojecteurs, projecteurs diapo et vidéo pour présenter leurs documents dans les réunions.

Le tableau noir reste un élément de la réunion très important car il offre un espace de travail partagé par l'ensemble des participants. Il leur fournit une mémoire partagée et il facilite également la compréhension du groupe en permettant le placement de textes et de dessins (ne dit-on pas qu'un bon dessin vaut mieux qu'un long discours ?). Il permet aussi à certains

instants de focaliser l'attention du groupe et de matérialiser sa progression.

Cependant, ses inconvénients nous sont aussi familiers. L'espace de travail alloué est limité et son extension peu facile. Le réarrangement des annotations n'est pas pratique car il nécessite un effacement préalable et une réécriture. De plus, le stockage de l'information n'est pas possible, car les salles de réunion sont généralement utilisées par plusieurs groupes, obligeant ainsi le nettoyage et l'effacement des tableaux à la fin de chaque réunion. Le stockage de l'information reste alors un problème primordial pour les sujets nécessitant plusieurs séances. Une légère exception est faite pour les "paperboards" dont les feuilles sont récupérables et utilisables pour les réunions suivantes.

C'est pour pallier ces lacunes que des études ont été entreprises sur des salles de réunions informatisées [STFO 87].

. *Les outils coopératifs de conception.*

Les outils coopératifs de conception permettent à différentes personnes attachées à la réalisation d'un projet de collaborer pour le mener à bien. En effet, à l'heure actuelle, l'importance des projets fait qu'il n'est plus possible de les confier uniquement à une personne mais qu'il faut la puissance de travail d'un groupe avec les compétences particulières de chaque individu. Les tâches assignées à chacun peuvent être spécifiques mais elles nécessitent à plusieurs moments du projet des phases de "synchronisation" et des phases de discussion pour avancer.

Nous-même, avons réalisé sur notre système, un outil qui permet un travail de développement coopératif entre plusieurs personnes. Il offre, dans l'environnement Smalltalk, la possibilité de récupérer en temps réel, toutes les modifications que les personnes connectées ont faites. C'est très pratique car il n'est plus nécessaire de sauver les modifications sur fichiers, de diffuser leurs noms avec une description de leur contenu pour que les autres puissent alors les charger dans leur environnement. Des mécanismes ont aussi été implémentés pour assurer une cohérence globale et éviter des erreurs de compilation.

. La consultation d'experts distants.

Elle est particulièrement utile dans le cas où les experts sont dispersés géographiquement ou ne sont pas disponibles au moment voulu par les demandeurs. Le Groupware permet ainsi un travail coopératif, sans imposer les contraintes inhérentes aux réunions de travail classiques.

Les champs d'applications qui nous intéressent particulièrement sont les suivants :

. Le domaine de l'enseignement : (télétuteurage, groupe de travail, étude de cas, résolution de problèmes...).

Le Groupware peut être utilisé dans le domaine de l'enseignement dit "ouvert" (nous entendons par le terme ouvert, les notions de durée, rythme et localisation). Cette nouvelle génération de systèmes doit pouvoir satisfaire un large choix de stratégies d'apprentissage allant du télé-enseignement à l'auto-formation utilisant des supports pédagogiques multimédia.

Ce domaine nous intéresse particulièrement car nous pensons que la connaissance n'est pas simplement une construction autonome, mais résulte également d'un processus de construction coopérative (principe d'écologie cognitive ou de socio-psychologie). Des recherches ont effectivement démontré que des apprenants qui travaillent en groupe réalisent mieux les tâches assignées que ceux qui travaillent seuls [GARI 80].

. Le domaine de la santé.

Là encore, des fonctionnalités de type Groupware semblent manquer. Par exemple, elles permettraient à un médecin, de consulter à distance un expert (spécialiste) pour préciser un diagnostic ou pour l'aider dans une intervention. Le bénéfice de ce type d'action est un gain de temps et d'argent considérable qui profite immédiatement aux patients car dans certains cas, chaque minute compte.

Nous nous proposons de faire dans cette première partie une présentation générale du C.S.C.W. (synchrone et asynchrone). Cependant, nous nous intéresserons plus particulièrement aux systèmes synchrones, en faisant référence notamment aux articles dont nous disposions au début de nos travaux fin 1988, pour en illustrer les différents concepts.

1.2 - Les différentes formes de communication

1.2.1 - Les systèmes asynchrones

Depuis de nombreuses années, l'idée que des ordinateurs puissent aider des personnes à résoudre des problèmes en groupe a été émise. Des précurseurs comme Bush [BUSH 45] présentaient en 1945 un système hypothétique appelé "MEMEX" incluant une base de données interactive. Dans les années soixante, des systèmes expérimentaux comme NLS [ENEN 68] et AUGMENT [ENGL 84] commencèrent à utiliser les ordinateurs pour supporter la collaboration d'un groupe. Ces systèmes étaient bâtis sur une architecture centralisée utilisant de multiples terminaux reliés à une machine centrale. Cela permettait d'accéder à une boîte aux lettres électronique et à un système de partage de fichiers. Le système NLS fournissait également un mode d'écran partagé pour la rédaction simultanée de documents structurés. Cependant, ces systèmes ne fonctionnaient correctement que si les terminaux étaient de même type. Une exception importante fût le système "Tymshare's Augment" (successeur commercial de NLS) qui permettait la connexion avec des terminaux différents.

Depuis, des progrès importants ont été réalisés dans ce domaine et il existe maintenant de nombreux produits commercialisés. Dans cette catégorie de systèmes asynchrones, nous trouvons les systèmes de courrier électronique, les messageries et les systèmes de conférence par ordinateurs appelés téléconférences.

a) Le courrier électronique

Le courrier électronique puise son origine dans les milieux scientifiques. Il permet aux possesseurs d'ordinateurs ou de terminaux, d'échanger des messages via des réseaux locaux, nationaux et internationaux. La définition donnée par E.M.A. (Electronic Mail Association) est la suivante : "*courrier électronique est le terme générique désignant la communication non interactive de textes, données, images ou messages vocaux entre un émetteur et un ou des destinataire(s) utilisant les liens offerts par les télécommunications*".

Au cours de la dernière décennie, le nombre de boîtes aux lettres ouvertes aux Etats-Unis a été multiplié par vingt pour atteindre le chiffre d'un million. N'étant plus l'apanage des informaticiens, nombreux sont ceux qui dans les entreprises l'utilisent quotidiennement pour des communications en interne et en externe, d'autant plus volontiers lorsque l'on sait que 75% des appels téléphoniques échouent parce que le destinataire est en réunion ou en déplacement. De plus, le contenu des messages évolue.

Limité dans ses débuts à du texte, il peut maintenant contenir des tableaux générés par tableur, des graphiques, des images, du son et bientôt de la vidéo.

Ainsi, le courrier électronique devient un excellent moyen pour préparer et augmenter la productivité d'une réunion en informant les participants au préalable. Il peut également être dans une certaine mesure, un outil d'aide pour un travail coopératif. Jean-Marie Hullot, auteur d'Interface Builder sur Next, expliquait récemment : "*Chaque soir, de Paris, j'expédiais mon travail de la journée à ceux qui étaient en Californie. Avec le décalage horaire, c'était pour eux le matin ; ils pouvaient prendre la relève sans perte de temps*".

Dans notre société où la communication est un élément essentiel afin de favoriser la dynamique groupe, le courrier électronique devient chaque jour de plus en plus employé et utile. Il offre de par sa structure, la possibilité de dialoguer d'une façon informelle et augmente ainsi les communications entre individus. L'un de ses inconvénients est lié à la surcharge des messages qui peut se produire. En effet, à cause des listes de diffusion, les utilisateurs peuvent avoir leur boîte aux lettres envahie (les américains appellent cela "Junk Mail") par des messages non pertinents.

b) la téléconférence assistée par ordinateur (T.C.A.O.)

Les systèmes de téléconférence offrent à des groupes d'utilisateurs des espaces de travail publics appelés conférences. Les participants communiquent entre eux par des envois de messages. Ces messages pourront être écrits et lus à n'importe quel moment. Jusqu'à présent, la plupart des messages sont comme dans les systèmes de courrier électronique, uniquement composés de texte mais la tendance est d'évoluer vers le multimédia. Ce qui différencie la T.C.A.O. du courrier électronique est la diffusion implicite des messages pour un groupe et pas seulement pour un individu. De plus, si une même personne fait partie de plusieurs conférences distinctes, les messages correspondant aux différents sujets ne seront pas mélangés comme dans une boîte aux lettres.

A l'origine, la T.C.A.O fût développée dans le cadre du projet Darpa, pour permettre le travail coopératif de scientifiques que l'on ne pouvait pas réunir au même moment ni au même endroit et/ou pour résoudre des situations de crise qui nécessitaient le concours d'experts dispersés géographiquement. Le "New Jersey institute of technologie" présenta l'un des premiers systèmes appelé E.I.E.S. qui permit de mettre rapidement en évidence les limites de ce type de système (présentées un peu plus loin) [HILT 86] [HITU 85].

Des expériences d'utilisation de la T.C.A.O. dans l'enseignement à distance (E.A.D.) ont été faites depuis quelques années par différentes équipes comme l'avait prédit Roxanne Hiltz [HITU 78] dès 1978. La plus significative fût celle de l'Open University avec Cosy, qui est toujours utilisée de nos jours et accessible sur le plan national. Afin de faciliter l'utilisation de ces systèmes, des métaphores ont rapidement été proposées dont la plus citée : celle de la salle de classe. Le système est comparé à un bâtiment composé de plusieurs classes qui seraient les conférences. L'utilisateur peut donc en fonction des cours où il est inscrit (c'est à dire de ses droits), participer au travail. Dans chaque classe se trouve un tuteur (le maître de conférence) qui en a la gestion. Il décide entre autres, de l'ouverture et de la fermeture de sa conférence.

Nous même, au sein du C.U.E.E.P., disposons depuis 3 ans de notre propre système de T.C.A.O. spécifiquement conçu pour l'E.A.D [VIDEa90] [VIDE 91]. Le statut des participants peut être de trois sortes : Gestionnaire, Conférencier ou Utilisateur. Le gestionnaire est chargé de l'organisation du système et de la création des conférences. En reprenant la métaphore, il est celui qui s'occupe du bâtiment et de l'aménagement des salles. Le conférencier a lui pour mission d'assurer le bon déroulement de la conférence. En outre, il est le seul à pouvoir ouvrir et fermer une session. Quant à l'utilisateur, il est comparable à l'apprenant et son rôle consiste à prendre une part active au débat par des envois de messages afin de parvenir au but fixé par le conférencier.

Une conférence peut avoir deux statuts distincts. Elle est, soit réservée à des abonnés (des utilisateurs déclarés par le maître de conférence et possédant un code d'accès), soit ouverte à tout public. Les messages échangés par les participants sont accessibles par tout le groupe. Cependant, des messages *cachés* sont possibles entre conférencier/utilisateurs et vice versa, mais interdits entre utilisateurs. Les messages comportent plusieurs champs afin d'y inscrire des informations utilisateurs et systèmes. Ces informations sont : le nom de l'émetteur, la date, l'heure, le numéro du message, le type du message et le contenu proprement dit. Avec l'expérience, le typage associé à des régles d'utilisation s'avère être un élément essentiel pour la structuration de la conférence.

Dans l'état actuel, les déficiences du système sont encore nombreuses. La première n'est pas directement liée au système, mais au fait que les conférences reposent sur un système vidéotex (écran trop petit, pas de souris et limite de l'éditeur de texte). La seconde difficulté majeure est due au manque d'outils d'analyse des messages. Comme les références arrières sont les seules possibles au moment de l'écriture, et que les messages sont numérotés séquentiellement par le système en fonction de leur date

de création, la possibilité de suivre une conversation n'est pas aisée (le numéro ne signifie pas grand chose). Il faut regarder tous les messages rendant ainsi la lecture rapidement fastidieuse.

Afin que ces systèmes soient d'une utilisation plus facile et plus conviviale, il est nécessaire de pallier les problèmes précédemment cités. Il faudrait des outils de type hypertexte qui permettraient une navigation plus aisée en offrant par exemple une simulation de la conversation. Il est également indispensable de fournir aux utilisateurs des méthodes pour retrouver la logique de la communication.

1.2.2 - Les systèmes synchrones

Les systèmes de communication synchrones regroupent l'ensemble des outils qui permettent le travail coopératif d'un groupe en temps réel. Les audio et vidéo conférences sont à classer dans cette catégorie. Contrairement à la T.C.A.O., il n'existe pas encore, sur le marché informatique, de produits de conférence temps réel assisté par ordinateur disponible couramment.

La présentation, que nous nous proposons de faire, est une synthèse d'articles concernant quelques prototypes dont nous disposions au tout début de nos travaux. Les raisons pour lesquelles nous les avons choisi, sont les suivantes : Colab de Xerox pour les aspects WYSIWIS, Emce du SRI pour les aspects multimédia et les travaux de I. Greif (Rtcal) pour les différentes architectures.

Dans la dernière partie de cette thèse, nous ferons le point sur des travaux plus récents, en particulier sur les éditeurs de textes multi-utilisateurs qui occupent une part importante des recherches dans ce domaine.

1.2.2.1 - L'audio conférence

L'audio conférence consiste en un réseau de studios publics ou privés, spécialement aménagés, pour réunir plusieurs groupes de personnes. Les groupes de personnes se rendent dans un studio et s'installent autour d'une table équipée d'un micro directionnel personnel, d'un haut parleur central, et d'un tableau de voyants permettant d'identifier l'interlocuteur distant qui parle. Il n'y a pas de contrainte et les participants peuvent parler et être entendus comme dans une réunion ordinaire. Les participants à la conférence ont aussi accès à des services complémentaires tels que le télécopieur et la tablette de téléécriture. Cette tablette permet aux participants de

dessiner des schémas sommaires et d'écrire quelques mots, de les visualiser ou de les modifier simultanément d'un studio à l'autre.

Le principal avantage de l'audio conférence est de ne demander qu'un faible apprentissage pour un participant habitué à des réunions. Elle est bien adaptée aux réunions "d'information" et de "coordination" mais se prête peu aux réunions de "négociation". Depuis ces dernières années, nous bénéficions d'une possibilité de répartition sur plusieurs studios avec une qualité de transmission supérieure (fréquence de 7k Hz sur RNIS). Les inconvénients sont cependant plus importants :

. nécessité de bien suivre la conversation, car l'identité des interlocuteurs n'est pas évidente. La prise de parole et l'ordre doivent être précisés verbalement (impossibilité comme dans une réunion classique de "lever la main") ;

. l'échange d'informations n'est pas très pratique et il est presque uniquement limité à l'aspect verbal. Cependant, il est à noter que les systèmes plus récents permettent maintenant pour deux studios (c'est du point à point) d'échanger des documents composés de textes et de graphiques. Ils sont conçus pour des illustrations visuelles et les documents peuvent être annotés (écrire, souligner, effacer..) et imprimés en cours de réunion. Cela s'effectue à partir d'une tablette d'écriture équipée d'un stylo électronique.

1.2.2.2 - La visio conférence

La visio conférence reprend les fonctionnalités de l'audio conférence, mais offre en plus la possibilité de voir la personne qui parle sur un écran de télévision. Les services proposés permettent de mettre en relation deux groupes de quatre à dix personnes. Chaque participant dispose d'un poste avec un écran qui lui permet de voir les interlocuteurs du studio distant. Dès la prise de parole, le participant est détecté par une régie automatique : son image est visualisée par tous les autres participants, tandis que lui-même garde l'image de l'intervenant précédent. On peut également utiliser le visiophone pour visualiser des documents, en les glissant à l'endroit prévu sous l'appareil. La visioconférence utilise des liaisons numériques à très haut débit. Ce type de conférence est valable pour des réunions où l'aspect facial est important comme des réunions de négociation. Cependant l'information échangée n'est pas éditable (on ne voit que l'information) et les participants sont obligés de se rendre dans des locaux spécialisés.

1.2.2.3 - Les systèmes de conférence temps réel assistés par ordinateur

Les laboratoires Bell ont développé un système appelé "TOPES" [PFPE 79] qui supportait la conception et l'analyse autour de plans de constructions dès 1978. Il avait les caractéristiques d'une téléconférence graphique et permettait à un groupe d'ingénieurs de tenir une réunion de conception tout en ayant une conversation téléphonique. Lors de la conférence annuelle en octobre 1982 de "American Society of Civil Engineers", une démonstration de téléconférence sur un système d'ébauche de dessin a été faite. Cela permettait à deux personnes travaillant sur des PC reliés par une ligne téléphonique, d'éditer conjointement le même projet de dessin en utilisant des curseurs indépendants mais visibles par les deux utilisateurs. De même, à la conférence informatique fédérale qui se déroula en septembre 1984, "Teneron Corp" a fait une démonstration du "Tango-writer Interactive Word Processor". Il autorisait un lien téléphonique entre deux PC et la communication alternative entre les données et la voix. Il permettait ainsi à chaque utilisateur d'éditer un document pendant que l'autre le regardait faire.

Les travaux plus récents (RTCAL, MBLINK, SRI, COLAB, etc) ont permis d'exposer les problèmes à résoudre pour concevoir des Groupwares. Une grande partie de ces problèmes n'est pas encore résolue. Il est à noter qu'une différence existe dans la conception de ces systèmes. Comme nous l'avons présenté dans l'introduction, elle réside dans le fait que les participants se trouvent, soit dans une même pièce, soit éloignés les uns des autres. Dans le premier cas, on parlera plus spécialement de salle de réunion informatisée et de "Group Decision Support System" (GDSS). De telles salles se composent généralement de stations de travail connectées sous réseau local, avec un ou plusieurs écrans géants, et des systèmes audio et vidéo. Le projet Colab de Xerox, que nous présenterons plus en détail, en fait partie. On peut également citer le "PlexCenter Planning" [APKO 86] de l'Université d'Arizona.

Nous allons, à partir de maintenant, nous intéresser essentiellement aux systèmes de conférences temps-réel. Dans un premier temps, nous présenterons rapidement les quelques systèmes que nous avons précédemment cités pour information. Nous détaillerons ensuite les différentes caractéristiques du Groupware temps-réel.

a) Colab de Xerox

Ce projet [STBO 87] a été lancé chez Xerox afin d'expérimenter une salle de réunion appelée Colab. Elle devait permettre l'étude d'un support informatique susceptible de résoudre des problèmes collaboratifs dans des réunions de face à face. Leur objectif à court terme était de développer de nouveaux outils intégrés, dans un système de conférence en utilisant des technologies informatiques et de communication. L'objectif du projet à long terme était d'étudier, à partir de ces prototypes, des outils informatiques spécifiques à un type de réunion afin de les rendre plus efficaces. Trois outils de réunion Colab furent développés : le BoardNoter, le CogNoter et l'ArgNoter. Le BoardNoter, qui imitait les fonctionnalités d'un tableau noir, était destiné à des réunions informelles, alors que le CogNoter et l'ArgNoter étaient destinés à des réunions plus formelles, puisqu'ils permettaient respectivement l'organisation d'idées pour planifier la présentation et l'évaluation de propositions.

BoardNoter

BoardNoter est l'outil qui imite le plus les fonctionnalités d'un tableau noir classique et son utilisation se veut aussi simple que possible. Pour dessiner on utilise la craie, pour effacer, une gomme, pour écrire, un minuscule clavier et pour designer, un pointeur. Dans sa version initiale, BoardNoter ne permettait pas cependant de copier, de déplacer et de redimensionner un objet.

CogNoter

Il s'agit d'un outil Colab utilisé pour préparer collectivement des présentations. Le résultat d'une réunion avec CogNoter est un plan d'idées précisées, avec du texte. Cet outil est, d'une certaine façon, similaire aux produits du type ThinkTank, Freestyle et NoteCards, mais il est prévu pour être utilisé simultanément par un groupe de plusieurs personnes. CogNoter décompose une réunion en trois phases distinctes que sont : le brainstorming (séance de réflexion), l'organisation et l'évaluation. Chacune de ces phases correspond a une activité bien particulière de la réunion.

Tout est fait pour encourager l'activité et la participation des personnes présentes. Pendant le Brainstorming, les participants travaillent d'une façon parallèle et leurs idées ne sont ni évaluées, ni éliminées. Effacer une idée dans cette phase pourrait avoir un effet négatif sur la productivité d'une

personne (elle n'est même pas effaçable par celui qui l'a écrite). Durant la phase d'organisation, il s'agit d'établir un ordre dans les idées qui ont été produites précédemment. Ces opérations sont accompagnées de discussions verbales pour argumenter. La dernière phase permet de déterminer la forme finale de la présentation.

ArgNoter

Cet outil Colab imite là aussi un processus de réunion classique utilisé pendant de nombreuses années par leur équipe de recherche chez Xerox. Il a été développé pour la présentation et l'évaluation de propositions. Ce type de réunion démarre lorsqu'une ou plusieurs personnes ont une proposition à faire sur quelque chose. Il peut s'agir de choses diverses telles que la conception d'un programme ou un plan de recherche. L'objectif de la réunion est alors de choisir la meilleure proposition.

Contrairement aux réunions type CogNoter qui ne demandent pas de préparation, les propositions qui sont faites dans l'ArgNoter ont bien été travaillées avant, permettant ainsi une meilleure argumentation en cas de discussion. Découvrir, comprendre et évaluer un désaccord sont généralement une partie essentielle des prises de décision dans ces réunions. L'ArgNoter organise également la réunion en trois phases: la proposition, l'argumentation et l'évaluation.

b) Les travaux d'Irène Greif et Sunil Sarin

I. Greif et S. Sarin [SAGR 85] [GRSA 86] ont travaillé sur deux prototypes de système de conférence temps-réel qui ont été développés aux laboratoires du MIT et qui illustrent deux stratégies d'implémentation possibles. Dans ces systèmes, les communications orales ont été établies avec des supports de communication différents de ceux des données (téléphone).

Prototype RTCAL (Real Time Calendar)

Ce premier prototype est une extension du produit PCAL (personnel calendrier) qui se trouvait être utilisé par beaucoup de personnes. RTCAL propose une planification des réunions en construisant un espace partagé d'informations, à partir des calendriers personnels des participants concernés. Les principales caractéristiques de RTCAL sont : des espaces partagés et privés, un alignement des informations relatives aux espaces

privés et publics, un mécanisme de vote, l'autonomie des participants, la séparation des commandes application et gestion de conférence, les rôles des utilisateurs dans la conférence et la présentation de l'état de la conférence.

Un système de bitmap partagé : MBlink

Le système MBlink était la version collaborative du protocole existant "Blink" conçu par D Reed au MIT. Il permettait d'afficher un même objet Bitmap d'une façon identique sur plusieurs stations. Il offrait en plus un suivi visuel de la souris et affichait la position sur le bitmap partagé, permettant ainsi de pointer une information dans la discussion.

c) Le projet CCWS du SRI

Ce projet [POAC 85] a été soutenu par la Marine Américaine en vue de développer un système d'informations multimédia pour supporter les activités de contrôle et de commande navales. Un ensemble de conditions à respecter fût fixé pour sa réalisation :

. Il devait permettre aux utilisateurs, l'accès, l'exécution, et l'échange d'informations représentées dans des média différents d'une façon simple.

. Les média initiaux supportés sont des données alphanumériques, des vecteurs graphiques, des images bitmap et de la parole digitalisée.

. Il devait supporter des conférences synchrones et asynchrones.

. Il devait fournir une assistance intelligente aux utilisateurs, leur permettant ainsi de concentrer un travail sur la prise de décision.

Le système SRI supporte des interactions de deux types :

. Synchrone : ces interactions sont essentielles en situation de crise et pendant les périodes de prise de décision. (Ces interactions peuvent être, soit visuelles, soit audibles, ou alors résulter de l'échange de données entre des processus informatiques).

. Asynchrone : ces interactions sont identiques aux systèmes de boîte aux lettres électroniques avec cependant des messages multimédia.

Un très gros effort a été fait dans la conception de l'interface utilisateur. Celui-ci utilise un écran couleur, un prototype de reconnaissance vocale et un digitaliseur de voix. Cet interface doit en effet, offrir de larges fonctionnalités. Les documents pouvant

être complexes et conçus dans plusieurs média, l'interface doit protéger le participant de cette complexité en lui fournissant un moyen d'agir de façon consistante sur chaque outil (ex. : les actions sur les boutons souris doivent avoir les mêmes actions sémantiques) et en lui offrant des facilités pour la conception des documents (possibilité d'accéder à des gabarits).

Ainsi, interagir avec le système, doit être aussi naturel que possible. La spécification d'objets ou d'actions doit pouvoir se faire avec l'un des moyens suivant : le pointage, le langage synthétique et naturel. Des travaux importants ont été faits pour traduire le langage de requête à la base de données, en un langage naturel permettant à des novices d'effectuer des requêtes sophistiquées. Pour les interactions asynchrones, un système de boîte aux lettres multimédia avec un interface classique est à la disposition des utilisateurs.

Différents éditeurs spécialisés (texte, graphique, vocal) sont proposés aux utilisateurs. Pour construire et structurer leurs messages, deux éditeurs multimédia appelés Prompter et Structure Editor ont été développés pour les aider dans ces tâches. Le Prompter est utilisé pour créer des documents dont la forme générale est prévisible ou pour des documents qui sont utilisés d'une façon répétitive, tandis que le Structure Editor est un éditeur visuel orienté objet qui présente le document sous forme de hiérarchie.

1.3 - Le Groupware temps réel

Pour S.J Gibbs [ELGI 91], le Groupware peut être vu comme un logiciel dont l'utilisateur serait remplacé par un groupe ; l'objectif étant d'obtenir une communication, une collaboration et une coordination qui permettent aux participants de réaliser une tâche commune.

1.3.1 - Les caractéristiques du Groupware

Ainsi que nous l'avons dit, dans le Groupware, les notions **d'environnement partagé** et de **tâche commune** sont primordiales. L'environnement partagé est un ensemble d'objets où les objets et les actions accomplies sur eux-mêmes sont visibles par tous les participants. La tâche commune correspond à l'objectif à atteindre par les utilisateurs et elle est fixée en début de séance.

La conception d'un Groupware nécessite des investigations dans différents domaines spécifiques : l'interface multi-utilisateurs, la gestion de la concurrence, les rôles des participants, les télécurseurs. En détaillant et en illustrant par des exemples chacun d'eux, nous allons essayer de présenter les problèmes auxquels sont confrontés les développeurs.

1.3.1.1 - L'interface multi-utilisateurs

La conception de l'interface utilisateur est généralement la partie du projet qui demande le plus de réflexion. En effet, elle cumule les problèmes de conception et d'ergonomie de toute interface mono-utilisateur mais aussi les problèmes liés à son utilisation par un groupe (gestion des interactions simultanées). La rétro-action (ou feedback) prend ici une place très importante. Il faut que l'interface renvoie à chaque participant une vision de l'activité du groupe sans pour autant être perturbatrice.

Une abstraction fondamentale pour ces systèmes a été définie ; il s'agit du W.Y.S.I.W.I.S. (What You See Is What I See). Celle-ci a pour la première fois été employée dans le projet Colab de Xerox. Le W.Y.S.I.W.I.S. reprend les caractéristiques d'un tableau noir dans une salle de réunion en créant l'impression que les membres du groupe interagissent avec des objets partagés par tous. Une stricte interprétation du W.Y.S.I.W.I.S. impliquerait que les écrans des participants soient absolument identiques, et que l'on puisse y voir l'activité des autres membres comme la

représentation de leur curseur ou de leur souris.

Bien que l'on reconnaisse l'importance de cette abstraction, il s'avère préférable dans la pratique d'utiliser une version "relâchée" du strict W.Y.S.I.W.I.S. [STEF 87]. Le relâchement peut s'opérer dans quatre dimensions : l'espace, le temps, le groupe et la congruence.

La dimension "espace" :

Dans un système de conférence, on trouve généralement deux types de vue : les vues publiques et les vues privées. Le relâchement de la dimension espace peut donc s'appliquer sur les vues privées qu'il n'est pas nécessaire de diffuser. Pouvoir personnaliser son environnement de travail doit être également possible (comme par exemple positionner une fenêtre à un endroit différent ou la redimensionner).

Seulement, il faut savoir que satisfaire cette exigence ne se fait pas sans poser certains problèmes. En effet, les références verbales deviennent alors compliquées voire impossibles. Ne pas afficher les curseurs de chaque participant sur les écrans conduit aussi à relâcher un peu plus cette dimension. Il s'est avéré, à partir des expérimentations faites dans Colab, que leurs mouvements et leurs clignotements devenaient rapidement fatiguants et distractifs.

La dimension temps/simultanéité :

Le strict W.Y.S.I.W.I.S. nécessite que les fenêtres publiques soient synchronisées dans l'affichage des mises à jour. Permettre un délai conduit à relâcher la contrainte de simultanéité. Cette dernière est directement liée à la granularité des mises à jour.

La déterminer est délicat car il faut tenir compte des performances du système (en particulier du réseau).

. Choisir une large granularité fera diminuer le flux d'informations entre les stations, mais provoquera une perte d'informations dans la construction des données.

. Le choix d'une petite granularité (comme diffuser par exemple chaque caractère frappé) va par contre augmenter fortement le flux d'informations à échanger mais offrira une collaboration plus grande dans le groupe.

Il paraît donc difficile de fixer une règle pour choisir la taille de la granularité dans les échanges d'informations.

En plus du compromis entre la taille et la rapidité, le type d'application et les besoins de l'utilisateur seront des éléments déterminants dans le choix. On peut par exemple citer le déplacement d'un objet. Doit on voir le mouvement ou simplement la position finale ? Développer un système avec une possibilité de granularité variable semble la solution la plus adéquate. Dans certains cas, la mise à jour est laissée au choix de l'utilisateur qui décide du moment opportun, ou faite automatiquement après un temps raisonnable.

La dimension congruence :

Le relâchement de cette dimension est liée à toutes les modifications de la représentation visuelle des vues publiques. Les exemples que nous donnons vont dans ce sens. Nous concevons qu'il puisse être laissé à l'utilisateur la possibilité d'avoir un "point de vue" différent de son collègue sur un même objet. Par exemple, représenter les valeurs d'un tableau sous forme de courbes ou sous forme d'histogrammes. Il est souvent utile de faire une distinction entre la façon de symboliser un objet sélectionné localement et celle d'un objet sélectionné à distance. Cela peut ,par exemple, se réaliser en utilisant des grisés de couleur différente.

La dimension groupe :

L'interprétation du strict W.Y.S.I.W.I.S. oblige la mise à jour des informations à l'ensemble des participants. Pouvoir choisir pour quelles personnes les mises à jour seront faites, en créant des sous-groupes au sein de la conférence, conduit à un relâchement de cette dimension. Autoriser des discussions privées entre les utilisateurs va également dans ce sens.

Comme nous venons de le voir précédemment, le W.Y.S.I.W.I.S. avec ces différents relâchements est un élément fondamental pour ce type d'application. Cependant, les concepteurs sont toujours contraints de faire, dans leur choix, un compromis entre les besoins d'un individu et ceux du groupe (la personnalisation d'un environnement pénalisera le groupe pour la désignation).

Des techniques particulières sont nécessaires pour gérer l'espace écran qui est limité et cela devient encore plus problématique avec des interfaces multi-utilisateurs où de nombreuses fenêtres peuvent être ouvertes par les utilisateurs (utilisation de

métaphores et d'icônes). La rétro-action (feedback) est très délicate à résoudre, car elle doit rendre compte de l'activité du groupe sans pour autant perturber le travail de chacun.

De plus, les changements soudains occasionnés par les autres peuvent être parfois difficiles à interpréter. En effet, avec une interface mono-utilisateur, l'individu dispose de son contexte mental pour interpréter les modifications sur son écran, ce qui n'est pas le cas ici. Une aide peut être apportée, en faisant évoluer lentement les transformations [ELGI 90] [ELGI 91] (par exemple, si du texte doit être effacé, cela se fait progressivement par un grisé de couleur) ou peut plus simplement, s'effectuer en annonçant oralement au groupe les actions que l'on souhaite entamer. Il va sans dire que cela ne se fait pas sans poser de problèmes. Si tout le monde parle ensemble, la discussion devient inaudible très rapidement.

Trouver des métaphores visuelles et les implémenter n'est pas si simple et leur exécution a pour conséquence une monopolisation du processeur qui peut être pénalisante sur les performances globales du système. Il faut également tenir compte du fait, que si les modifications se font sans interruption, il peut se produire un décalage entre le moment où elles sont effectivement faites et celui où elles sont annoncées aux autres.

1.3.1.2 - Gestion de la concurrence

Comme avec un tableau noir, la surface de travail se partage entre l'ensemble des participants présents. Dès le moment où plus d'une personne a accès à une donnée, des problèmes de concurrence se posent. Pour les résoudre, différentes méthodes, dont certaines sont utilisées depuis de nombreuses années dans les bases de données, existent. Nous allons en présenter quelques unes.

Mécanisme à verrouillage :

Cette solution consiste à poser un verrou sur les objets. Ceux-ci deviennent alors inaccessibles pour les autres utilisateurs. Cette méthode pose cependant des inconvénients. Il faut à chaque requête vérifier s'il y a le "verrou", ce qui conduit à une dégradation du temps de réponse.

D'autre part, le choix de la granularité de verrouillage des objets (paragraphe, phrase, mot ou caractère ?) est délicat à déterminer car il dépend principalement de l'application. Afin d'éviter des conflits et des temps d'attente inutiles, ce mécanisme peut être couplé à un artifice visuel

au niveau de l'interface pour signaler aux autres utilisateurs l'indisponibilité momentanée de l'objet.

Mécanisme avec un contrôleur centralisé :

Cette solution consiste à utiliser un contrôleur centralisé qui va collecter les requêtes afin d'en assurer la diffusion à l'ensemble des stations. Une condition est toutefois nécessaire ; il faut que les données soient dupliquées sur toutes les stations. Avec le contrôleur centralisé, on a alors les problèmes liés aux architectures centralisées ; en particulier, les temps de diffusion, car il ne faut pas que l'utilisateur qui émet une requête puisse en exécuter une autre avant que la première ne soit accomplie.

Mécanisme à exécution réversible :

Cette technique consiste à associer une étiquette aux requêtes émises. Cette étiquette comporte plusieurs renseignements dont la date à laquelle elle a été envoyée. Lorsqu'une requête arrive, une vérification est faite sur cette date. Si la date est postérieure à la référence (date de la dernière requête exécutée), elle est exécutée et la nouvelle date indiquée sert de référence. Dans le cas contraire, on va annuler l'effet des requêtes dont la date est supérieure à la dernière afin de les exécuter à nouveau dans le bon ordre. L'inconvénient de cette méthode est qu'elle peut être perturbatrice pour l'utilisateur qui risque de voir des transformations "inopportunes" de son interface.

Mécanisme optimiste :

Dans ce cas, on se base sur le fait que la probabilité d'accès concurrent à un objet est faible. Aucun mécanisme particulier n'est implémenté ce qui revient à exécuter les messages dans l'ordre d'arrivée. Si il y a conflit, il est résolu par une négociation entre les différentes parties concernées.

Le choix de l'un de ces mécanismes aura une incidence importante dans la conception du gestionnaire de conférence.

1.3.1.3 - Rôle des participants

Dans un Groupware, le rôle des participants ainsi que leurs droits d'accès aux objets peuvent être différents les uns des autres.

En régle générale, un maître de conférence est généralement désigné dans chaque session afin de garantir le bon déroulement de la séance. Son rôle consiste, dans un premier temps, à ouvrir la conférence afin de permettre la connexion des utilisateurs. C'est également lui qui sera chargé de la fermer lorsque la séance de travail sera terminée. Ses droits et ses possibilités d'intervention sont souvent supérieurs aux autres, pour permettre le déblocage de certaines situations conflictuelles. Comme dans une réunion classique (c'est à dire sans ordinateur), il doit aussi intervenir et conduire la réunion de façon à dynamiser le groupe.

Le rôle des participants est très fortement associé à celui du gestionnaire de conférence qui défini suivant un protocole établi, le droit d'accès aux objets. Cet accès et plus spécialement les droits du participant lorsqu'il a ou non le contrôle sur un objet, est géré par "l'algorithme de contrôle" de la conférence.

En effet, une seule personne, à un instant donné, peut modifier un objet. L'objet peut être l'ensemble de l'espace public ou une partie le constituant (fenêtre, sous-fenêtre, paragraphe etc..). Au cas où le contrôle est accordé sur seulement une partie, l'impression de parallélisme du point de vue utilisateur pourra être perçu au niveau du Groupware.

Assurer une gestion correcte sur un nombre important d'objets dans l'espace public n'est pas très compliqué d'un point de vue informatique mais l'est pour les utilisateurs qui doivent pouvoir demander/rendre ce contrôle et savoir à tout instant à qui il a été attribué. Il faut donc avoir conscience que la complexité va croissante avec une diminution de la granularité et qu'il faut alors adapter l'interface à ce besoin de représentation des objets avec les personnes qui ont le droit momentanné de les modifier.

Il n'existe pas de règles bien définies pour faire ces spécifications, mais il faut les adapter aux besoins de l'application coopérative et plus particulièrement aux types d'activités qu'elle génère. Il est certain que plus le nombre d'objets de l'espace public augmentera, plus les utilisateurs pourront intervenir en même temps, ce qui conduira à avoir une activité du groupe plus intense car moins entrecoupée de "temps morts".

Cependant, en nous référant à l'application que nous avons développée, il ne faut

pas systématiquement chercher à tout "paralléliser".

Dans l'exemple que nous présenterons ultérieurement, l'une des phases de la conférence consiste à établir un plan de production. Ce plan est un tableau qui au départ est vide et qui doit être rempli pour enchaîner sur les phases suivantes. Dans ce tableau, toutes les cases impliquent les participants, c'est à dire que les valeurs inscrites dans ces cases ont une conséquence directe pour chacun d'eux.

Pour entrer une valeur, il faut donc qu'ils soient tous d'accord. L'objet partagé que nous avons considéré dans cette première phase, est le tableau en entier. Ce qui revient à dire, qu'une seule personne, à un instant donné, aura la possibilité de le modifier, et que les autres ne pourront que faire des propositions qui seront évaluées par l'utilisateur autorisé. Nous aurions pu attribuer le contrôle à des utilisateurs différents sur les cases, mais cela n'aurait eu aucun sens et cela aurait rapidement conduit à une pagaille complète. Nous sommes ici dans un cas typique qui nécessite une activité focalisée de la part du groupe entier.

1.3.1.4 - Le télépointage

Il est généralement nécessaire lors d'une conversation, de faire référence à certaines parties de l'écran. Le télépointeur est un outil des plus importants, car il offre cette désignation distante qui permet de compléter les références verbales et il permet, lorsqu'il est actif, de revenir à une forme plus stricte de W.Y.S.I.W.I.S. (En particulier la dimension espace).

Les télépointeurs peuvent se présenter sous plusieurs formes. Ils peuvent avoir l'aspect d'un curseur classique, différencié de celui attaché à la souris locale par une forme ou une couleur différente. De plus, un télécurseur peut ou non laisser une trace à l'écran. Un télépointeur au sens large, peut être aussi un moyen de désigner une zone d'une fenêtre et non pas seulement un point de l'écran en la plaçant par exemple, en vidéo inverse.

Avec les différentes relâches du W.Y.S.I.W.I.S., se posent des difficultés pour faire des références distantes. En effet, si les fenêtres publiques ont la possibilité d'être "scrollées" indépendamment les unes des autres, il faudra les réaligner au début du télépointage. Il est évident que cela ne se fait pas sans poser de problèmes, car si l'utilisateur est en train de lire ou de copier une partie de texte et qu'on lui modifie soudainement sa vue, son travail en sera perturbé. Plusieurs solutions sont envisageables, comme par exemple, l'ouverture d'une copie de la fenêtre le temps de la

télésélection.

De même, autoriser des tailles et des localisations des fenêtres différentes, nécessitera le calcul des nouvelles coordonnées du télépointeur sur chaque station. Si la gestion de l'espace écran autorise une mise sous forme d'icône des fenêtres, leur réouverture devra également se faire automatiquement. Dans ce cas, il faudra tenir compte de la surcharge visuelle occasionnée pour l'utilisateur et évaluer dans quelle mesure et jusqu'à quel point, elle sera acceptable. Les fenêtres textes sont particulièrement difficiles à résoudre lorsque leur taille peut être modifiée. En effet, le texte étant dans la majorité des cas, rejustifié automatiquement en fonction de la largeur, un simple calcul d'homothétie ne suffit plus. Il faut pouvoir accéder à chaque caractère et à sa position. Cela est d'autant plus vrai, lorsque la relâche permet l'édition du texte dans des polices de caractères différentes. Suivant les environnements, il faudra généralement accéder à des fonctions que l'on appelle de "bas niveaux" qui sont délicates à implémenter.

Nous nous apercevons à travers ces exemples que le problème du télépointeur n'est pas simple à résoudre car sa complexité augmente au fur et à mesure que les libertés permises aux utilisateurs dans la gestion de l'écran, augmentent. Une fois encore, il s'agit de faire un compromis entre la liberté de la gestion de son espace de travail et la complexité voire l'impossibilité du télépointage.

1.3.1.5 - Propriété des données

La sauvegarde et la récupération de données posent également des problèmes. En effet, pour qu'un système de conférence puisse fonctionner correctement, il faut pouvoir accéder à des informations externes qui ont une existence indépendante. De même, une séance de travail engendrera des informations qui pourront être utilisées par d'autres utilisateurs. Le transfert de ces données à l'intérieur ou à l'extérieur d'une conférence soulève les problèmes du droit d'accès à l'information.

> Pour la récupération des données, suffit-il qu'un seul participant détienne les droits d'accès pour pouvoir les ramener dans la conférence ? Lorsque des données sont sauvegardées sur fichiers, à qui appartiennent-elles ?

1.3.2 - Architectures physique et logicielle de la conférence temps réel

1.3.2.1 - Architecture physique

L'importance et la qualité des fonctionnalités, dont on souhaite disposer dans une conférence temps réel, dépendra des aspects logiciels et matériels que l'on intégrera au système. L'aspect communication étant très important, le type et la bande passante du réseau influenceront fortement les performances du système.

Pour un travail agréable, les utilisateurs doivent pouvoir accéder à une station de travail performante avec un écran haute-résolution, un clavier, une souris, un micro, des écouteurs et éventuellement une caméra et un moniteur vidéo ou une carte d'incrustation, si le réseau le permet.

Il existe différentes architectures qui seront contraintes par le type du réseau de transport retenu : réseaux locaux à courte distance (permet une répartition maximale), réseaux publics à longues distances (répartition plus difficile).

1.3.2.2 - Architecture logicielle

Trois types différents d'architecture logicielle peuvent servir de modèle pour l'implémentation d'un système de conférence temps-réel.

a) Trois types d'architecture

Ces trois architectures sont : distribuée, centralisée et hybride. Une architecture hybride est une combinaison des deux précédentes [LANT 86] [CHCH 91].

Les systèmes distribués.

Ce qui caractérise un système informatique réellement distribué est une absence de mémoire commune, l'utilisation d'un système de communication, et l'impossibilité d'avoir à un instant donné un état global qui soit perceptible par un utilisateur [RAYN 85]. Deux choses peuvent être distribuées dans un système :

. Les données ; une distinction peut être faite entre une duplication, où l'on retrouve une copie des données sur chaque site, et le partitionnement, qui fixe une partie des données sur chacun des sites, tout en offrant un accès

depuis chacun d'eux.

. Le contrôle ; chaque site à la gestion locale de tâches particuliéres. Des échanges de messages ont lieu pour assurer la cohérence globale mais il n'y a pas de processus de supervision.

Dans un système distribué, il est possible d'introduire un degré variable de répartition qui permette de répartir plus ou moins le système.

La figure 1.4 illustre l'architecture d'un système de conférence complètement dupliqué.

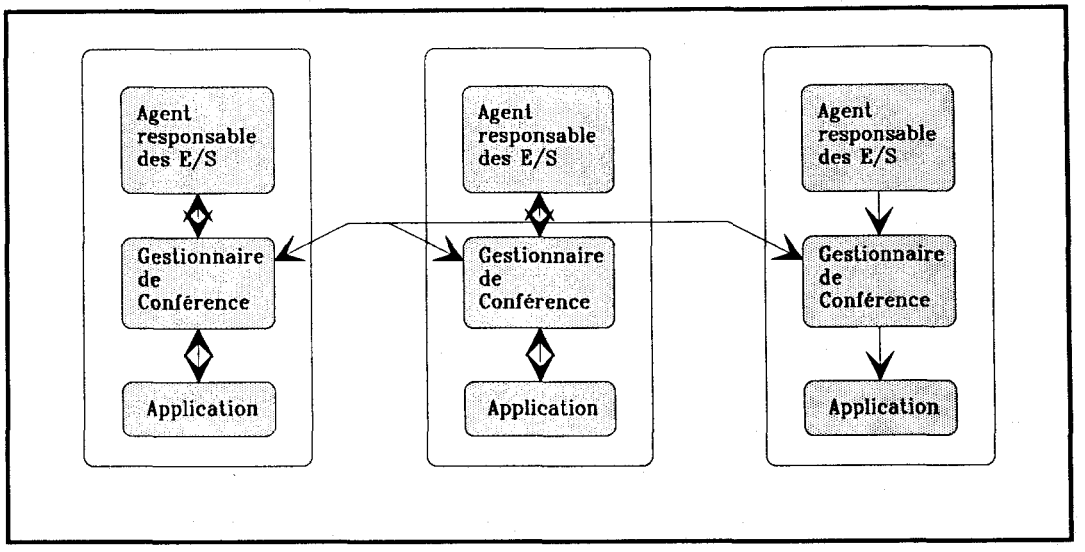


Figure 1.4. *Architecture complètement dupliquée. Nous retrouvons tous les composants du système sur chacune des stations.*

Choisir une architecture distribuée doit se faire en ayant conscience des problèmes que son implémentation pose. En effet, ses inconvénients sont :

. Assurer l'intégrité des données pour conserver la cohérence augmente la complexité des algorithmes (problèmes de terminaison et d'interblocage).

. Les problèmes de synchronisation et de démarrage sont délicats à résoudre.

. En cas de coupure des communications, la reprise est particulièrement difficile.

. Lors d'appels à certaines fonctions propres à l'environnement système, des problèmes de cohérence peuvent apparaître (avec

des fichiers ou des fonctions aléatoires).

. La plateforme matérielle doit être plus lourde car chaque station doit avoir la puissance nécessaire pour supporter l'application ainsi que les fonctionnalités dues au système de conférence.

Une architecture distribuée a heureusement des avantages capitaux pour certaines applications.

. Elle permet de fonctionner dans un mode dégradé en cas de panne d'une des machines ou lors de coupures de communication dans le réseau. C'est pour cette raison que ce type d'architecture est souvent choisi dans des applications militaires.

. Elle permet, lorsque les traitements sont lourds, de diminuer fortement les temps de réponse.

Les systèmes centralisés

Dans un système complétement centralisé, tous les sites partagent le même environnement. Les sites peuvent alors être assimilés à des terminaux même s'il s'agit de stations de travail. Toutes les actions sont envoyées sur un serveur qui se charge du traitement et qui renvoie sur les stations un résultat. Ici encore, il est possible d'introduire différents degrés de centralisation. Le plus haut correspond a un mode terminal qui ne fait aucune interprétation en local et qui renvoie sur le serveur chaque touche frappée. En diminuant ce degré, on peut commencer à échanger des messages dits "de plus haut niveau" qui permettent de diminuer le nombre de messages échangé, en utilisant par exemple des terminaux de type X-Windows. Cependant, nous tombons rapidement dans une architecture hybride.

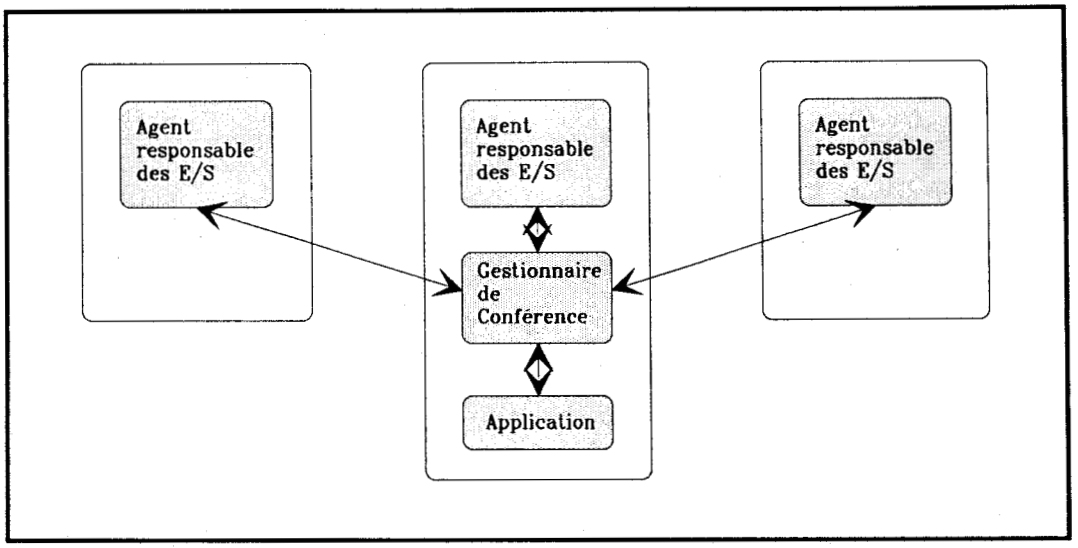


Figure 1.5. *Architecture complètement centralisée. Sur chaque station, on ne trouve qu'un agent responsable des entrées/sorties. Il renvoie sur la station centrale toutes les interactions souris/clavier et recoie de cette dernière les informations qui lui permettent de mettre à jour l'interface visuelle.*

Les inconvénients d'une architecture complètement centralisée sont :

. Le trafic sur le réseau est très important. Cela peut être envisageable sur des réseaux locaux qui ont des bandes passantes très importantes, mais cela ne l'est pas sur des réseaux publics.

. Temps de réponse plus importants.

. En cas de problème dans les communications, la station est inutilisable.

Les avantages sont :

. La plate-forme matérielle est plus souple. C'est à dire que les stations clientes peuvent être de puissance plus faible.

. Le développement des applications est beaucoup plus simple que dans une architecture dupliquée.

. Les problèmes de cohérence et de consistance des données ne se posent pas.

Architecture hybride

Une architecture hybride est une combinaison des deux architectures précédentes.

Comme nous l'avons vu, les systèmes de conférence temps-réel sont constitués d'une partie applicative et d'une partie de gestion de conférence. L'une et l'autre pourront alors être dupliquées ou centralisées en fonction des besoins.

En effet, dans un système de conférence il faut absolument diminuer les temps de réponse pour que la rétro-action de l'activité des différents participants présents soit la plus efficace possible. D'autre part, il est nécessaire d'assurer à tout moment une cohérence globale du système.

Pour ces différentes raisons, il apparaît qu'une architecture hybride soit particulièrement adaptée en faisant un compromis entre une répartition et une duplication. Il s'agit de centraliser ce qui est nécessaire pour assurer une cohérence globale et de dupliquer tout ce qui ne nécessite pas de coordination centrale. Cela permet également d'échanger des messages de plus haut niveau en réalisant localement une partie du traitement.

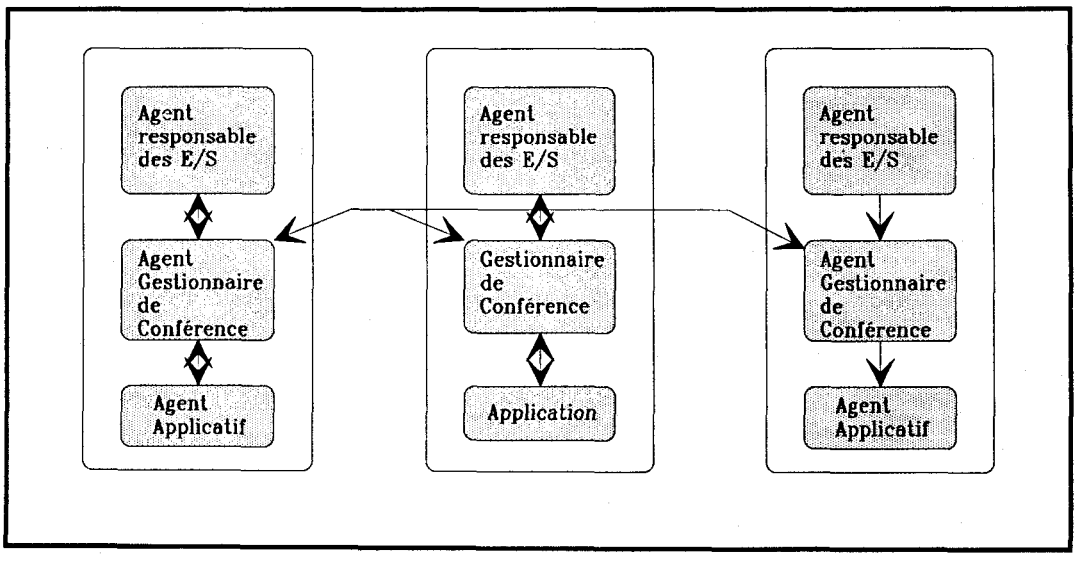


Figure 1.6. *Exemple d'une architecture hybride sur un système de conférence. Sur chacune des stations on trouve un agent responsable des entrées/sorties, un agent applicatif et un agent du gestionnaire de conférence. Il sont en mesure d'effectuer en local certaines tâches mais la cohérence globale est assurer par la station centrale.*

b) Deux tendances : coopératisation d'une application mono-utilisateur ou conception d'une application spécifique multi-utilisateurs

Lorsque l'on étudie les architectures des systèmes de conférence temps-réel, on s'aperçoit que l'on trouve deux tendances différentes. La première consiste à prendre une application mono-utilisateur existante et à la "coopératiser" ou "collectiviser", tandis que la seconde conçoit spécifiquement une application de groupe.

Il est à noter que certains systèmes de téléconférence sont spécialement conçus pour une application, alors que d'autres sont un cadre d'accueil pour différentes applications coopératives.

Coopératisation d'une application mono-utilisateur existante

L'idée de départ est la suivante : En rendant coopérative une application mono-utilisateur qui est familière à des personnes, seules les nouvelles commandes propres à la gestion de la conférence seront à apprendre.

La coopératisation d'une application nécessite cependant certaines conditions. La principale suppose qu'une distinction ait été faite entre l'interface utilisateur et les algorithmes propres à l'application. D'autre part, la nature des problèmes d'une "collectivisation" (ou coopératisation) est liée au fait que les applications mono-utilisateur sont principalement séquentielles et non distribuées, alors qu'une conférence est par nature distribuée, et que la concurrence y est présente.

L'une des techniques utilisées, dans une perspective centralisée, consiste à adopter l'approche dite du "Terminal Virtuel". Au lieu d'interagir directement avec un terminal utilisateur, le programme interagit avec un noeud de contrôle par l'intermédiaire d'un terminal virtuel. Ce noeud de contrôle, qui peut être un processus, va lire les sorties du programme et renvoyer l'entrée du programme. Le noeud de contrôle communique avec les stations de travail des utilisateurs au moyen d'un multiplexage des entrées et des sorties du programme de façon à réaliser les fonctionnalités désirées au niveau de la conférence. Les sorties du programme sont envoyées par le contrôleur à chaque station. Le contrôleur peut d'une façon sélective renvoyer les entrées des stations au programme.

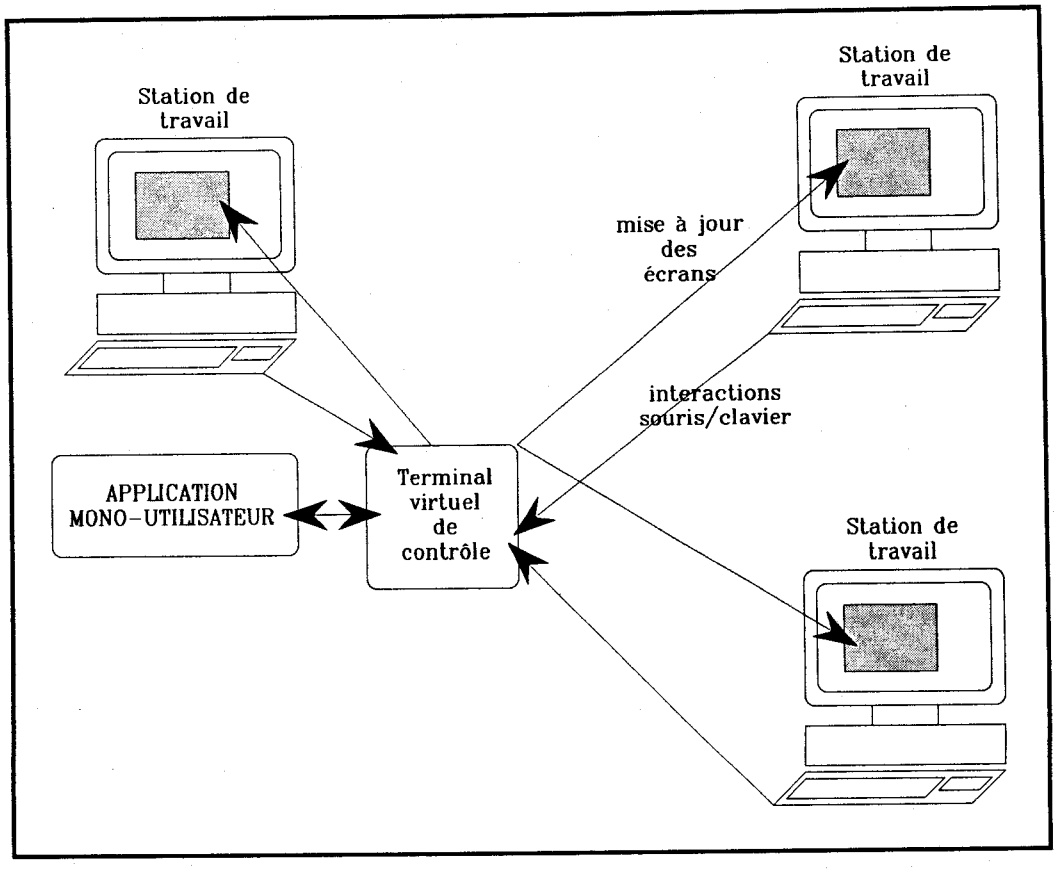


Figure 1.7. *Approche du terminal virtuel.*

Cette approche du terminal virtuel possède plusieurs avantages :

. Les programmes existants ne demandent quasiment aucune modification. La réalisation du contrôleur n'est pas très difficile et il peut être utilisé pour d'autres applications.

. Il est possible de partager plusieurs programmes sur des systèmes multi-fenêtres.

Cependant, cette approche souffre de certaines limites :

. Des contextes concurrents multiples dans le même espace partagé ne peuvent être supportés (comme l'édition des curseurs).

. Des vues privées de l'information partagée ne sont pas possibles (des fenêtres privées dans des programmes d'application séparés peuvent être supportées, mais il ne s'agit pas de la même chose).

. Le transfert d'informations entre les fenêtres privées et partagées ne peut être fait qu'à un très bas niveau.

. Les participants ne peuvent pas avoir des privilèges d'accès différents.

L'ensemble des opérations exécutées par le programme partagé sera fondé sur les privilèges d'un utilisateur spécifique. Pour l'accès à certains fichiers, comme les droits peuvent être différents pour les participants présents, des problèmes de confidentialité à l'information risquent de se poser.

Une variante de cette approche du "terminal virtuel", qui a des caractéristiques légèrement différentes mais qui partage beaucoup de limitates exposées précédemment, est le partage de terminaux d'entrées plutôt que de sorties (Il s'agit alors d'une architecture dupliquée). Sur chacune des stations des participants, tourne une instance identique du programme d'application et lorsqu'une entrée est faite par participant qui en a les droits, elle est diffusée à l'ensemble, afin que son exécution ait lieu localement. L'envoi des entrées plutôt que de la sortie, généré, par une instance unique du programme, peut faire diminuer la bande passante. Cette approche qu'a utilisée le Brown Institut pour le système Balsa est pratique lorsque les "clients" connectés sont des stations avec un processeur, mais devient lente s'il de simples terminaux.

Il semble qu'une architecture hybride soit la solution la plus satisfaisante. Il s'agit de déterminer ce qui peut être géré de façon locale, de ce qui doit être centralisé. L'inconvénient est que des modifications importantes doivent alors être apportées au programme et que l'on tombe rapidement dans le second cas de figure qui est le développement spécifique d'une application.

Il existe sur Macintosh un produit appelé "Timbuktu" [TIMB 90] qui permet de travailler en groupe sur n'importe quelle application (implémentée sur Macintosh). Timbuktu est un logiciel de réseau (sous le réseau Appletalk) qui offre, plus précisément, la possibilité de partager des écrans et des fichiers entre plusieurs machines.

Lors du lancement du programme (Timbuktu), une fenêtre apparaît sur l'écran de l'utilisateur qui peut alors choisir s'il accepte ou non de partager son écran.

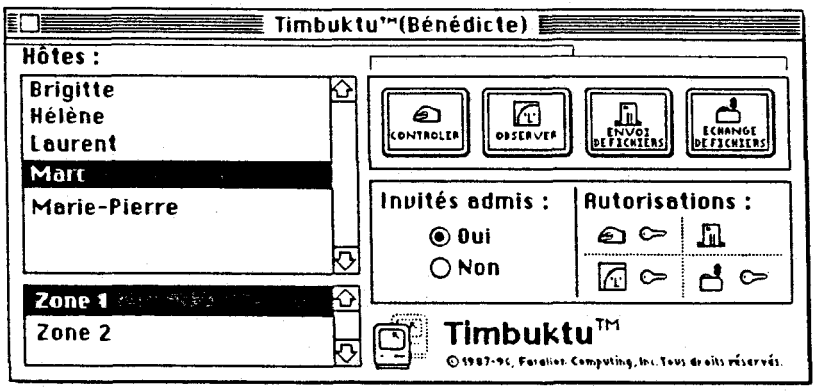


Figure 1.8. *Fenêtre Timbuktu. Dans la partie gauche apparaissent les noms des utilisateurs connectés qui ont lancé le programme. S'ils ont choisi d'avoir des invités, leur nom apparaît en noir et dans le cas contraire, en gris.*

Dans l'affirmative, il pourra devenir ce que l'on appelle un "hôte" pour les autres utilisateurs. Tous les utilisateurs sur le réseau qui auront lancé leur programme Timbuktu pourront eux aussi choisir d'être des "hôtes" potentiels en acceptant des "invités".

Un utilisateur qui souhaite voir "apparaître" l'écran d'un hôte sur sa station, réalise cela, en sélectionnant l'une des deux options situées sur la partie droite (haut) de la fenêtre Timbuktu. Il peut choisir d'être, soit un "observateur", soit un "contrôleur". Sur ces deux options, des mots de passe peuvent avoir été placés par les hôtes qui sélectionneront de cette manière les utilisateurs avec qui ils souhaitent travailler. En effet, il n'est pas possible lorsque l'on choisit d'accepter des invités de spécifier lesquels.

Lorsque l'utilisateur a choisi l'une de ces deux options, une fenêtre apparaît alors sur son écran. Il s'agit d'une copie de l'écran de l'hôte.

. S'il est un "observateur", il ne peut que regarder le travail de l'hôte sans pouvoir agir.

. S'il est "contrôleur", il peut effectuer toutes les opérations accessibles et permises par l'hôte.

Si l'on veut effectuer un travail coopératif, il faut au minimun un hôte et que les autres utilisateurs soient dotés du statut de "contrôleur" ou éventuellement "d'observateur".

La difficulté avec Timbuktu réside dans le fait que la gestion de l'espace public entre les utilisateurs n'est pas assurée. Chacun peut à tout instant, utiliser sa souris et son clavier pour modifier l'espace. On ne sait pas exactement quelles interactions souris/clavier sont prises en compte. Lorsqu'un contrôleur à la "main" (c'est à dire que ses interactions sont prises en compte), il va la garder tant qu'il frappe des caractères ou qu'il bouge sa souris. A partir du moment ou il s'arrête, ce sera à l'un des utilisateurs qui interagit de reprendre la "main".

Les déficiences de Timbuktu sont nombreuses.

. Il n'existe pas de contrôle et de gestion de l'espace public. Ce sont les utilisateurs qui doivent eux-même l'assurer en adoptant la discipline de leur choix.

. Il n'est pas possible pour l'hôte de se réserver un espace privé inaccessible aux autres utilisateurs (contrôleurs).

. Les utilisateurs ne peuvent pas récupérer par des opérations classique de copier/coller, des données de l'espace public (l'écran de l'hôte) pour les ramener dans leur espace privé. En fait, la communication entre les deux espaces n'est possible que par l'intermédiaire d'un transfert de fichiers.

. Les utilisateurs ne connaissent pas à chaque instant quel utilisateur est en train de modifier l'espace public.

Nous avons fait une expérience [CLVI 91] sur notre réseau de Macintosh (cinq machines), pour mettre en évidence les lacunes de Timbuktu. Il s'agissait de travailler et d'élaborer la solution d'un problème à partir du logiciel Hypercard. Les conclusions de cette expérience corroborent les déficiences de Timbuktu que nous avons précédemment exposées, notamment, l'absence totale de gestion de l'espace public par le système. Cela a cependant pu être compensé en partie par une négociation verbale entre les utilisateurs.

Timbuktu n'est pas dans son état actuel, un système qui permette, de façon efficace, un travail réellement coopératif. Il s'agit simplement d'une "redirection" des entrées et des sorties (multiplexage et démultiplexage). Lorsqu'un utilisateur (contrôleur) interagit dans la fenêtre partagée, ses commandes sont envoyées sur la station "hôte" (via Timbuktu). Elles vont être exécutées et des données vont alors être renvoyées aux programmes Timbuktu résidant sur les différentes stations pour permettre l'affichage et la mise à jour de la fenêtre partagée.

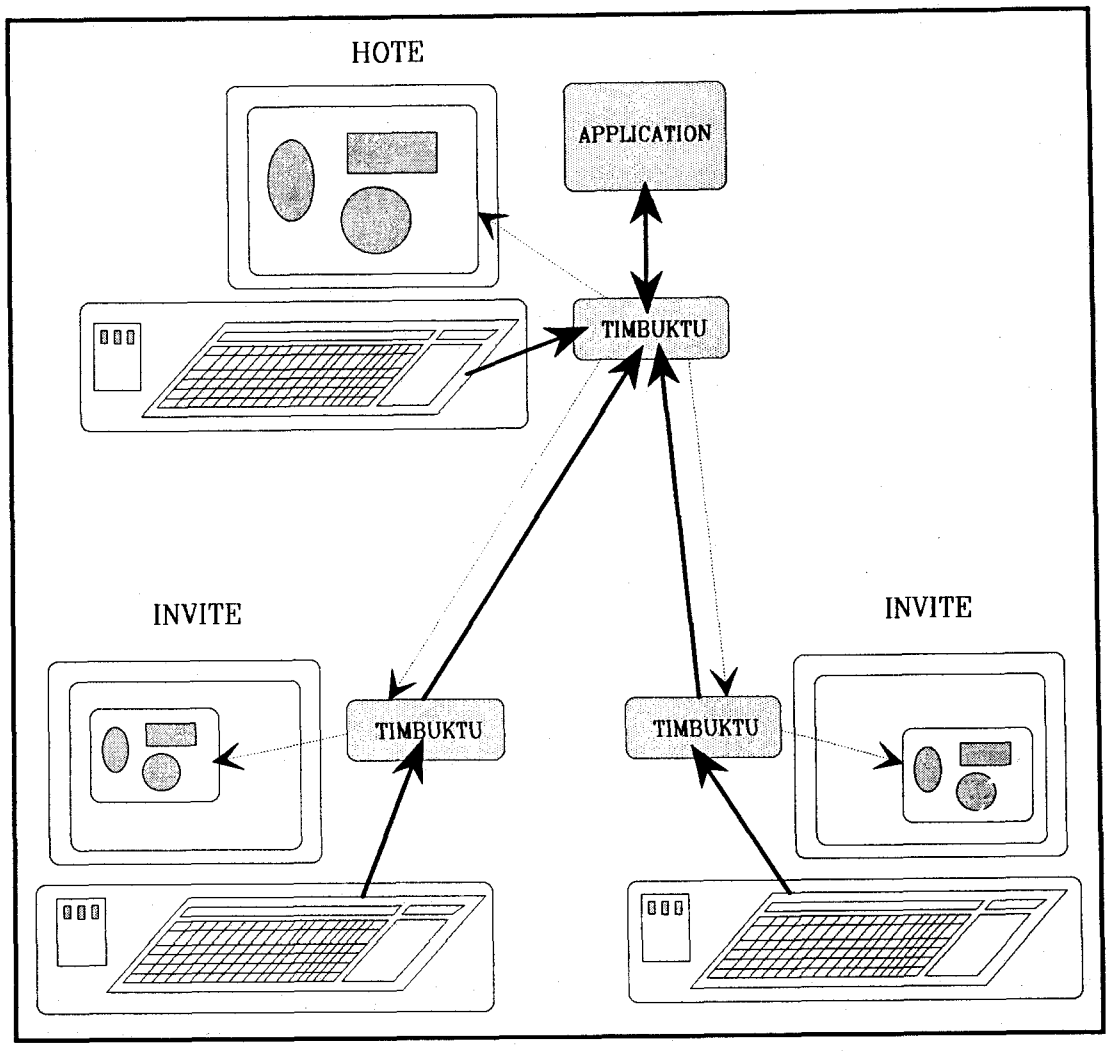


Figure 1.9. *Schématisation du fonctionnement de Timbuktu. On remarquera que l'espace public (fenêtre partagée) sur les stations des invités correspond à l'écran de la station hôte.*

En conclusion, il ressort que ces différentes techniques de coopératisation d'applications mono-utilisateur sont inadaptées au Groupware. En effet, une application de groupe est beaucoup plus complexe dans son utilisation et dans les activités qu'elle génère. Vouloir régler le problème en opérant uniquement un multiplexage des entrées et (ou) des sorties, ne conduit pas à un résultat satisfaisant. Une application multi-utilisateurs a des contraintes beaucoup plus importantes qui doivent se traduire par des fonctionnalités applicatives et une interface plus adaptées à la manipulation de groupe.

Conception d'une application multi-utilisateurs spécifique

Pour concevoir une application coopérative spécifique, les concepteurs devront trouver des solutions à l'ensemble des problèmes induits par les différents concepts (espace partagé, rôle des participants, W.Y.S.I.W.I.S., gestionnaire de conférence) que

nous avons précédemment définis. Il faut prendre en compte, l'ensemble des facteurs humains spécifiques à la coopération.

En effet, pour I. GREIF [GRSA 86] [SAGR 85], ce qui distingue une application multi-utilisateurs d'une application mono-utilisateur, est le fait qu'il faille prendre en compte l'identité des différents participants dans la conférence et qu'il faille assurer la rétro-action des activités de chacun à l'ensemble du groupe (il semble que la vidéo soit un moyen efficace pour régler une partie du problème). Effectivement, les participants à ces réunions informatisées ont besoin de savoir ce que font les autres. Par exemple, dans CogNoter, après quelques minutes d'activités parallèles dans la phase de brainstorming, les participants avaient envie de faire le point sur l'état de la conférence pour pouvoir situer le travail respectif de chacun.

Ainsi une alternance d'activités sérielles et parallèles, qui est nécessaire pour avancer vers l'objectif, suppose des négociations verbales (il faut laisser le temps aux participants de terminer les travaux en cours).

Indépendamment du modèle d'architecture choisi, une application multi-utilisateurs doit offrir un accès à des informations d'un haut niveau qui permettront aux participants de gérer l'espace public comme ils le souhaitent (Il ne s'agit pas de diffuser par exemple, des copies d'écrans sous forme de "bitmap"). Le choix de l'architecture doit être fait avec le souci constant de minimiser les temps de réponse dans la mise à jour des stations de tous les participants. Choisir entre un modèle centralisé, dupliqué ou hybride dépendra fortement de la plate-forme logicielle et matérielle choisie.

L'interface utilisateur qui est l'élément fondamental du système doit être particulièrement soignée par le concepteur d'un Groupware. Il dispose de l'abstraction du W.Y.S.I.W.I.S. avec ses "relâches", qui est actuellement l'élément essentiel pour ces systèmes afin de la concevoir.

Il doit également régler les conflits et faire un compromis entre les besoins de l'individu et ceux du groupe. Un exemple de ces problèmes rencontrés dans le projet Colab avec le Cognoter et l'Argnoter était dans la gestion des fenêtres à l'écran. Lorsqu'un individu fermait une fenêtre en raison de la surcharge visuelle à l'écran, elle était automatiquement réouverte lorsque quelqu'un d'autre y travaillait, obscurcissant ainsi de nouveau son espace de travail. (Résolution possible en utilisant des icônes qui signalent une activité en clignotant, mais qui ne réouvrent pas la fenêtre). Les fenêtres pouvant être placées à des endroits différents sur les écrans de chacun, elles devraient comporter un nom pour que l'on puisse y faire des références verbales.

D'autre part, le "gestionnaire de conférence" doit être adapté à l'application et plus particulièrement aux types d'activités qui doivent se dérouler lors de la réunion. Choisir de développer un système de téléconférence comme une architecture d'accueil pour différentes applications offre l'avantage d'implémenter une fois pour toute l'ensemble des commandes et fonctions propres à la gestion de conférence. Il est généralement nécessaire de fournir l'environnement de développement adéquat des applications (en offrant des boîtes à outils au programmeur et des outils de débuging). Il est évident que cette démarche devient rentable si de nombreuses applications sont implémentées sur le système.

En définitive, il est très important de bien comprendre le processus de déroulement d'une réunion, le comportement et les attentes des participants, avant d'entreprendre quoi que ce soit. L'un des objectifs prioritaires de ces nouveaux outils étant d'y maintenir une collaboration la plus intense possible.

1.4 - Présentation de notre système de téléconférence temps réel

Avant d'exposer les choix d'architecture logicielle de notre prototype, il nous semble préférable de présenter les fonctionnalités générales du système ainsi que son interface utilisateur.

1.4.1. Introduction

Contrairement à des projets correctement définis qui conduisent à la rédaction d'un cahier des charges précis, nous ne disposions pas au départ de nombreux éléments pour prendre des décisions de conception. L'un des aspects qui nous semblait cependant très important, était d'opter pour un système qui soit non pas dédié à une seule application, mais à un ensemble d'applications ; c'est à dire parvenir à une architecture d'accueil pour diverses applications coopératives qui nous permettrait en les expérimentant, d'améliorer le système. Un des concepts clés de ce prototype, est qu'il est ouvert (évolutif).

Concevoir ce système comme une architecture d'accueil est plus délicat à résoudre dans un premier temps mais permet par la suite de bénéficier d'un ensemble de fonctionnalités qu'il n'est plus nécessaire de réécrire. Ainsi, nous avons distingué les fonctionnalités utiles au déroulement de la conférence, de celles propres aux applications. Notre idée était de fournir aussi des facilités aux programmeurs afin qu'ils puissent implémenter des applications coopératives sans avoir besoin de connaître le fonctionnement interne du gestionnaire de conférence.

Dans un système de téléconférence, l'un des éléments les plus problématiques est la définition du protocole d'utilisation et de partage de l'espace public, qui conduit à l'implémentation de ce que l'on appelle "l'algorithme de contrôle". A postériori, nous pouvons dire qu'il n'en existe pas un seul, mais plusieurs, qui doivent être adaptés aux types d'activités induites par l'application. Celui que nous avons implémenté a été déduit d'un processus classique utilisé dans la majorité des réunions de face à face.

En général, chacun des participants a un rôle particulier à tenir et l'un deux doit assurer celui d'animateur (ou encore responsable de la réunion). La mission de ce dernier est de conduire la réunion afin de faire progresser le groupe vers son but et pour cela, il doit en principe, respecter les trois étapes suivantes : présenter l'objet et le sujet de la réunion, lancer et guider la discussion puis contrôler et exploiter les conclusions.

Pour qu'une discussion se déroule normalement, il faut qu'une certaine discipline

régne et soit adoptée par l'ensemble des participants. Cela implique plus particulièrement qu'une seule personne à la fois parle ou utilise un quelconque support (tableau, rétroprojecteur, etc..). En effet, nous conservons souvent une impression négative lorsque nous avons participé à une réunion ou suivi un débat dans lesquels les participants ont parlé en même temps.

La prise de la parole ou du support par les participants peut se faire par une demande explicite ou d'une façon tout à fait implicite. Demander explicitement la parole peut, par exemple, se faire par un signe de la main à l'animateur. Celui-ci l'accordera alors, en fonction de l'ordre des requêtes ou suivant un ordre qu'il jugera bénéfique pour le déroulement de la réunion.

Cependant, dans la majorité des cas, il n'y a pas de protocole aussi rigide et la parole est prise implicitement. C'est à dire qu'un ensemble d'indices comportementaux va permettre aux individus de déterminer le moment opportun pour prendre la parole sans nuire au locuteur présent. Ces indices sont de toutes natures comme par exemple, une baisse d'intensité de la parole, un changement de ton, un temps "mort" ou une déduction faite par rapport au contenu des phrases.

Ne pouvant pas dans un premier temps implémenter un algorithme permettant un tel degré de subtilité, les choix que nous avons faits pour définir notre protocole de gestion de l'espace public ont été les suivants :

. Dans chaque conférence, on trouve un maître de conférence (animateur) qui dispose de droits (ou de possibilités d'intervention) supérieurs a ceux des autres. Ainsi, il pourra notamment, reprendre le contrôle de l'espace privé à tout instant et l'attribuer à la personne qu'il souhaite, même si celle-ci n'en a pas fait la demande.

. Les droits des autres utilisateurs sont identiques. C'est à dire qu'ils n'ont pas de priorité supérieure les uns par rapport aux autres dans la demande du contrôle. L'attribution de ce dernier se fera dans l'ordre des demandes.

. Il n'y a qu'une personne qui a le contrôle global de l'espace public à un instant donné.

. Nous avons également considéré que le contrôle serait rendu par l'utilisateur au moment où il le souhaiterait (relachement explicite). Cependant, les autres ont toujours la possibilité d'insister et de lui demander verbalement de le rendre. Là encore, de nombreux conflits peuvent être réglés oralement pour permettre un gain de temps.

1.4.2. Contraintes de l'interface

Nous avons, dans un premier temps fixé les différentes informations que nous souhaitions visualiser sur l'interface du gestionnaire de conférence. La liste de ces paramètres était la suivante :

. Le nom de tous les participants présents.

. Le nom de la personne qui contrôle l'espace public.

. Le nom du prochain utilisateur qui aura le contrôle.

. Le nom de ceux qui ont fait la demande de contrôle.

. Les noms des personnes absentes momentanément de la conférence.

. Le temps de connexion de chacun

. Le nom de la conférence avec une présentation rapide de son objectif.

De même, les fonctionnalités minimales désirées dans la conférence étaient les suivantes :

. La connexion et la déconnexion.

. La possibilité de s'absenter momentanément et de revenir dans la conférence.

. Des outils pour la désignation distante (télépointeurs).

. Un système de vote collectif.

. La prise de contrôle de l'espace public et sa relâche.

La liste des applications coopératives disponibles est l'une des commandes supplémentaires dont dispose le maître de conférence. Il sélectionnera une de ces applications et la diffusera aux participants connectés.

Les deux figures suivantes montrent respectivement l'interface du gestionnaire de conférence pour le "maître de conférence" et celui de l'une des personnes connectées. Nous avons choisi de placer le nom de chaque participant à gauche pour qu'il puisse plus facilement accéder aux fonctionnalités. Dans cette fenêtre, tous les menus sont dédoublés, c'est à dire que le même menu est attaché à deux boutons (celui de gauche et celui du milieu) de la souris. Toute sélection d'une option à partir du bouton du milieu engendrera son exécution classique. Par contre, la sélection de cette même option avec

le bouton gauche ne provoquera pas son exécution mais l'affichage de sa fonction dans la zone de messages dédiée à cet effet.

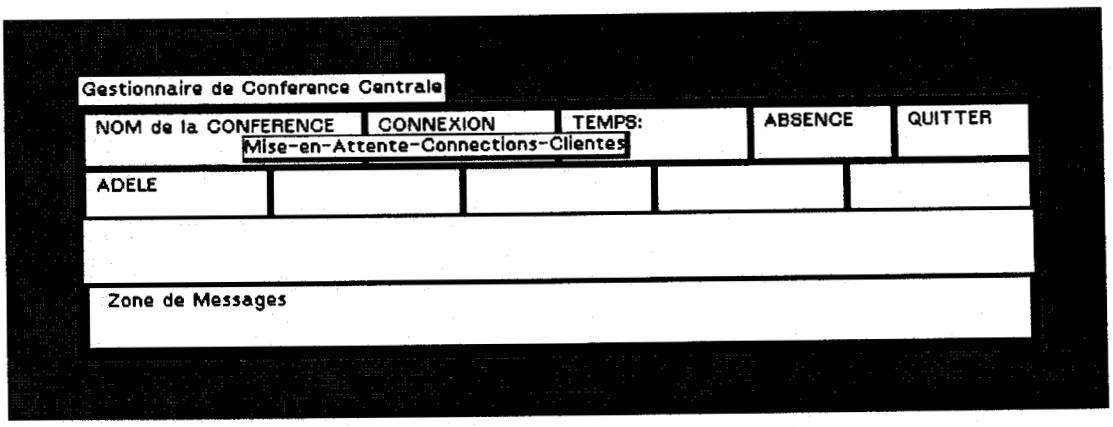


Figure 1.10. *Exemple de l'interface du gestionnaire de conférence pour l'animateur de séance.*

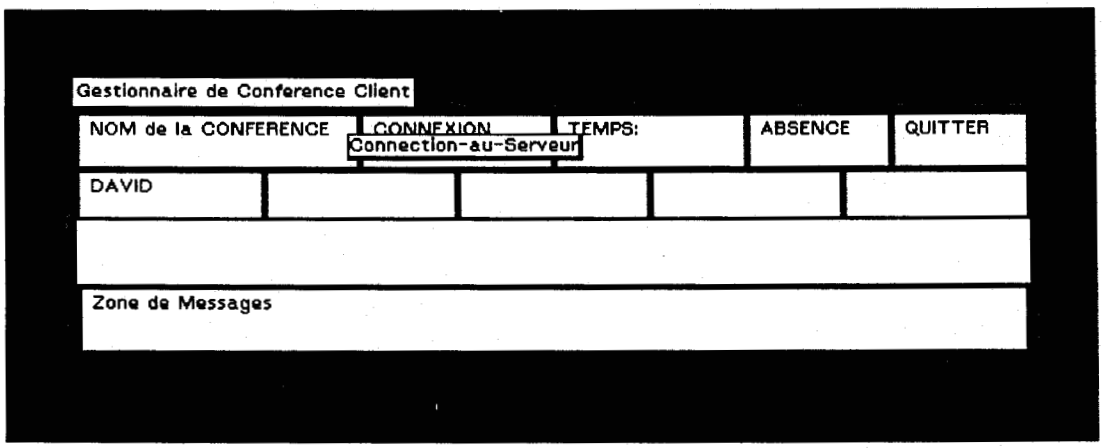


Figure 1.11. *Exemple de l'interface du gestionnaire de conférence pour l'un des participants.*

1.4.3. Processus de déroulement d'une réunion

Pour commencer une séance de travail, il faut que les participants se soient donnés rendez-vous à une date et à une heure précise. A partir de ce moment là, le serveur doit être prêt à accepter les connexions des conférenciers. Un mécanisme d'identification peut également être mis en oeuvre pour s'assurer que les participants qui souhaitent se connecter sont bien ceux prévus pour la réunion.

Concrètement, le maître de conférence doit réaliser les opérations suivantes sur le serveur :

. Il ouvre tout d'abord la fenêtre de gestion de conférence à partir d'une option affichée dans son menu principal.

. Après cette ouverture, il lui suffit de sélectionner l'option menu "mise-en-attente-connexions-clientes" pour être effectivement en mesure d'accepter les connexions. Cela a pour conséquence, le lancement d'un processus qui se chargera de détecter, d'identifier et d'autoriser ou non la connexion d'un utilisateur.

Au fur et à mesure que les clients se connectent, leur nom apparaît dans le bandeau réservé à cet effet sur la fenêtre de gestion de conférence du serveur. Le maître de conférence peut ainsi suivre dynamiquement leur entrée dans la conférence. Lorsque tout le monde est connecté, ou après un délai jugé suffisant par l'animateur, la séance de travail proprement dite peut commencer.

L'animateur va, à partir d'une option menu, ouvrir une fenêtre qui lui donnera la liste des applications coopératives disponibles. Comme nous pouvons le voir sur la figure 1.12, un texte descriptif de chaque application est associé à chacun des noms. Lorsque son choix est fait, il lui suffit de provoquer son exécution pour qu'elle soit active dans son environnement. Pendant ce temps, il ne se passe rien sur les autres stations mais la conversation peut avoir lieu sur le choix de l'application la plus adaptée au besoin de la réunion.

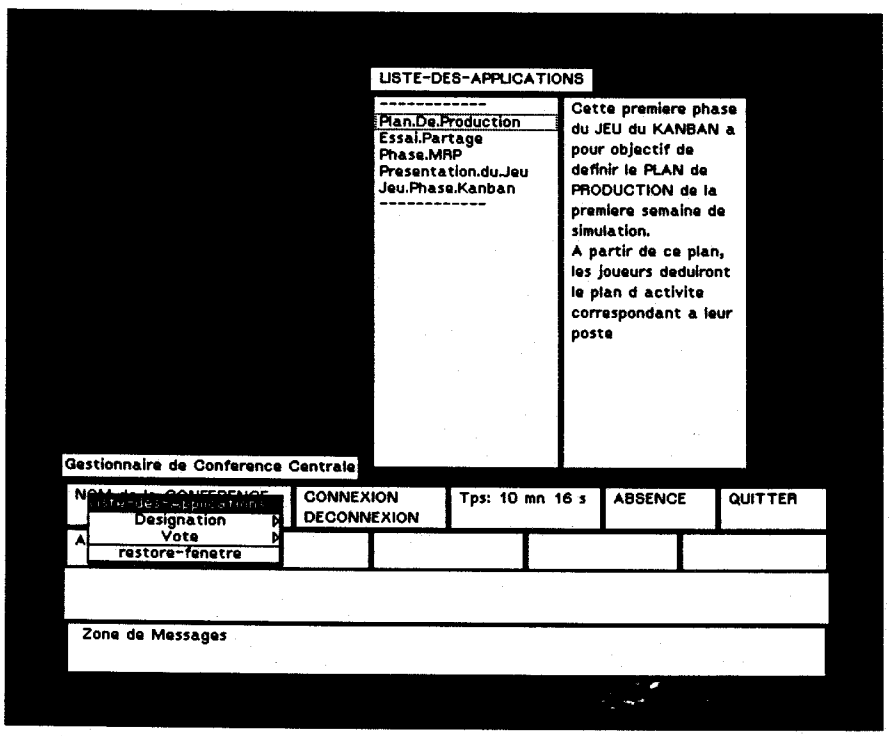


Figure 1.12. *Exemple de l'écran du maître de conférence.*

L'ouverture de l'application sur les stations clientes va être déclenchée par l'animateur sans que les participants aient à intervenir. Lorsque cela est fait, les fenêtres des gestionnaires de conférence sont mises à jour automatiquement en attribuant par défaut le contrôle de l'espace public au maître de conférence.

Les activités qui se dérouleront, seront très fortement liées à l'application. Le passage du contrôle de l'espace public entre les participants va se faire par des demandes qui seront envoyées au gestionnaire de conférence implémenté sur le serveur et chargé d'assurer la cohérence globale. A chaque demande, l'interface des utilisateurs va être mise à jour au fur et à mesure des demandes afin qu'ils puissent toujours savoir comment et pour qui le contrôle de l'espace public va évoluer. Cela est représenté sur leur fenêtre de gestion par des petits icônes placés sous les noms (figure 1.12).

A tout moment, un vote collectif peut être demandé pour évaluer une proposition. Cette dernière se fait alors verbalement à l'ensemble du groupe qui dispose dès lors d'une minute ou deux (laissé à l'appréciation de l'animateur) pour exprimer son choix. Les avantages de ce type de vote qui se déroule de façon parallèle sont : un gain de temps par rapport à un tour de table classique, une confidentialité accrue et une non-influence par rapport aux votes des autres. Le résultat du vote est diffusé au groupe par l'animateur quand l'ensemble des participants a voté ou après un délai jugé suffisant. Nous avons choisi d'afficher les résultats dans une fenêtre indépendante du gestionnaire pour éviter la surcharge d'informations.

Les outils de télépointage que nous avons implémentés sont de quatre natures différentes et permettent à la personne qui a le contrôle de désigner une partie de l'écran à l'ensemble du groupe pour faire ainsi des références relatives à la conversation. Les quatre types de télépointeurs conçus pour un type de désignation particulier et des besoins différents sont les suivants :

. le placement d'une flèche pour désigner un point particulier.

. une mise en vidéo inverse d'une partie de l'écran. Cette zone a la forme d'un rectangle que l'on va construire en plaçant deux coins diamétralement opposés.

. Le déplacement dynamique d'un curseur sur l'écran mais qui ne laisse aucune trace. Il est comparable à la baguette utilisée lors d'une présentation à l'aide d'un rétro-projecteur.

. Le déplacement dynamique d'un curseur qui dans ce cas laisse une trace sur l'écran d'une manière identique au crayon.

1.4.4. conclusion

Cette présentation, d'un point de vue externe, de notre système de conférence avait pour objectif de montrer les différentes fonctionnalités ainsi que son interface utilisateur avant d'exposer plus en détail l'architecture retenue.

II - ARCHITECTURE RETENUE

POUR LA PLATE-FORME

Ce chapître est consacré à la présentation de l'architecture générale de notre système de téléconférence temps-réel.

Il ne faut pas perdre de vue que l'ensemble des décisions que nous avons prises a été contraint par le fait que nous souhaitons exploiter et expérimenter dans un futur proche notre prototype sur le réseau R.N.I.S. (Réseau Numérique à Intégration de Services). Les deux points particuliers différents de notre réseau actuel sur lequel le prototype est implémenté sont :

. Ce réseau ne permettra pas d'avoir les connections directes entre toutes les stations. Nous aurons un modèle de réseau de type étoilé.

. Le débit de ce réseau est nettement plus faible (2*64kb/s).

Dans la première partie, il s'agira d'une description de la plate-forme matérielle et logicielle. Nous détaillerons ensuite les extensions nécessaires que nous avons dues réaliser pour parvenir à l'implémentation d'un prototype.

Par la suite, nous présenterons le modèle que nous avons adopté et décriverons sur des exemples la façon dont nous avons modifié les composants de base pour parvenir à les utiliser dans notre système.

2.1 - Présentation de la plateforme et de Smalltalk

2.1.1 - Présentation de la plateforme matérielle

La figure suivante représente l'environnement matériel de développement du prototype de téléconférence temps-réel.

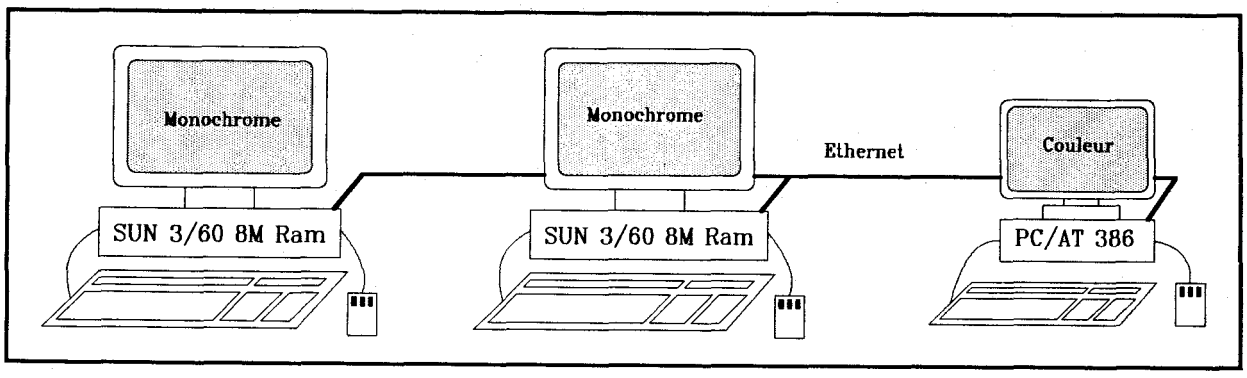


Figure 2.1. *Plateforme matérielle. Deux stations SUN 3/60 connectées sous Ethernet et un PC/AT386.*

Cette plateforme est constituée de deux stations de travail SUN 3/60 et d'un PCAT 386. L'ensemble est connecté sur un réseau local Ethernet. Le PCAT est relié au réseau par l'intermédiaire d'une carte Ethernet avec le logiciel PC/NFS.

Le prototype actuellement implémenté, l'est sur la configuration matérielle décrite précédemment. Cependant, nous avons depuis quelques temps fait l'acquisition d'un Sparc serveur2 et d'une station IPC couleur. La puissance de ces nouvelles machines est environ dix fois supérieure aux anciennes.

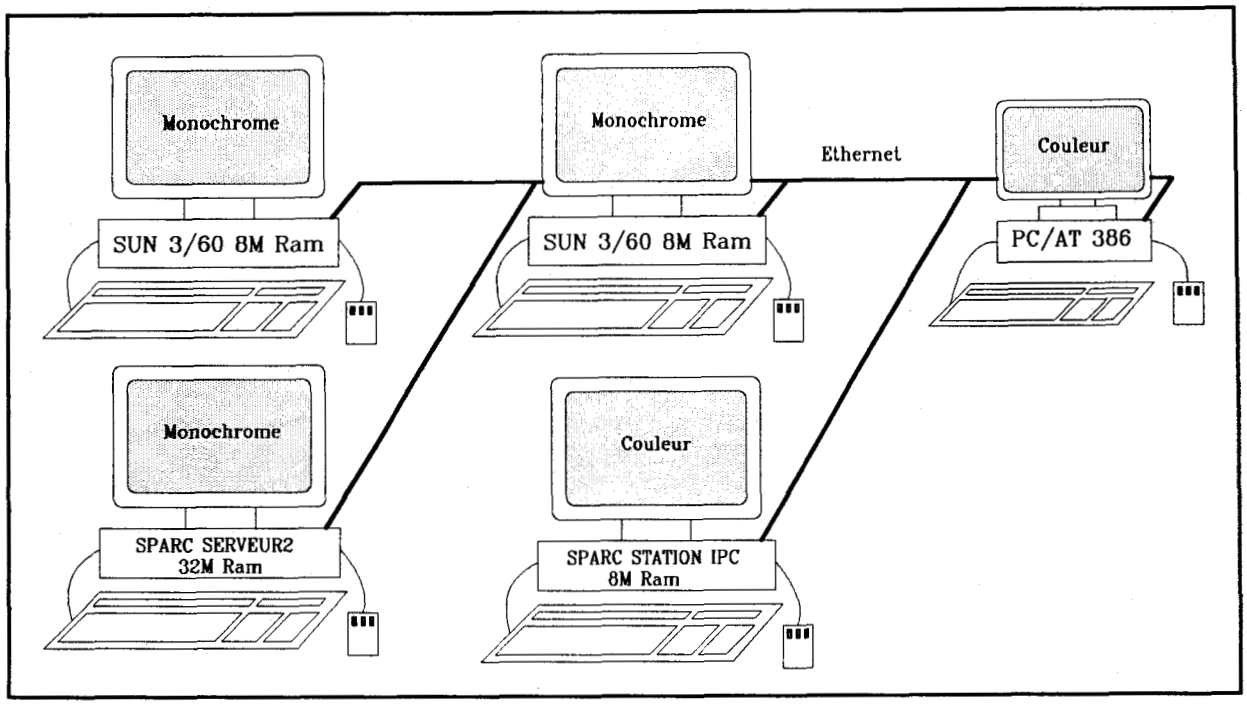


Figure 2.2. *Plate-forme matérielle actuelle. Deux stations SUN 3/60 connectées sous Ethernet, un PC/AT386 et deux Sparc (IPC couleur et Serveur2).*

2.1.2 - Présentation de Smalltalk : environnement de développement orienté objet

2.1.2.1 - Introduction

Smalltalk-80 est l'un des langages à objets les plus connus de nos jours [DEUT 89]. Il puise son origine dans Simula-67, et la version disponible actuellement est le fruit de différentes versions ayant chacune contribué à son enrichissement. Il est à noter que Smalltalk n'est pas uniquement un langage mais aussi un environnement de programmation interactif.

Les travaux d'Alan Kay dans le début des années 70 aboutirent à la première version : Smalltalk-72. On y trouvait déjà les concepts de classes et de messages de Simula. C'est également à cette époque que Xerox Parc pris la décision de travailler sur l'utilisation d'écran bitmap, en vue d'implémenter un environnement multi-fenêtres ainsi que différentes fonctions graphiques.

C'est avec la version suivante (Smalltalk-76) qu'a été introduite la notion d'héritage avec la définition de l'arbre d'héritage ayant pour racine la classe Object. Contrairement à la version précédente, la syntaxe du langage avait été fixée et une innovation très importante due à Larry Tesler [TESL 81] fût offerte. Il s'agissait du

Browser qui offrait la possibilité de parcourir l'ensemble des classes et des méthodes d'une façon plus efficace. Une librairie de classes permettant de concevoir des interfaces utilisateurs fût également proposée.

Au début des années 80, un groupe de 4 constructeurs (DEC, Apple, Hewlett-Packard et Tetronix) implémentèrent sur leur machine respective la dernière version: Smalltalk-80. Ce projet fût conduit par Adèle Goldberg [GORO 83] qui avait remplacé A.Kay au sein de la division recherche. L'objectif était d'aboutir à un Smalltalk portable. En terme d'implémentation, cela se traduisit par la définition de deux éléments: une machine virtuelle dépendant du système d'exploitation et de la station et une image virtuelle commune.

Smalltalk-80 est un des rares langages livré en code source. En effet, son image virtuelle est écrite en Smalltalk, rendant ainsi l'ensemble accessible et modifiable. L'image se compose d'objets prédéfinis. On y trouve aussi le compilateur qui fournira les "byte code" à l'interpréteur, le débugeur, les éditeurs et le système de gestion de fichiers. La machine virtuelle est constituée de primitives écrites en C mais dont le source n'est pas donné. Les deux composants principaux sont un interpréteur et un gérant de mémoire. Pour étendre sa machine virtuelle, le programmeur a cependant la possibilité d'écrire de nouvelles primitives appelées : Users primitives.

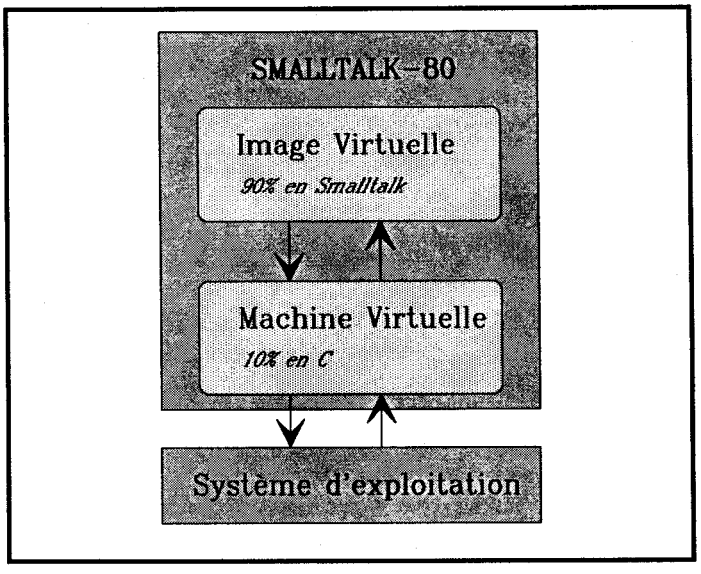


Figure 2.3. *Architecture de Smalltalk. Composée de deux couches: une machine virtuelle (dépendant du système et du processeur) et une image virtuelle (commune sur toutes les stations).*

2.1.2.2 - Smalltalk-80 : un langage à objets

Pour faire face à une complexité de plus en plus importante des applications informatiques, il est apparu nécessaire de fournir des outils de développement aussi performants que les environnements de programmation. Parallèlement à celà, la méthodologie dans la conception des programmes a suscité un intérêt de plus en plus important [BRAD 86] [MEYE 90]. Les concepts clefs qui s'en dégagent sont : la modularité, l'abstraction de données et la réutilisabilité des composants logiciels. A ce titre, de nombreux langages ont vu le jour et en particulier les langages à objets.

Contrairement à la programmation dite "procédurale" qui considère un programme comme un ensemble de données et de traitements (procédures) séparé, la programmation par objets regroupe ces deux ensembles au sein d'une même entité appelée: Objet. Avec une programmation procédurale, une modification dans la structure de données peut entraîner d'importants changements dans les procédures. Cela est d'autant plus grave car d'après des études récentes [MEYE 90], la maintenance, les changements de spécification et les corrections d'erreurs prennent une part très importante dans le coût de développement d'un logiciel. De ce point de vue, la programmation par objets est beaucoup plus souple. Elle permet également une analyse du projet à la fois ascendante et descendante.

En Smalltalk, tous les composants du système sont des objets. Un objet est constitué de données privées et de méthodes, permettant une manipulation de l'information. L'accès aux données privées ne peut se faire que par l'intermédiaire de ces méthodes qui seront exécutées par l'envoi d'un message. Ceci nous conduit à l'un des premiers concepts très important des langages à objets : l'Encapsulation. L'encapsulation assure, via un protocole de communication, une intégrité et une abstraction de données. Elle permet également de cacher la structure interne de l'objet.

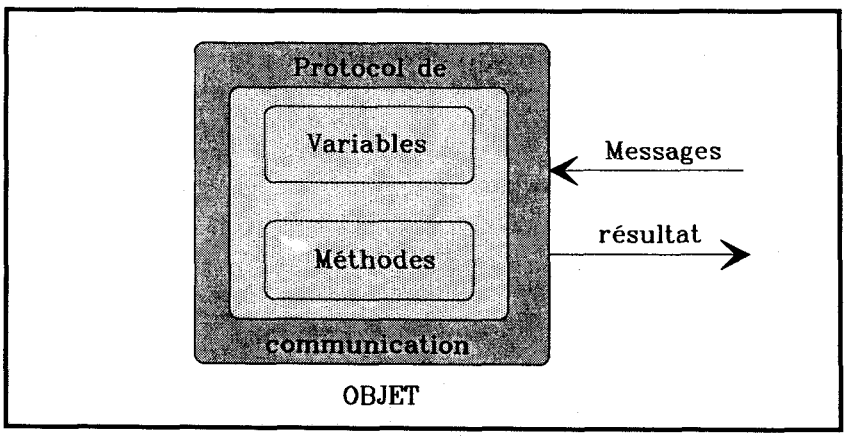


Figure 2.4. *Structure d'un objet.*

En résumé, l'objet peut être vu comme une boîte noire qui communique avec le monde extérieur par des messages. L'implémentation physique des données n'est pas connue. L'objet est le seul à manipuler ses variables. Cette encapsulation offre par ailleurs, une lisibilité plus facile des programmes, en cachant les données et les méthodes internes. Un choix judicieux dans le nom des méthodes est également un gage de réussite dans la compréhension du programme.

Comme nous l'avons vu, Smalltalk est un langage de classes. Une classe permet de gérer un ensemble d'objets qui auront tous la même structure de données et les mêmes méthodes. Les objets créés à partir de ces classes s'appellent des instances. En Smalltalk, tout objet est instance d'une classe. De même, les classes sont instances de classes particulières appelées : métaclasses. Dans la version actuelle, chaque classe à une métaclasse qui lui est directement associée, contrairement à la version précédente qui n'avait qu'une métaclasse. Cela est très important, car cela offre la possibilité d'avoir des messages dits "de classe" personnalisés pour chacune d'elles. Ainsi, il est possible d'initialiser directement les attributs de l'objet au moment de l'instanciation.

Les classes sont organisées d'une façon arborescente à partir de l'arbre d'héritage. L'héritage est un des concepts fondamentaux de Smalltalk. Qu'il soit simple ou multiple, il permet de factoriser des structures de données et des méthodes offrant ainsi la possibilité de réutiliser code et données. Ainsi, une classe qui est sous-classe d'une autre :

- récupère la structure de données et les méthodes de sa super classe ;

- peut définir à nouveau d'autres variables ainsi que d'autres méthodes. Une méthode portant le même nom peut être réécrite inhibant alors la méthode héritée. En effet, la recherche dynamique d'une méthode sur un objet commence dans la classe de l'objet et c'est la première méthode rencontrée qui est exécutée (héritage simple). Cette possibilité de surcharge est très pratique, car elle permet d'obtenir une uniformité quant aux noms des messages.

La figure suivante représente les arbres d'héritages des classes et des métaclasses.

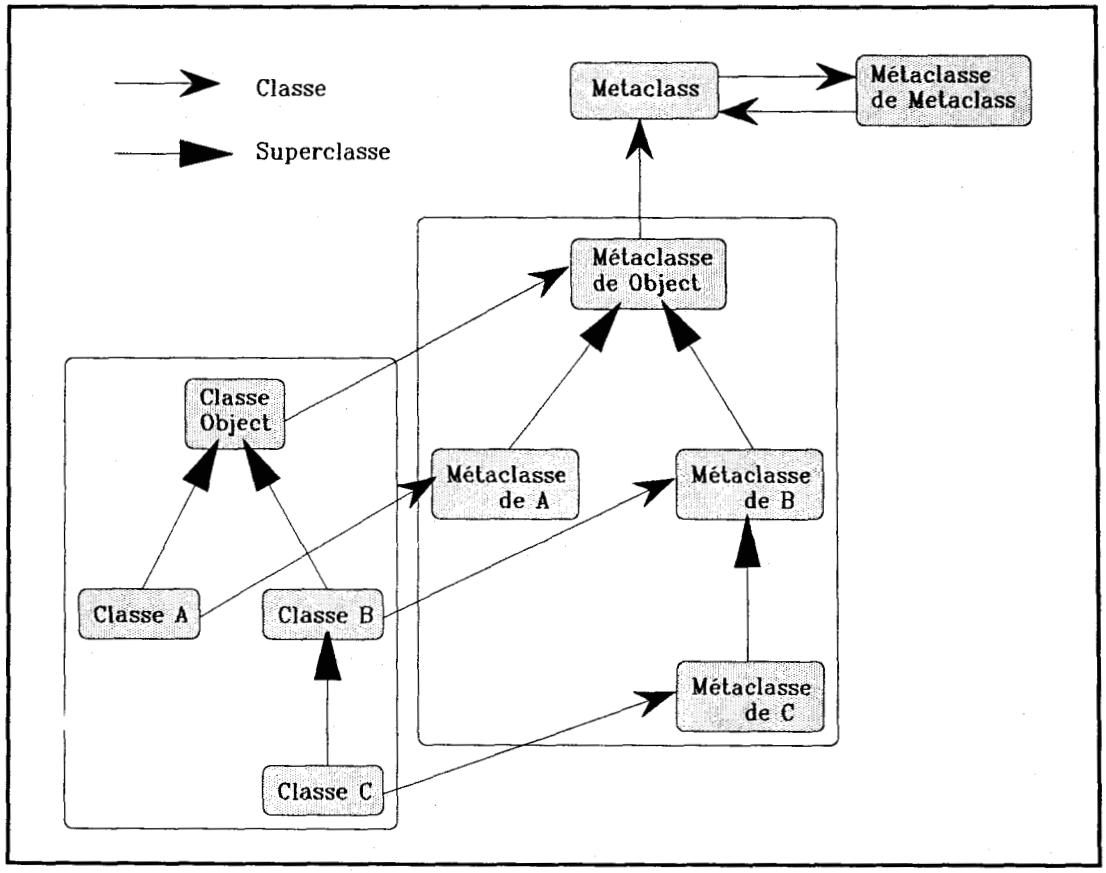


Figure 2.5. *Structure de l'arbre d'héritage des classes et des métaclasses.*

2.1.2.3 - Conclusion

L'environnement de programmation Smalltalk est très riche en offrant une panoplie d'outils d'aide très puissants et une riche bibliothèque de classes. Programmer en Smalltalk, c'est avant tout lire ce qui a déjà été écrit pour se familiariser avec le style de programmation. En effet, vouloir respecter le concept de réutilisabilité commence par une recherche des classes se rapprochant le plus de son besoin en vue d'y définir une sous-classe et d'y réécrire les méthodes nécessaires. Cela peut également consister à récupérer le code d'une méthode afin de l'adapter à ses besoins. Cependant, la version actuelle se compose de 400 classes et de 6000 méthodes. Son apprentissge prend donc beaucoup de temps. Par la suite, l'investissement "temps" du départ devient très rentable, car Smalltalk permet de programmer très rapidement lorsque l'on connaît l'environnement.

Le choix de Smalltalk, comme langage support de notre prototype [VIDEc90], s'est imposé pour les raisons suivantes :

- Smalltalk offre un environnement multi-fenêtres très performant et une bibliothèque permettant le développement rapide d'applications interactives.

- étant livré en source, il nous était possible de modifier et d'adapter les différents composants à nos besoins.

- les versions disponibles sur stations SUN étaient fournies avec des mécanismes de communication.

- la version Smalltalk 80 de Parc Place utilisée est portée sur un grand nombre de stations (PC/AT386, stations de travail : HP, DEC, IBM, SUN, SONY...).

La partie suivante présente l'extension que nous avons apportée au système, afin d'obtenir un mécanisme de partage d'objets entre des stations distantes.

2.2 - Extension de l'environnement Smalltalk 80 pour la coopération

2.2.1 - Présentation du système de partage d'objets

L'environnement Smalltalk, tel qu'il est fourni par Parc Place, ne permet pas une distribution d'objets entre plusieurs images. Cependant, nous disposons de certaines facilités pour communiquer et pour donner un format de représentations externes à des objets. Ces facilités se traduisent dans l'environnement par un accès aux Sockets sous Unix et par la fourniture d'un outil appelé B.O.S. (Binary Object Storage). Ce dernier permet de sérialiser et de désérialiser des objets Smalltalk.

2.2.1.1 - Etat de l'art

Différents travaux, menés dans le domaine d'un Smalltalk distribué, nous ont permis de définir notre cahier des charges.

Ces travaux, que nous allons présenter brièvement, ont entraîné des modifications plus ou moins profondes de Smalltalk. Les ambitions des chercheurs étant différentes, certains n'ont modifié que l'image virtuelle, alors que d'autres se sont intéressés directement à la machine virtuelle.

Les travaux peuvent se classer en trois catégories :

. Dans la première catégorie, l'on trouve les travaux de Vegdalh sur le déplacement de structures entre images Smalltalk [VEGD 86]. L'ensemble est fondé sur une représentation linéaire des objets. Le B.O.S. disponible actuellement, ne l'est que depuis la version 2.5. Cet article, que nous n'allons pas développer, nous a permis de réaliser un outil de sérialisation d'objets dans la première version de Smalltalk que nous avions utilisée (version 2.3). Il est à noter que le B.O.S. réalise dans Smalltalk ce que les XDR (eXternal Data Representation) font sous N.F.S. (Network Files System).

. La deuxième catégorie correspond aux travaux portant sur la réalisation d'un Smalltalk distribué moyennant une modification de l'image virtuelle [CULL 87].

. Enfin, la troisième catégorie fait référence à des travaux sur un Smalltalk distribué ayant entraîné une modification profonde de la machine virtuelle.

a) Smalltalk distribué par modification de l'image virtuelle

L'objectif de Mac Cullough [CULL 87] et Bennet [BENN 87] consistait à développer un Smalltalk distribué avec les conditions suivantes :

. Un message Smalltalk peut être envoyé à un objet localisé sur une machine différente.

. Une transparence de programmation au niveau de l'utilisateur.

. Pas ou très peu de changement au niveau de la machine virtuelle.

. Une modification minime du compilateur (utilisation du langage de base).

. Le moins de changement possible à apporter à l'image standard.

Les décisions essentielles prises par Mac Cullough et Bennet, face au choix qui s'offrait à eux pour répartir leur système distribué et conserver l'intégrité entre les instances et leurs classes, sont résumées dans le tableau suivant (figure 2.6).

Dans la réalisation, Mac Cullough et Bennet ont décidé que les classes et instances seraient corésidentes.

Ils proposent dans un premier temps, un modèle simple pour arriver à un système distribué. Les objets accessibles, depuis les autres machines, sont représentés par des objets mandataires (Proxy) qui acheminent les messages à travers le réseau.

Afin d'améliorer les performances, ils suggèrent, dans un second temps, de ramener l'objet distant sur le même site que l'objet exécuteur. Dans ce cas, la règle de transparence sera violée, car le programmeur devra indiquer qu'il s'agit d'un "call by move". Des problèmes d'équivalences d'objets et de ramasse-miettes doivent être résolus.

Décisions essentielles	Avantages	Inconvénients	Notre avis
Ne pas permettre la modification des classes (tolérer la création de sous-classes)	- Plus de test de compatibilité en classes - les instances peuvent être acceptées sur toutes les machines	- Le système n'est pas souple - Difficilement acceptable pour un programmeur	- Cette approche peut être intéressante si on ne veut livrer que le support d'application et non un système de développement
Maintenir une copie étalon de la hiérarchie des classes	- On peut continuer à faire évoluer les classes (système évolutif)	- Difficulté à maintenir la cohérence de l'ensemble sur le réseau (instant de mise à jour) - Fiabilité du système (passive) - Performance	- Cette approche sera difficilement fiabilisable
Maintenir plusieurs copies de la hiérarchie des classes	- Tout en conservant un système évolutif ceci permet de fiabiliser le système car chaque machine peut supporter la copie à jour	- Il faut utiliser des algorithmes de maintenance de données dupliquées et réparties	- Difficilement réalisable - Performances très faibles sur un réseau (volume d'échange important)
Faire pointer les objets vers les classes qui les ont créés	- On se retrouve dans la même situation qu'en Smalltalk 80 (mono-utilisateur)	- Il faudra mettre à jour le pointeur lorsque la classe se déplacera	- Plus facile d'interdire aux classes de se mouvoir - Les performances risquent d'être assez mauvaises
Lorsque cela est possible, faire pointer l'objet vers une classe "compatible" dans la machine où il est situé	- Résoud les problèmes de performances (particulièrement vrai s'il s'agit d'une application où les classes sont très stables	- Difficile de tester la compatibilité des classes	- Seule une personne peut s'assurer de la compatibilité - Démarche pas très réaliste
Décider qu'il n'y aura pas de classes "distantes" vis à vis des instances	- Fiabilité plus grande - Performance acceptable	- La mobilité des objets est réduite	- Dans ce système où les classes sont constantes cela revient au deuxième cas

Figure 2.6. *Tableau récapitulatif des décisions essentielles prises par Mac Cullough et Bennet.*

b) Smalltalk distribué par modification de la machine virtuelle

En constatant les principaux défauts de l'approche précédente, à savoir :

. difficulté avec les entrées/sorties (par exemple, lorsqu'une machine virtuelle distante exécute une méthode qui fait appel à des variables globales, à qui va-t-elle envoyer les messages ?).

. pas de debugging à distance.

M. Schelvis [SCBL 88] et D. Decouchant [DECO 86] ont choisi l'approche suivante pour résoudre ce problème :

. La machine virtuelle est modifiée afin d'ajouter des attributs dans l'entête de l'objet.

. Les processus contiennent la localisation de la machine qui les a créés. Les objets appartenant à chaque machine et gérant les entrées/sorties (Display, Sensor...) auront un flag qui indiquera qu'ils sont attachés à leur hôte.

. Pour maintenir l'intégrité des données en ce qui concerne les objets dupliqués, il est nécessaire d'utiliser un algorithme d'estampillage ordonné.

. Le ramasse-miettes spécifique, mis en oeuvre par Schelvis, repose sur celui de Smalltalk de Berkeley appelé "generation Scanvenning".

c) Tableau récapitulatif des avantages et des inconvénients des différents travaux

	Inconvénients	Avantages
Vegdahl	* ne marche pas entre des machines virtuelles différentes (méthode compilée) * précision des nombres	* uniformisation du mécanisme * prise en charge de "gros" objets * prise en charge de structure circulaire * utilisation facile
Bennet / Mac Cullough	* mobilité réduite des objets * debugging des objets distants impossible * difficulté de prise en charge des entrées/sorties	* pas (peu) de modification de la machine virtuelle et du compilateur * portabilité sur des machines virtuelles différentes
Schelvis / Decouchant	* perte de temps en local * pas de portabilité entre machines virtuelles	* bonne gestion des Entrées/Sorties * gestion d'objets dupliqués * optimisation du test des objets locaux / distants / hôte intégré à la machine

2.2.1.2 - Définition de notre système

Après avoir analysé ces différents travaux, nous sommes arrivés à la conclusion suivante. Pour réaliser un Smalltalk réellement distribué, il faut modifier la machine virtuelle. N'ayant pas besoin de toutes les fonctionnalités d'un système distribué et ne disposant pas des codes sources de la machine virtuelle, nous n'avons modifié que l'image virtuelle [VIDEb90].

Notre principal objectif était de permettre un accès aussi transparent que possible à des objets distants. Pour l'atteindre nous nous sommes fixés certaines hypothèses :

. Les classes sont identiques pendant le temps d'exécution de l'application.

. La création de ces objets partagés n'a pas besoin d'être transparente pour le programmeur. Une phase d'initialisation pour partager ainsi que pour accéder à un objet est nécessaire.

. Le nombre d'objets que nous avons à partager n'est pas élevé. Il s'agit d'objets "applicatifs" et non pas d'objets "systèmes".

a) Modèle du système de partage d'objets

Notre modèle pour ce système est relativement simple. Les objets partagés ('objetPartagé') accessibles depuis d'autres machines sont représentés sur celles-ci par des objets mandataires que nous appellerons 'proxy'. Le rôle d'un 'proxy' consiste à rediriger les messages sur l'objet partagé. Les messages seront alors éxécutés et les objets "résultat" renvoyés.

La création d'un objet partagé sur une station et des objets mandataires correspondants sur les autres stations n'est pas transparente, car elle nécessite une phase d'initialisation. Toutefois, après cette phase d'initialisation, l'utilisation de l''objetPartagé' ou du 'proxy' est totalement transparente pour l'utilisateur.

L'initialisation d'un objet que l'on souhaite partager a pour conséquence la création d'un 'objetPartagé' qui va l'encapsuler et lui offrir un mécanisme de protection contre l'accès concurrent de messages. Une identité globale (nous entendons globale à l'ensemble des machines) sera également attribuée à l' 'objetPartagé'.

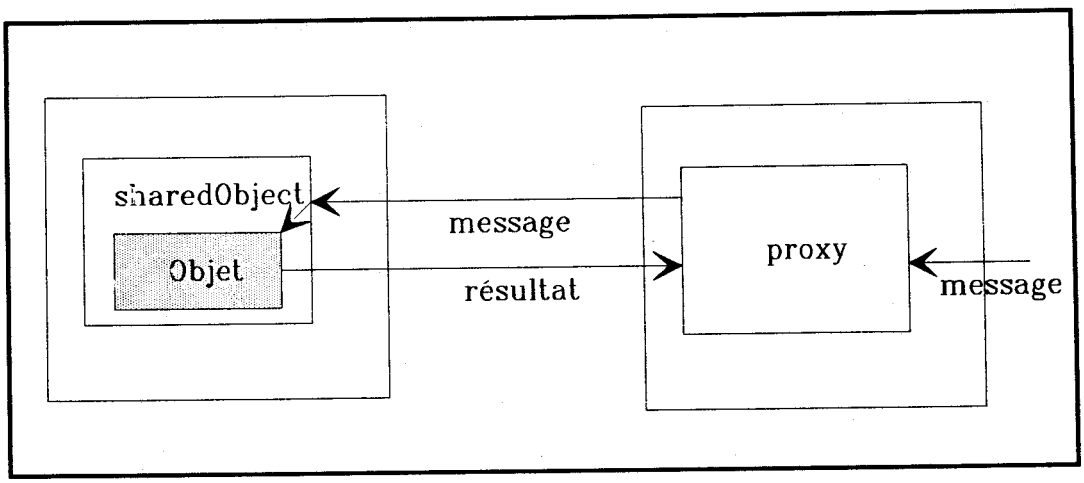


Figure 2.7. *Exemple d'un objet partagé sur une station avec son objet mandataire sur une autre.*

Un message est envoyé au 'proxy'. Celui-ci le récupère et le met en forme afin de l'envoyer par le réseau sur la station où le 'objetPartagé' correspondant (même identité globale) est localisé. Le 'objetPartagé' exécute le message et renvoie au 'proxy' l'objet 'résultat'. Il est à noter que le processus lié à la méthode qui vient d'envoyer un message à un 'proxy' est en attente jusqu'au moment où l'objet résultat est récupéré.

Comme nous l'avons déjà dit, notre architecture réseau est de type étoilé avec le serveur au centre. Le système de partage d'objets permet d'avoir les objets partagés répartis sur l'ensemble des stations. La figure suivante est un exemple de répartition.

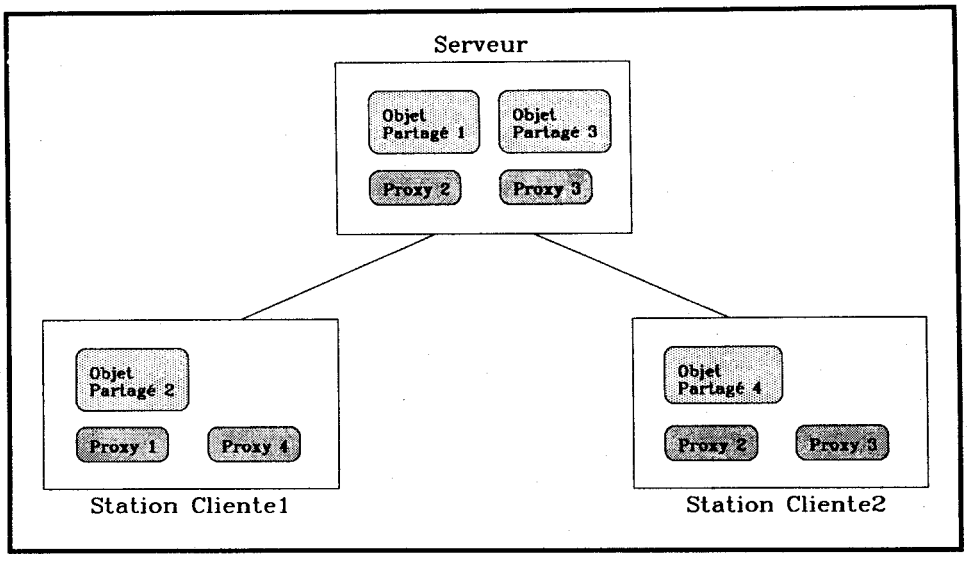


Figure 2.8. *Exemple de répartition d'objets.*

La station cliente 1 peut par exemple envoyer des messages aux objets 1, 2 et 4. L'utilisateur n'a pas à se soucier de la localisation réelle des objets (dans ce cas, seul le 2 est local). Il ne peut par contre pas accéder au 3 car la phase d'initialisation n'a pas eu lieu et il ne dispose pas du 'proxy3' correspondant.

b) Architecture du système de Partage d' Objets

L'interface avec les sockets fût le point de départ pour la réalisation de ce système (version 2.5). Les sockets sont des mécanismes qui permettent d'échanger d'une façon bidirectionnelle des données entre processus situés sur des stations différentes. Ces échanges s'effectuent toujours suivant un mode client/serveur. La socket client sera celle qui devra faire les démarches appropriées pour acquérir les services de la socket serveur. Une socket s'identifie par une adresse internet et par un numéro de port. L'adresse internet correspond à l'adresse de la machine hôte et le numéro de port correspond à l'un des services de la machine hôte.

Dans la version 2.5 de Smalltalk-80, l'interface avec ces sockets a été réalisée. C'est à dire que l'on peut par instanciation de la classe Socket créer une socket serveur. Il suffit de passer comme paramètre dans le message de classe un numéro de port (choisi arbitrairement ex 1000). Sur une station cliente l'opération est quasiment identique. Il faut instancier la même classe mais avec un message différent. Il faut donner comme paramètre le nom de la machine hôte et le même numéro de port (ex 1000).

A partir de ce moment, on a sur les stations, des objets qui vont permettre un échange fiable d'informations (cette vérification et cette assurance est gérée par les couches T.C.P.). En simplifiant, ces objets 'socket' Smalltalk ont deux variables d'instances qui pointent respectivement sur un stream d'entrée et un stream de sortie. Pour envoyer un message, il suffit d'écrire sur le stream de sortie par l'intermédiaire d'un message approprié que l'on envoie à notre objet 'socket'. Lorsqu'un message est envoyé par une station, il est placé dans le steam d'entrée de la station réceptrice. Il suffit alors d'aller lire le message. Il existe par ailleurs des messages qui permettent de savoir s'il y a quelque chose dans ce stream d'entrée.

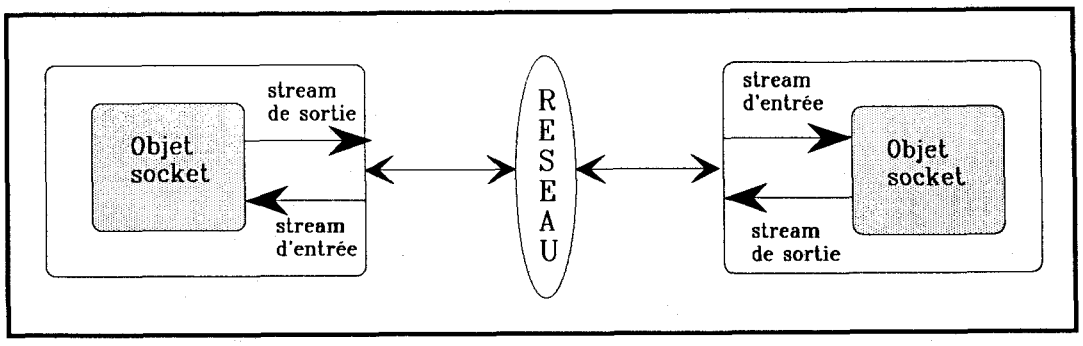


Figure 2.9. *Schématisation des sockets*

Le B.O.S. (Binary Object Storage) est un outil qui permet de sérialiser et de désérialiser n'importe quel objet Smalltalk. La sérialisation d'un objet consiste à lui donner un format de représentation externe. Sérialiser un objet se fait par l'instanciation de la classe BinaryStorage avec le message "put: unObjet". L'objet renvoyé par cette méthode est une chaîne de caractères. Cette chaîne peut ensuite être gérée par des streams et en particulier ceux d'entrées et de sorties des objets 'sockets'. La remise en forme de l'objet se fait d'une façon identique par le message "from: uneChaîneDeCaractère".

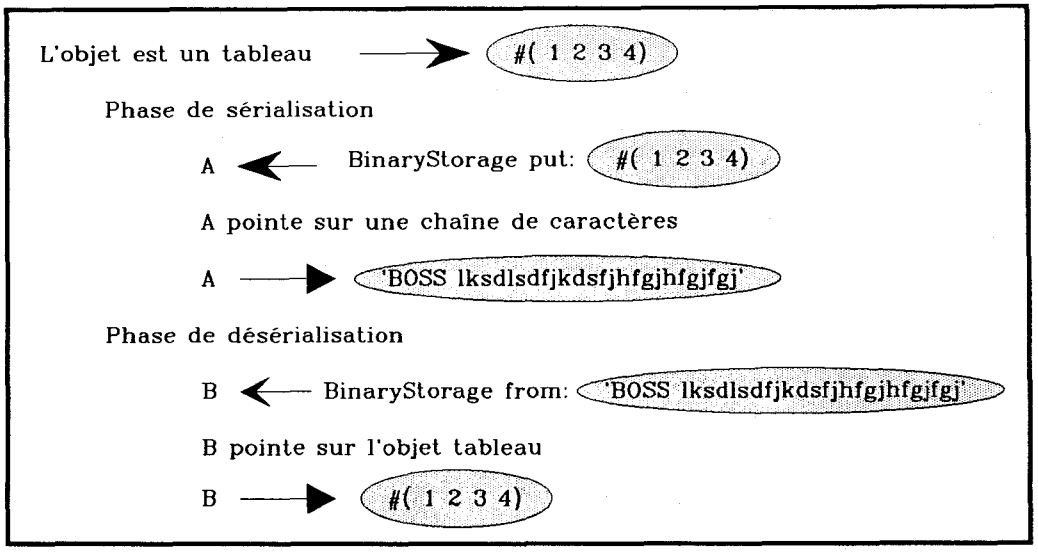


Figure 2.10. *Exemple de sérialisation et de désérialisation d'un objet Smalltalk.*

Nous allons maintenant présenter le fonctionnement ainsi que les divers éléments constituant le système. Nous terminerons par une présentation des diverses classes caractéristiques. Un exemple précis de partage d'un objet avec les méthodes principales est donné en Annexe I.

Fonctionnement du système de partage d'objets :

Pour la mise en place du système de partage d'objets, certains pré-requis sont nécessaires. Il faut que l'ensemble des images soit cohérent. C'est à dire que les classes d'objets qui pourront être échangées dans les paramétres des messages devront être identiques. En effet, pour pouvoir exécuter des messages, il faudra qu'ils aient été définis dans les classes. D'autre part un noyau minimal de classes doit être implémenté sur chaque station pour que la connexion s'effectue correctement. Il s'agit principalement des classes du B.O.S., des sockets et du système de partage que nous présenterons ultérieurement.

Nous n'avons pas mis en place au moment de la connexion une vérification de cette cohérence. Toutefois, face à ce problème, il est toujours possible de mettre les classes à jours par un échange de fichiers.

* Phases de connexion :

La connexion au serveur est la première opération que les stations clientes doivent accomplir. Pour ce possible, le serveur doit se trouver dans une phase d'attente de connexions. Cette phase est obligatoirement la première chose à réaliser dans une session de téléconférence. Placer le serveur dans cet état aura pour effet les actions suivantes:

. Un processus a été lancé pour détecter la demande de connexion des stations clientes.

. Une variable globale nommée "LinkTable" se trouvant sur toutes les stations a été initialisée. Elle permet l'accès à un tableau dont les éléments pointeront sur des objets particuliers. Ces derniers (nous les présenterons un peu plus loin) permettront l'accès à l'objet partagé ou à l'objet mandataire. Les identités globales des objets partagés serviront d'indices au tableau.

. Un objet partagé très important, localisé sur le serveur, est également initialisé. Il s'agit du serveur de nom. S'agissant du premier objet partagé du système, son identité globale sera égale à 1 (donc accessible à partir du premier élément de la "LinkTable").

En se connectant au serveur, une station cliente créera une socket entre les deux stations. Pareillement aux conséquences décrites précédemment, la phase de connexion au serveur servira à réaliser les actions suivantes.

. Un processus sera lancé sur le serveur ainsi que sur la station cliente.

Son rôle va consister à récupérer les paquets reçus du réseau et les remettre en forme pour pouvoir être traités. Sur le serveur, on trouvera un processus pour chaque station cliente connectée. Le format d'échange des données sera présenté un peu plus loin.

. La variable "LinkTable" sera initialisée. Il s'agit toujours d'un pointeur sur un tableau.

. Nous avons vu que sur le serveur, l'objet pointé par le premier élément du tableau était le serveur de nom. Ici, l'objet pointé sera l'objet mandataire correspondant. C'est par son intermédiaire que le partage ou l'accès à des objets distants sera possible.

La figure 2.11 schématise le système à ce niveau de présentation.

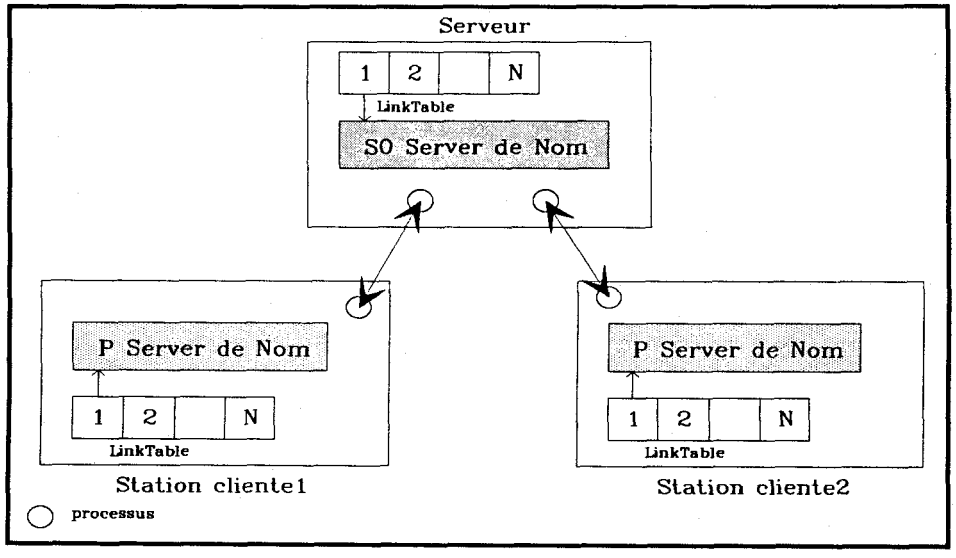


Figure 2.11. *Architecture du système de partage d'objets. Sur chaque station, on trouve une "LinkTable" dont le premier élement pointe sur le "serveur de nom" (sur le serveur il s'agit de l'objet partagé et sur les autres stations il s'agit de l'objet mandataire). On remarque également que sur le serveur, il y a un processus déclenché par station cliente connectée.*

* Partage d'un objet :

Lorsque les connexions sont établies, le partage d'un objet est possible à partir de n'importe quelle station. Il se fait par l'envoi d'un message au serveur de nom (ou par l'intermédiaire de son 'proxy' sur les stations clientes). Dans ce message deux paramètres sont à spécifier : un identificateur et l'objet à partager. Un objet partagé ('objetPartagé') sera alors créé avec une identification globale différente des autres. Les identités globales sont attribuées d'une façon incrémentale au fur et à mesure des

demandes. Seule, la "LinkTable" de la station concernée sera mise à jour. Les autres stations devront prendre elles-mêmes l'initiative de la mettre à jour par l'initialisation de l'objet mandataire correspondant. Nous n'avons pas voulu créer automatiquement les 'proxy' sur les stations clientes car elles n'en ont pas forcement besoin et qu'il est alors inutile d'encombrer le réseau.

* Accès à un objet partagé :

Il faut également passer par une phase d'initialisation pour accéder à un objet partagé. Elle consiste à l'envoi d'un message au serveur de nom. Cette fois-ci, le message n'a qu'un argument : l'identificateur de l'objet.

L'objet renvoyé par le serveur de nom sera un 'proxy' dont l'identité globale sera égale à l'objet partagé correspondant. La "LinkTable" sera également mise à jour.

La figure 2.12 illustre les explications précédentes.

Deux objets ont été déclarés comme objets partagés. Il s'agit de 'O1' sur le serveur et de 'O2' sur la station "client2". Un proxy 'PO1' correspondant à 'O1' a été créé sur le serveur. On trouve également 'PO2' sur la station "client1" pour accéder à 'O2'.

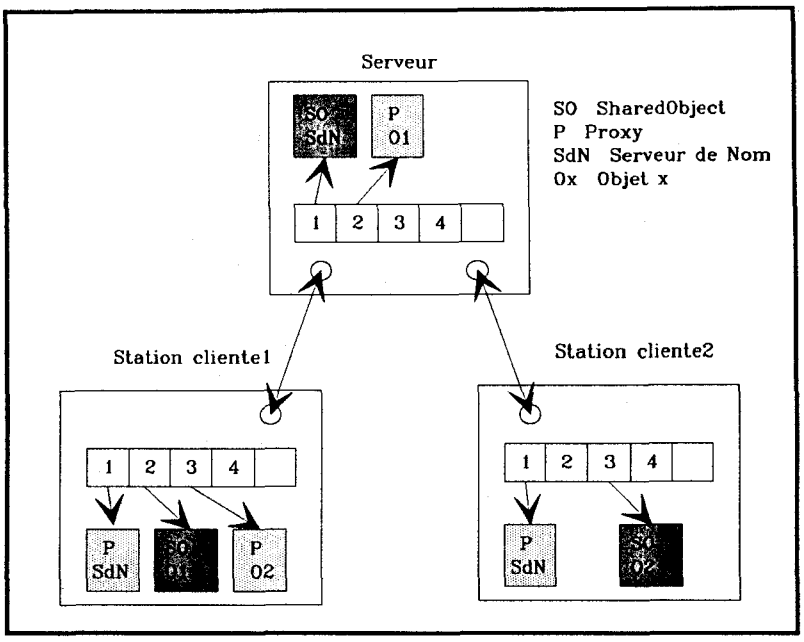


Figure 2.12. *Exemple de partage d'objets entre plusieurs stations*

* Format des informations échangées :

Pour permettre une interprétation correcte, il y a nécessités d'adopter une mise en

forme particulière et d'ajouter des données supplémentaires lorsqu'un message ou un objet résultat est envoyé sur une autre station. Comme nous l'avons vu, les paquets sont gérés à partir de "Streams". Un paquet va donc être structuré en quatre champs qui seront conformes à la représentation externe des données du B.O.S.. Le premier champ est un bit qui représente l'identité globale de l'objet récepteur.

Le second champ du paquet symbolise le type du paquet :

- paquet "Ouvert" pour indiquer la création d'un nouveau 'objetPartagé' ou 'proxy',
- paquet "Fermé" pour indiquer qu'un 'objetPartagé' ou 'proxy' n'est jamais utilisé,
- paquet "Exécution" pour envoyer un message (mécanisme d'appel),
- paquet "Réponse" pour envoyer un objet résultat après l'exécution d'un message (mécanisme de réponse).

Le troisième champ précise la taille du quatrième champ.

Le quatrième champ est l'objet linéarisé par le B.O.S. (il peut s'agir du message ou de l'objet résultat).

Présentation des classes du système :

La hiérarchie des classes est la suivante :

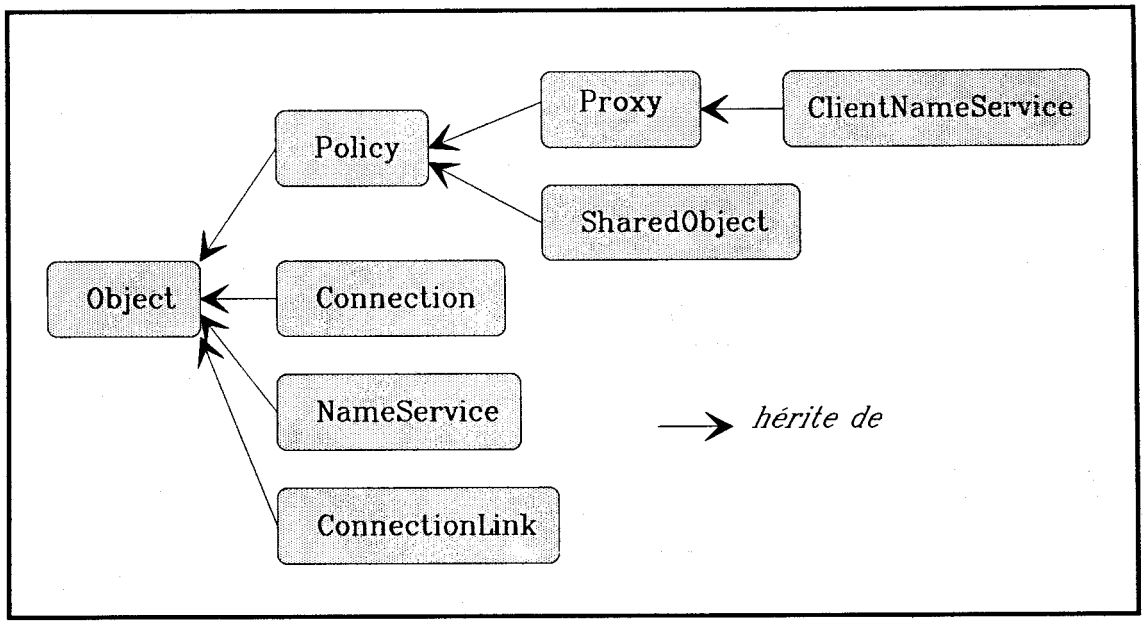


Figure 2.13. *Hiérarchie des classes de notre système de partage d'objets*

La classe Policy :

La classe Policy est une classe abstraite, que l'on n'instancie pas directement mais qui offre un comportement général pour ses sous-classes (SharedObject et Proxy). Cette classe a 5 variables d'instances, à savoir :

. localObject pointe sur l'objet que l'on a décidé de partager
. reply pointe sur l'objet renvoyé par l'exécution d'un message sur le "localObject"
. gid est l'identité globale de l'objet attribuée par le serveur de nom que nous détaillerons par la suite
. replySem pointe sur un sémaphore permettant de suspendre l'exécution d'un processus lorsqu'un message est envoyé sur une autre station
. mutex pointe sur un sémaphore d'exclusion mutuelle permettant de ne pas interrompre l'exécution du message sur le localObject.

Une instance de la classe Policy peut encapsuler n'importe quel objet Smalltalk. Cette instance capture tous les messages qui lui sont envoyés. La récupération des messages est effectuée en redéfinissant la méthode "doesNotUnderstand". Il s'agit d'un mécanisme fréquemment utilisé en Smalltalk. Le message "doesNotUnderstand" est généré par l'interpréteur lorsque la recherche dynamique échoue. Les deux caractéristiques principales de cette capture sont les suivantes. La première consiste en une redirection des messages à travers le réseau sur l'objet distant lorsque l'objet partagé ne se trouve pas sur la station. La seconde fournit un mécanisme de protection contre l'accès concurrent des messages.

La classe SharedObject :

Cette classe, sous-classe de Policy, permet à tout objet Smalltalk d'être partageable, c'est-à-dire de devenir accessible par plusieurs stations. Une instance de cette classe est créée lors de la déclaration d'un objet que l'on veut partager. Le serveur de nom lui attribuera alors une identité globale. La variable d'instance "localObject" héritée de la classe Policy pointe alors sur l'objet et la variable gid est initialisée avec son identité globale. Un 'objetPartagé' offre le mécanisme de protection contre l'accès concurrent des messages si l'utilisateur respecte une règle : ne pas envoyer directement de message à l'objet encapsulé sans passer par le 'objetPartagé'.

La classe Proxy :

Une instance de cette classe est un objet que nous avons appelé 'proxy', et présenté précédemment, qui fait référence à un objet partagé ('objetPartagé') lorsque celui-ci n'est pas local. La création d'un 'proxy' ne peut être faite que si l'objet partagé

correspondant existe. Elle nécessite un seul paramètre : l'identificateur utilisé pour créer l'objet partagé. Contrairement à une instance de la classe SharedObject, sa variable d'instance "localObject" est à nil. Lorsqu'un message lui est envoyé, il ne peut l'exécuter, mais grâce au mécanisme d'indirection, il renvoie le message au 'objetPartagé' correspondant et attend l'objet résultant de l'exécution du message sur le 'objetPartagé'.

La classe Connexion :

L'instanciation de cette classe permet tout d'abord d'établir une connexion entre deux machines (une station cliente et le serveur). Une instance de cette classe gère une socket, la sérialisation des messages et la remise en forme des paquets reçus. Au moment de l'instanciation, un processus est démarré sur la station cliente pour récupérer les données venant de la socket, les remettre en forme et les envoyer aux objets correspondants.

La classe ConnectionLink :

Cette classe a deux variables d'instances que sont "policy" et "connections". Une instance de cette classe permet donc un lien entre un objet (policy) qui est, soit un 'objetPartagé', soit un 'proxy' et une 'connection'. Les éléments de la LinkTable sont des instances de ConnectionLink. Les éléments ne pointent pas directement sur le 'objetPartagé' ou le 'proxy'. C'est pour simplifier les explications du système de partage d'objets que nous l'avions présenté de cette façon.

Les classes NameService et ClientNameService :

Ces deux classes correspondent au serveur de nom. L'instanciation de la classe NameService sur le serveur et de la classe ClientNameService sur les stations clientes vont respectivement servir à réaliser la mise en attente des connexions clientes et la connection au serveur que nous avons présentée précédemment.

Envoi de messages à des objets distants :

Nous allons illustrer le cheminement d'un message lorsqu'il est envoyé à un 'proxy'. Le 'proxy' se trouve sur une station cliente et l'objet partagé sur le serveur.

1. Dans l'exécution d'une méthode, un message est envoyé à un 'proxy'.

2. La recherche dynamique de la méthode correspondante échoue, ce qui implique l'exécution de la méthode "doesNotUnderstand" que nous avons redéfinie dans la classe Policy.

3. On récupère par l'intermédiaire de la LinkTable, l'objet 'connexion' (qui pointe entre autres sur la socket) et on va lui envoyer un message. La méthode associée va écrire sur le stream de sortie la chaîne de caractère correspondant à l'objet sérialisé et mis en forme.

4. Le processus sur le serveur correspondant à la connexion de la station cliente qui va scruter régulièrement le stream d'entrée de la socket va être averti qu'il y a effectivement un paquet.

5. La lecture va commencer et le premier champ qui indique l'identité globale du 'proxy' va permettre de récupérer l'objet partagé correspondant (par la LinkTable).

6. Le reste du message va être remis en forme et envoyé à l'objet partagé. Cela est fait en créant un processus avec une priorité maximale.

7. La méthode est alors exécutée et l'objet résultat renvoyé va être redirigé au 'proxy' sur la station cliente. L'objet résultat est sérialisé et mis en forme, puis il est envoyé au stream de sortie de la connexion.

8. Le processus client va détecter le paquet sur le stream d'entrée de sa connexion et va remettre en forme l'objet.

9. L'exécution de la méthode sur la station cliente peut alors continuer.

Le schéma suivant récapitule les différentes étapes précédentes.

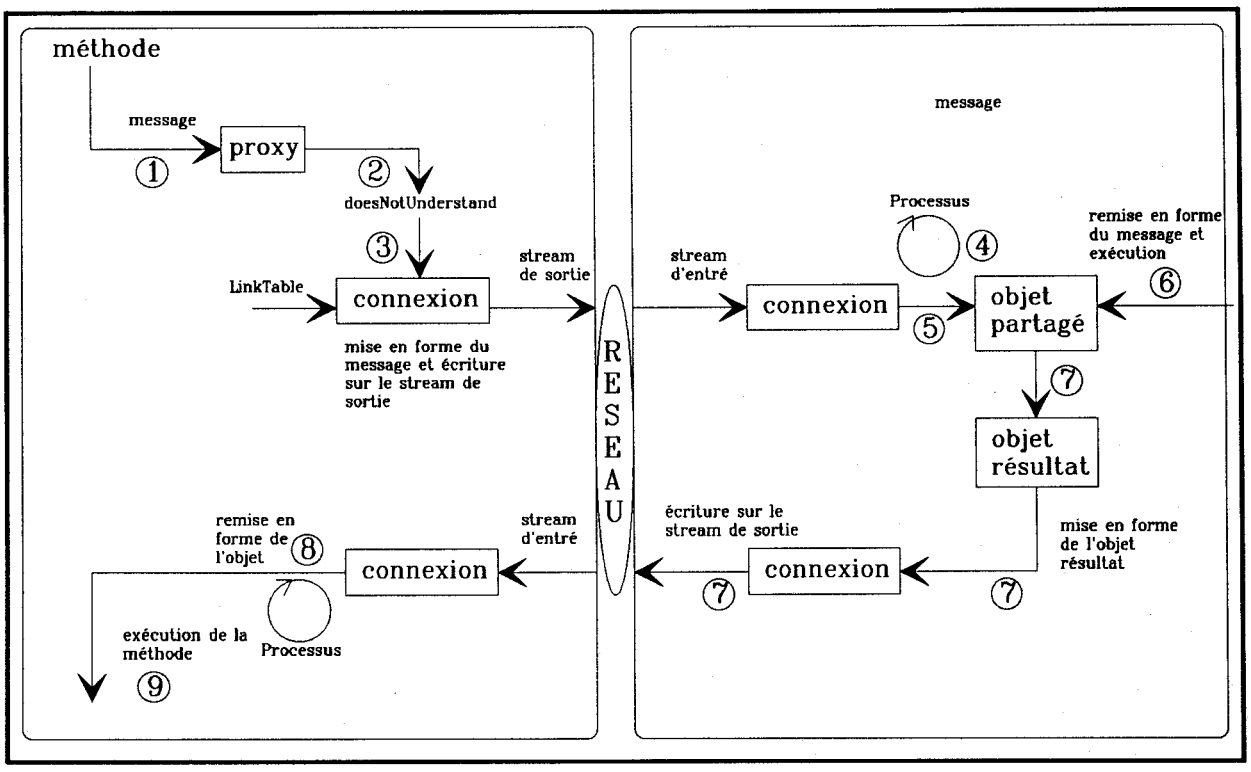


Figure 2.14. *Les différentes étapes lors de l'envoi d'un message à un proxy.*

c) Conclusion

Le système que nous avons mis en place nous permet de partager des objets Smalltalk entre plusieurs stations. Nous pouvons ainsi exécuter des messages dans des images différentes et récupérer un résultat. En développant des applications, nous nous sommes aperçus qu'il serait agréable de disposer d'un outil de débuging d'objets distants. En effet, lorsqu'un problème survient, il ne peut être résolu que sur la station où l'exécution a lieu. La partie suivante présente le débugeur que nous avons implémenté pour résoudre cela.

2.2.2 - Débugeur distant

La richesse de Smalltalk réside dans son environnement de développement. A ce titre, plusieurs outils d'aide sont offerts au programmeur afin de l'assister dans la mise au point de ses applications. L'un des outils les plus utilisés est le débugeur. Lorsque l'exécution d'une méthode échoue, le système crée une fenêtre de notification qui permet d'ouvrir une fenêtre de débuging. Le débugeur va permettre d'explorer l'ensemble des messages exécutés (pile d'exécution) pour remonter jusqu'à celui qui a échoué. Il permet ensuite de visualiser et d'inspecter si besoin l'ensemble des variables utilisées dans une méthode.

L'utilisation d'objets partagés dans des applications coopératives a pour conséquence l'exécution de méthodes distantes. Lorsque l'exécution échoue, la fenêtre de notification est ouverte sur la machine où a lieu l'exécution. Il y a alors obligation de se déplacer pour résoudre le problème. Ce n'est pas trop génant lorsque les stations se trouvent côte à côte, mais cela devient plus problématique lorsqu'elles sont réparties dans des locaux différents.

Pour résoudre cela, nous avons développé un débugeur distant. Il garde fidèlement les mêmes fonctionnalités que le débugeur local, c'est à dire que son interface, son utilisation, et les interactions autorisées (souris et clavier) sont identiques.

La conception de ce débugeur a été faite en tenant compte d'un scénario d'utilisation que nous avons fixé et défini en fonction des habitudes de programmation que nous avions adoptées dans le développement de nos premières applications coopératives. Ce scénario se résume à ceci : le programmeur développe sur le serveur, et lorsqu'une fenêtre de notification est ouverte sur l'une des stations cliente, elle l'est également sur le serveur. L'inverse n'est pas réalisé, car il est pour nous d'aucune utilité.

Il est à noter que le débuging sur la station cliente est toujours possible. Ce débugeur distant n'est actif que si l'utilisateur le souhaite. Il n'y a alors qu'une phase d'initialisation à effectuer pour le mettre en oeuvre. Du point de vue interface, ce qui le distingue d'un débugeur classique, c'est le mot "distant" ajouté dans le titre de la fenêtre.

La figure 2.15. illustre un exemple de déclenchement du débugeur. Sur le serveur, du code Smalltalk est exécuté et fait appel à un moment à un objet distant. Le message lui est envoyé. Un problème survient dans l'exécution du message. Les fenêtres de débuging locales et distantes sont alors ouvertes. Le problème peut alors être traité à distance.

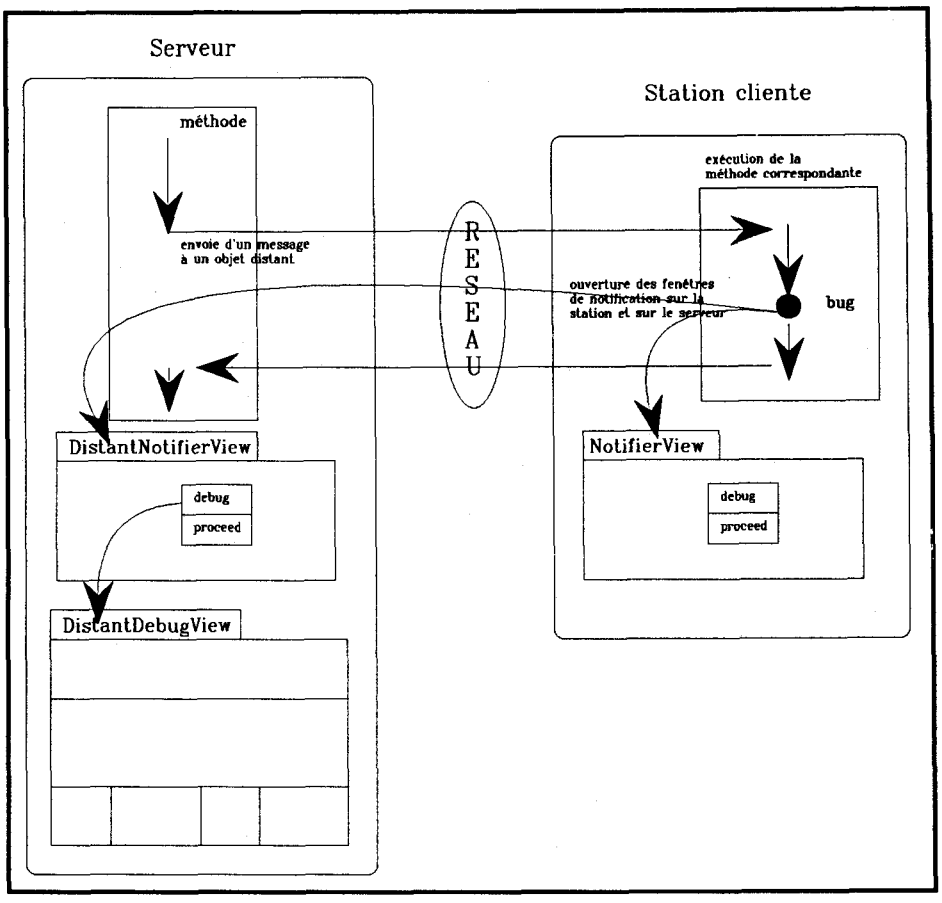


Figure 2.15. *Exemple de déclenchement du debugeur distant. Une méthode distante n'étant pas correcte à déclencher l'ouverture d'une fenêtre de notification sur la station locale et sur le serveur.*

La conclusion que nous tirons de cette étude est la suivante. Il est toujours délicat et généralement difficile, en terme d'implémentation, de mettre au point ce type d'outils. Cependant, leur disponibilité permet un travail beaucoup plus efficace. Il faut donc les développer en parallèle avec le système proprement dit.

Nous n'avons pas décrit l'interface de ce débugeur car elle est en tout point identique à celle proposée par Parc-Place. Son utilisation ne nécessite aucune période d'apprentissage pour une personne connaissant l'environnement Smalltalk. Le choix que nous avons fait, de développer les applications à partir du serveur, n'a rien de définitif car il est tout à fait possible de modifier les mécanismes de diffusion pour permettre le travail à partir d'une station cliente. Il suffira de changer quelques paramètres dans la phase de connexion/initialisation.

2.2.3 - Développement de l'interface Socket sur PC [PULE 91]

La version Smalltalk-80, disponible sur PC, ne permettant pas l'accès direct aux ressources du réseau Ethernet, nous a obligé à développer l'interface socket. Le hard et le soft dont nous disposions étaient respectivement :

. une carte 3 com Ethernet,

. le kit de développement PC/TCP de FTP Software et le compilateur High C 1.6 de Metaware imposé par Parc Place.

La figure 2.16 représente l'organisation des différentes couches logicielles sur PC.

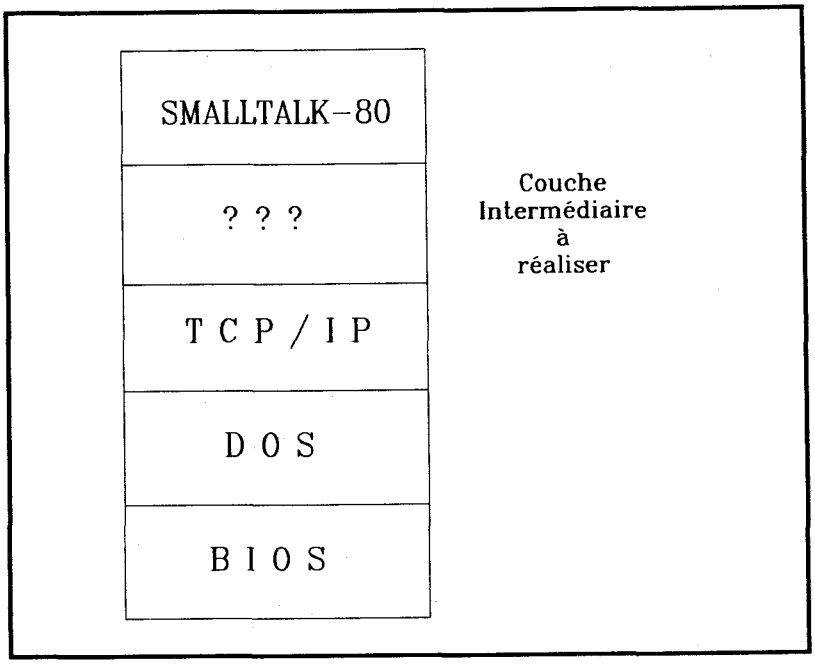


Figure 2.16. *Organisation des couches logicielles.*

Le kit de développement de FTP fournit une bibliothèque de fonctions compatibles au C Microsoft. L'utilisation de ces fonctions, au niveau de Smalltalk, ne

peut se faire que par extension de la machine virtuelle en créant de nouvelles primitives à l'aide du compilateur C Metaware. Faire appel aux fonctions FTP, directement dans le source des primitives n'est pas possible car les différentes bibliothèques ne sont pas compatibles. La solution que nous avons choisie est la suivante :

Il s'agit de réaliser deux unités logicielles, l'une en High C, et l'autre en C Microsoft ; le module en High C faisant appel à des fonctions qui seront effectivement activées dans un second module en C Microsoft. Lors d'un appel de fonction de communication en High C, celui-ci donne la main au module en C Microsoft qui réalisera réellement l'opération.

- avantages : le module en C Microsoft est indépendant, il peut être utilisé avec d'autres langages si son mode d'activation est bien conçu.

- inconvénients : les modules C Microsoft et Smalltalk doivent pouvoir cohabiter en mémoire ; il faut être capable d'activer le module en C Microsoft à partir du module High C, et savoir échanger des informations entre les deux modules, l'échange et l'activation devant être rapides.

La figure 2.17. illustre cette présentation :

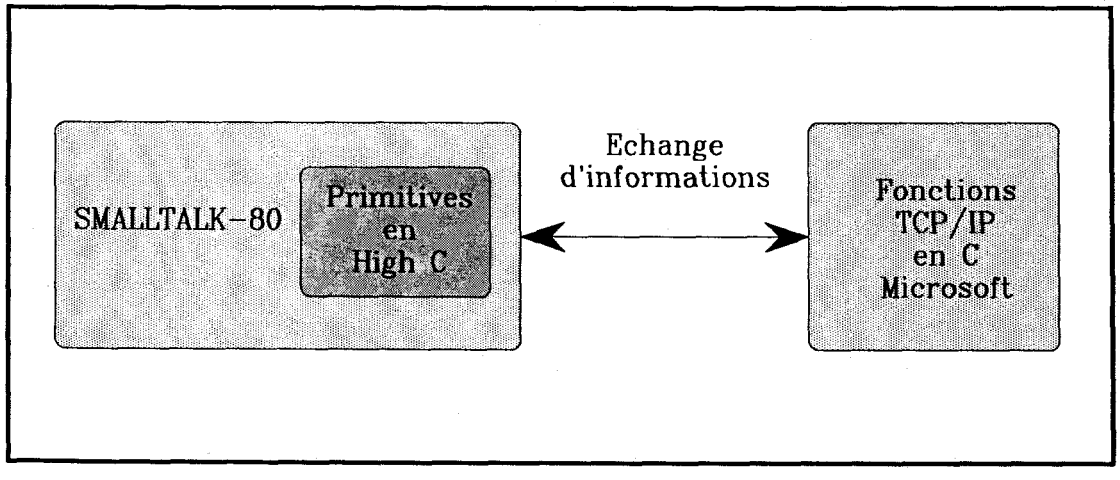


Figure 2.17. *Séparation des primitives et des fonctions TCP/IP.*

Conclusion : Nous avons passé sous silence les difficultés réelles pour intégrer des mécanismes de sockets à Smalltalk sous MS DOS. L'un de ces problèmes était la difficulté d'échanges de données entre mode protégé et mode réel. En effet, le compilateur High C de Metaware travaille en mode protégé, c'est-à-dire qu'il permet aux processeurs 80286, 80386 et 80486 de faire du "multi-tâches", de gérer une mémoire virtuelle et plusieurs Mo de mémoire centrale. Par contre le C Microsoft fonctionne en

mode réel : il ne permet ni le "multi-tâches", ni la gestion de plus d'un Mo de mémoire. Un autre problème délicat à régler a été la réalisation du programme résident qui est composé de deux parties : une partie en C Microsoft et une partie en assembleur. Ces difficultés étaient dues au fait que le "Dos" n'est pas réentrant. Aprés avoir résolu ces problèmes, nous avons maintenant le système de partage d'objets sous PC. Nous pouvons partager, de la même façon que de stations SUN à stations SUN, des objets et des messages entre un PC et une station SUN.

2.3 - Développement d'applications coopératives : une vue générale

2.3.1 - Interface Homme/Machine

L'informatique a considérablement évolué cette dernière décennie avec l'apparition de stations de travail de plus en plus performantes. Leur puissance a permis la mise en place de systèmes multi-fenêtres sophistiqués. Paradoxalement, on a constaté que les techniques de communication homme/machine n'avaient par contre pas évolué de façon aussi importante. Pour de nombreuses applications, notamment en matière de gestion de système, seul le mode de dialogue textuel existait.

On s'est également aperçu que le succès d'un produit était lié en grande partie à son interface utilisateur. Fort de ce constat, la conception et la réalisation d'interfaces homme/machine est en pleine évolution et constitue un domaine de recherche très actif [COUT 90] [KARS 87]. Les compétences associées à un travail collectif de deux disciplines sont nécessaires pour aboutir avec succès à la réalisation d'interfaces adaptées aux besoins des utilisateurs, à savoir : l'informatique et les sciences cognitives.

Afin d'aider les programmeurs à concevoir et à réaliser des interfaces conviviales, des boîtes à outils (ou toolbox) ont été proposées. Nous trouvons par exemple XToolkit [XT] construite au-dessus de X WindowSystem [SCHE 86] ainsi que la boîte à outils du Macintosh. Ces boîtes à outils contiennent des bibliothèques de fonctions permettant de résoudre certains problèmes. On y trouve généralement la possibilité de gérer des fenêtres, des menus, des icônes, des primitives graphiques et des boîtes de dialogues.

L'un des problèmes avec ces boîtes à outils est que les services offerts restent de bas niveau, et que la mise au point d'une interface demeure délicate. Un développement important de codes est nécessaire pour obtenir l'interface désirée, sans pour autant aboutir à quelque chose d'uniforme ou de consistant.

Il existe cependant certains produits qui proposent des améliorations par rapport à ces boîtes à outils. Ils sont conçus à partir de celles-ci avec une approche un peu plus objet permettant ainsi un niveau d'abstraction plus important et un développement plus uniforme. On peut citer en exemple MacApp [SCHU 86] construite à partir de la boîte à outils du Macintosh. Avec ces produits, le programmeur dispose d'un ensemble d'objets qui seront à personnaliser en fonction de l'application interactive.

Il semble que la tendance actuelle soit de fournir des outils génériques développés selon le modèle U.I.M.S. (User Interface Management System) [KASI 82], l'objectif

étant de séparer l'application, des problèmes de gestion d'interfaces. Les avantages de ce type d'architecture sont de permettre une réutilisabilité plus facile des composants logiciels et une plus grande cohérence entre les différentes applications.

Dans la partie suivante, nous présenterons les caractéristiques du modèle M.V.C. qui est l'U.I.M.S. de Smalltalk.

2.3.2 - Présentation de M.V.C.

Afin d'uniformiser le développement d'applications interactives Smalltalk-80, un concept d'architecture appelé triade M.V.C. (Modèle Vue Contrôleur) [GORO 83] [DUGE 90] a été proposé par les concepteurs de Smalltalk-80. Les trois composants constituant cette triade ont chacun un rôle bien particulier.

Modèle : il représente les données manipulées dans l'application dont une représentation sera affichée dans la vue. Les méthodes permettant le traitement de cette information sont également implémentées dans ce modèle.

Vue : définie à partir d'une hiérarchie de vues et de sous-vues, elle représente l'interface visuelle du modèle pour l'utilisateur. Différentes représentations ou "points de vue" sont possibles pour afficher l'information contenue dans le modèle.

Contrôleur : il permet à l'utilisateur, lorsqu'il est actif, d'interagir sur les données. Le contrôleur est chargé de détecter les interactions souris et clavier afin d'exécuter les méthodes correspondantes.

Pour fonctionner correctement, des relations sont nécessaires entre les trois composants de la triade M.V.C. Une vue est instance de la classe View ou d'une de ses sous-classes. De même, un contrôleur est instance de la classe Controller ou d'une de ses sous-classes. La classe View possède des variables d'instance appelées "controller" et "model", qui pointent respectivement sur le contrôleur et le modèle de la vue. La classe Controller possède aussi des variables d'instance appelées view et model qui pointent respectivement sur la vue et le modèle du contrôleur.

Généralement, on utilise comme modèle d'un couple vue/contrôleur, l'instance d'une sous-classe de Model pour la raison suivante : la notion de dépendance a été redéfinie dans cette classe, permettant ainsi d'avoir un pointeur sur l'ensemble des vues utilisant le modèle. Cette variable est une liste appelée "dependents". Toutefois, il n'est pas obligatoire que le modèle soit sous-classe de la classe Model car, en Smalltalk, on a

la possibilité, à partir de la classe Object, de définir des liens de dépendance entre objets. Ceci est réalisé à partir de la variable de classe nommée "DependentFields". La figure 2.18. représente les 3 éléments de la triade avec leurs pointeurs respectifs.

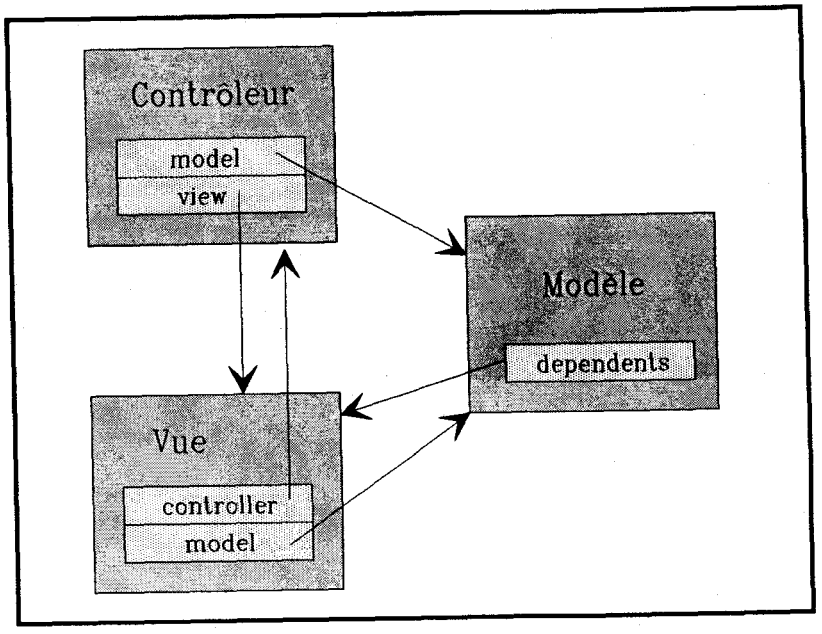


Figure 2.18. *Triade M.V.C. avec les pointeurs de chacun des trois composants.*

Une des motivations qui a conduit à l'architecture de la triade M.V.C. est qu'elle permet l'implémentation d'un ensemble de couples vue/contrôleur spécifiques à un type d'activité qui sont réutilisables pour des applications différentes. En effet, la façon de représenter et de manipuler certaines informations peut être commune à plusieurs applications (exemple : la gestion d'une fenêtre texte). Smalltalk fournit ainsi une bibliothèque de classes vue/contrôleur prédéfinie. Le mot "prédéfinie" correspond au fait que, lorsque l'on utilise un de ces couples, la représentation dans la vue ainsi que la façon dont seront gérées les interactions sont fixées.

Dans Smalltalk, les trois boutons de la souris ont une fonction précise. L'appui sur le bouton gauche de la souris va avoir un résultat différent selon le couple vue/contrôleur. Cela va par exemple, positionner un curseur dans une fenêtre texte et sélectionner un item dans une fenêtre liste, etc. Le bouton du milieu renvoie un menu qui doit être défini par le programmeur au moment de la création de la fenêtre. Un menu général commun à toutes les fenêtres est attaché au bouton de droite.

Une fenêtre est construite à partir d'une hiérarchie arborescente des vues. La vue principale appelée "topView" est une instance de la classe StandardSystemView associée à son contrôleur instance de la classe StandardSystemController. Le choix de ce couple vue/contrôleur est absolument nécessaire pour les raisons suivantes :

. Le rôle du contrôleur de cette vue consiste essentiellement à déterminer dans quelles sous-fenêtres le curseur de la souris se trouve pour lui donner le contrôle (c'est-à-dire, rendre actif le contrôleur de la sous-fenêtre).

. Le bouton de droite de la souris permet de renvoyer un menu commun à l'ensemble des fenêtres. Il s'agit des opérations générales sur celles-ci (opérations de repositionnement, de redimensionnement, de fermeture, de mise sous forme d'icône, etc..).

. Dans Smalltalk-80, l'unique instance de la classe "ControlManager" référencée par la variable globale ScheduledController, va gérer le passage du processeur aux contrôleurs des TopViews. L'une de ces variable d'instance "ScheduledController" va pointer sur une liste de contrôleurs qui seront obligatoirement instances de la classe StandardSystemController ou d'une de ses sous-classes (dans la liste on trouve également le contrôleur associé à l'écran).

Flux d'informations entre les 3 composants :

L'utilisateur va pouvoir agir sur son modèle par l'intermédiaire du contrôleur associé. Les modifications, qui devront entraîner les mises à jour adéquates des vues se feront à partir des interactions souris et clavier, suivant un protocole pré-établi dans le contrôleur. Une partie de la mise à jour est gérée à partir du modèle. Lorsque l'exécution d'une méthode entraîne une modification dans l'affichage, le modèle s'envoye le message "changed" ou "changed: unSymbol" qui aura pour conséquence l'exécution du message "update:" sur l'ensemble des vues dépendant du modèle. L'utilisation du message "changed: unSymbol" permet un réaffichage plus rapide lorsqu'une seule sous-vue est à raffraîchir.

Il est à signaler que pour certaines vues, une partie de la mise à jour des vues est également régie par le contrôleur. Par exemple, pour des fenêtres listes, la mise en vidéo inverse d'un item sélectionné est gérée par le contrôleur. Le modèle en est quand même averti, car cela peut avoir des conséquences pour d'autres vues (ex : mise à jour d'un nouveau texte).

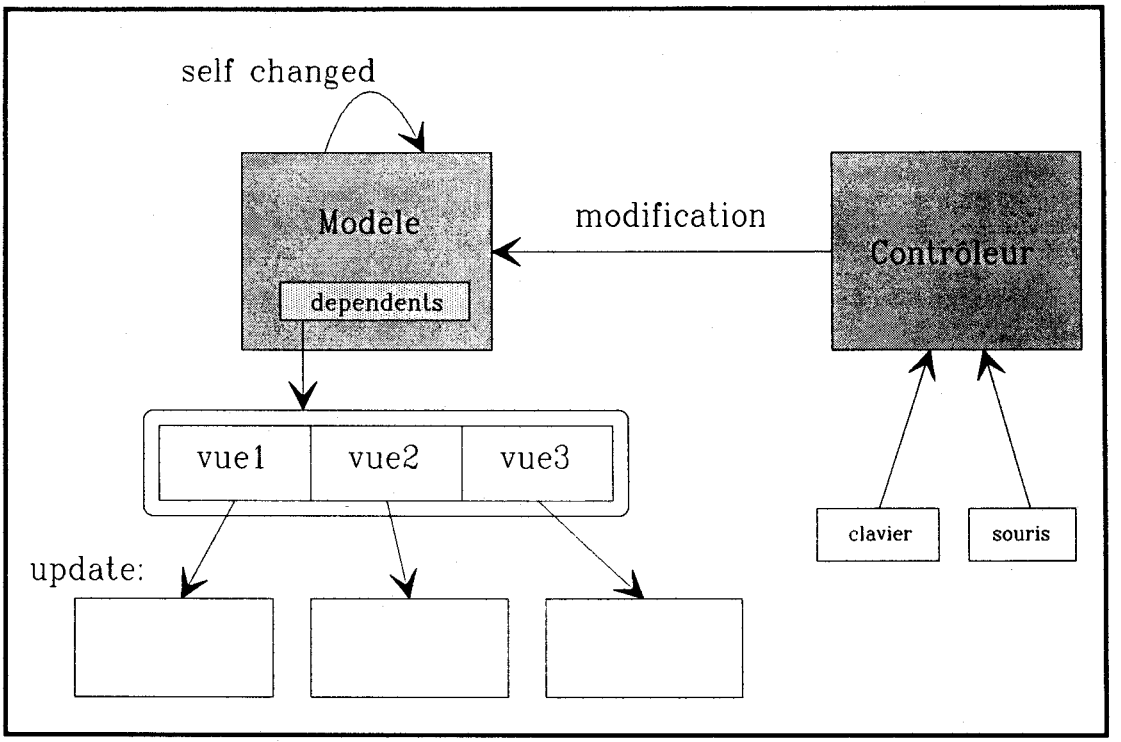


Figure 2.19. *Un exemple d'échanges d'informations entre les trois composants de la triade. Le contrôleur détecte la modification et prévient le modèle. Ce dernier effectue le traitement et avertit alors ses vues dépendantes qu'il vient de changer.*

L'exécution de la méthode modification va déclencher le message "changed" qui entraînera à son tour, l'exécution des messages "updates:" sur les vues.

Conclusion

Le schéma de développement d'applications interactives en Smalltalk-80 appelé M.V.C. permet une implémentation très rapide et efficace. Ceci est lié au fait qu'une bibliothèque de couples vue/contrôleur est fournie en standard. L'utilisation de ces couples impose cependant des règles strictes d'implémentation au niveau des méthodes dans la classe du modèle. Un des aspects souvent critiqué de la triade M.V.C., dû en grande partie au caractère générique de la boîte à outils, est que, contrairement à certains modèles proposés [COUT 90] pour des applications interactives, le rôle des trois éléments n'est pas clairement défini. Des critiques du système M.V.C. ont été faites par Yen Ping Shang [SHAN 89] [SHAN 90] [SHAN 91] qui a du le modifier pour des besoins relatifs à ses travaux sur les interfaces à manipulation directe.

2.4 - Architecture de la structure d'accueil pour des applications coopératives

2.4.1 - Association d'un gestionnaire de conférence et des applications

L'objectif principal de notre projet est d'étudier et de concevoir un système qui soit une structure d'accueil pour des applications coopératives différentes. Afin de satisfaire cela, nous avons scindé le système en deux composants principaux :

. le système de gestion de conférence

. la partie applicative.

Le rôle du gestionnaire de conférence est multiple.

Il doit gérer l'espace public (application) ; c'est à dire attribuer le contrôle en fonction de son "algorithme de contrôle".

Il permet de connaître à tout instant, l'état de la conférence.

Il fournit aux participants différents outils (télépointeurs, vote).

Le point le plus délicat concerne la définition de l'algorithme de contrôle de la conférence. Comme nous l'avons déjà présenté, celui que nous avons implémenté obéit aux critères suivants. Dans une session, il y a un maître de conférence avec des pouvoirs supérieurs à ceux des autres utilisateurs. Ces derniers ont tous les mêmes droits et la demande du contrôle leur est accordée en fonction de l'ordre des requêtes.

Quant au maître de conférence, il peut reprendre le contrôle à tout instant et peut l'attribuer à un participant de son choix. Lorsqu'une personne a le contrôle, elle est seule à pouvoir modifier l'espace public. Cependant, nous verrons que certaines opérations restent accessibles pour les autres. Cette dépendance de la partie applicative par rapport au gestionnaire de conférence impose, un échange d'informations entre eux.

L'architecture générale que nous allons présenter, devait respecter certaines contraintes fixées au départ. La principale était liée à des contraintes réseaux. Il s'agit dans un futur proche, d'utiliser et de valider le système sur un réseau de type R.N.I.S., sur lequel il ne sera pas possible d'avoir les connexions directes entre toutes les stations. Il s'agit d'un réseau étoilé avec le serveur au centre.

La figure suivante représente les différentes couches de ce système.

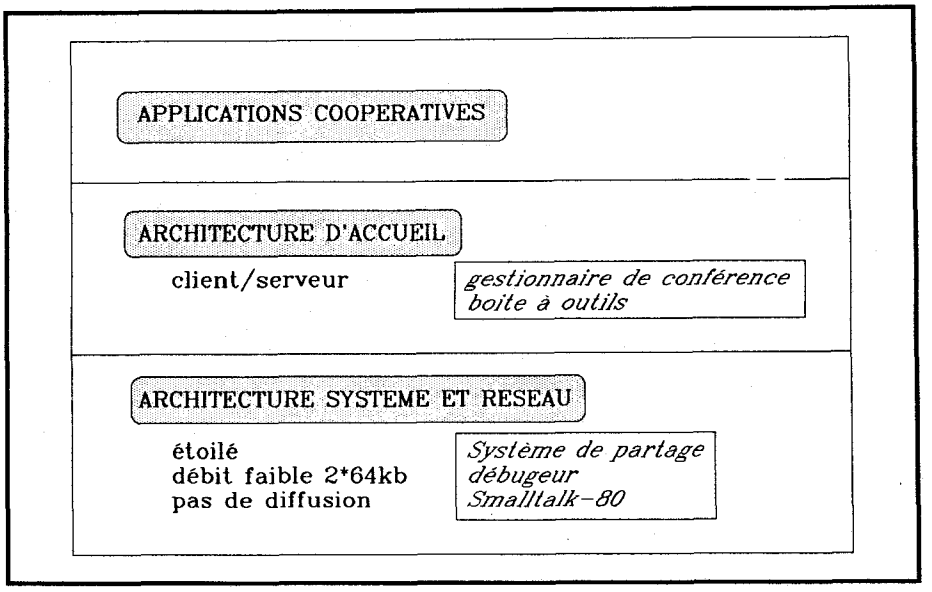


Figure 2.20. *Représentation de l'organisation des différentes couches de notre système*

Dans l'environnement Smalltalk, le gestionnaire de conférence et la partie applicative sont associés à des fenêtres différentes. Elles permettent de visualiser et d'agir sur les données. La figure 2.21., montre les deux fenêtres avec le flux d'informations échangées entre elles. Nous détaillerons ce flux plus en détail dans les parties suivantes.

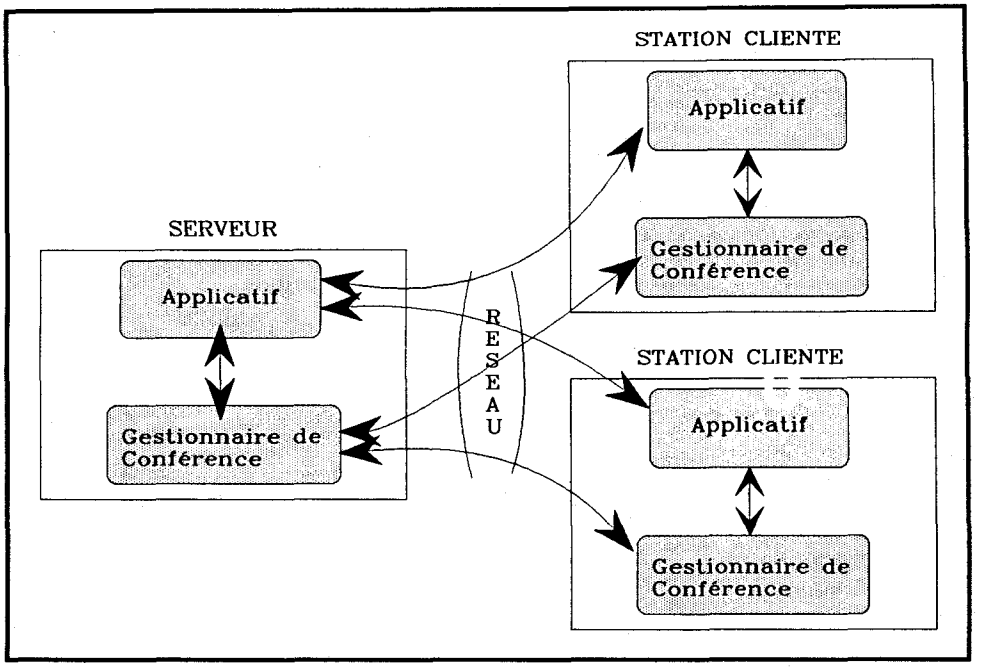


Figure 2.21. *Détail du flux d'informations entre les stations ainsi qu'entre la partie applicative et le gestionnaire de conférence.*

A partir des quelques applications coopératives que nous avons analysées et implémentées pour certaines, nous avons pu les classer selon trois catégories.

. La première catégorie correspond à des applications qui nécessitent des espaces de travail de type W.Y.S.I.W.I.S. (W.Y.S.I.W.I.S. relâché ou non). En effet, toutes les personnes du groupe ont la même tâche à accomplir et l'utilisation d'un support partagé commun est capitale. La prise de décision est collective.

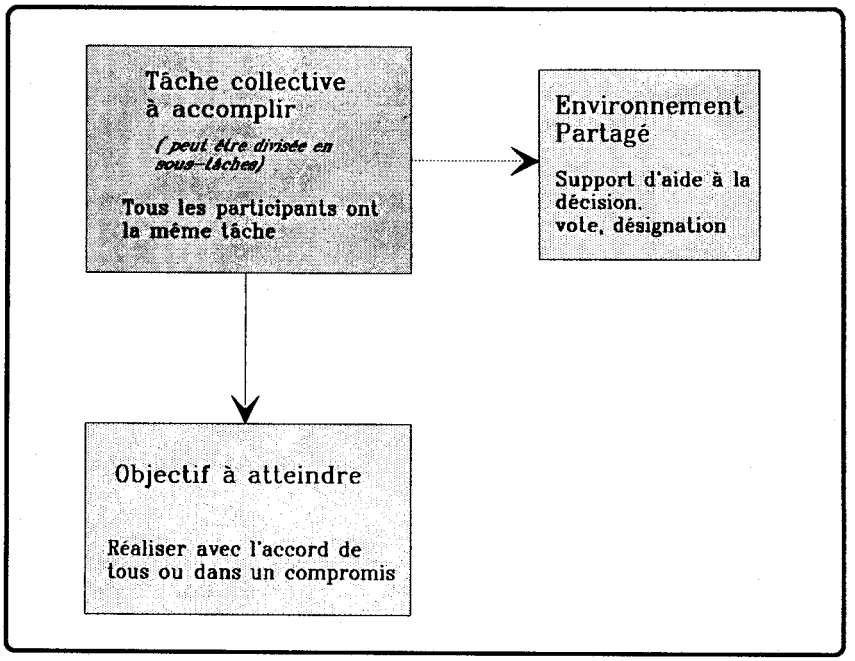


Figure 2.22. *Tâche collective de type W.Y.S.I.W.I.S.. Chaque participant a exactement la même tâche à remplir que ses collègues.*

. La deuxième catégorie correspond à des applications qui nécessitent uniquement le partage de certaines données. Les individus ont chacun une tâche personnelle à accomplir avec une prise de décision individuelle. Cependant, cette dernière devra être accomplie en tenant compte du contexte général et de l'objectif collectif fixé.

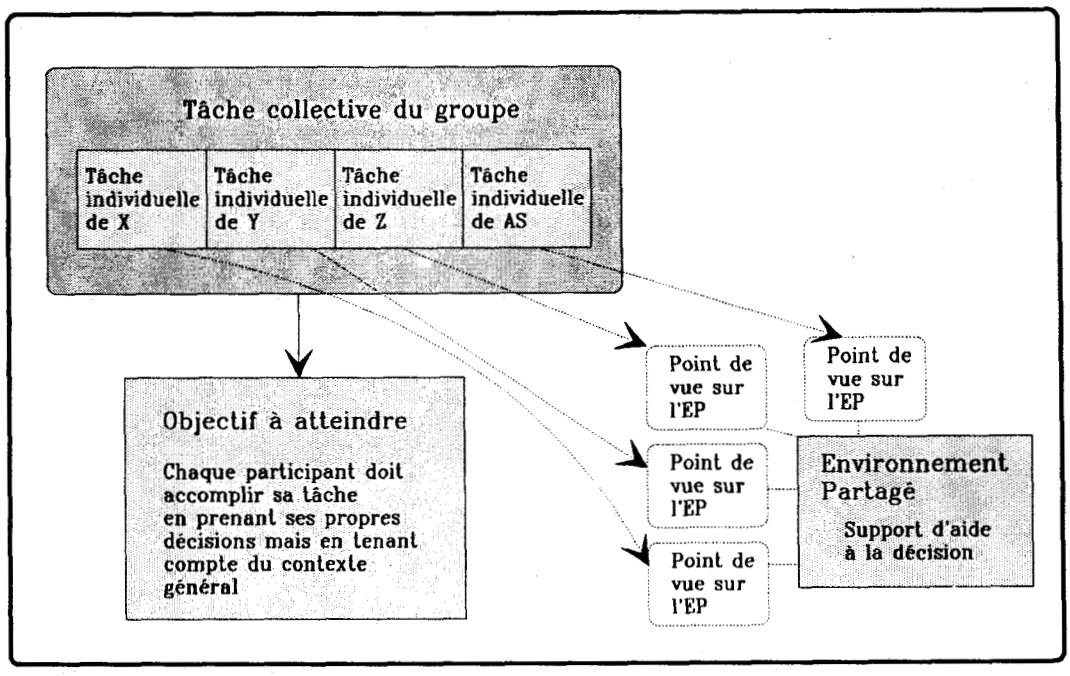


Figure 2.23. *Tâche collective non W.Y.S.I.W.I.S.. Chaque participant a une tâche particulière à remplir. Pour prendre leurs décisions, les participants ont encore accès à l'environnement partagé mais d'une façon différente à la précédente. Un point de vue propre à chacun de ces données partagées peut être suffisant.*

. Dans la troisième catégorie, on trouve des applications hybrides qui sont une combinaison des deux catégories précédentes.

La première catégorie est celle qui nous intéresse plus particulièrement et pour laquelle nous avons conçu le gestionnaire de conférence.

La seconde catégorie d'applications, possède des contraintes beaucoup moins importantes. En effet, tout en ayant besoin d'une rétroaction sur l'activité de l'ensemble des participants, ces applications ne s'appuient pas sur le gestionnaire de conférence (en particulier pour le contrôle de l'espace public). Les problèmes de concurrence sont réglés classiquement au niveau des données. Une présentation en sera quand même faite dans le chapître suivant car l'application que nous avons implémentée possède des phases qui correspondent aux deux premières catégories.

Nous avons essayé d'analyser et de représenter plus finement les activités liées à des applications de type W.Y.S.I.W.I.S. et non W.Y.S.I.W.I.S.. Le Grafcet est un moyen efficace de les représenter.

application W.Y.S.I.W.I.S.

Une tâche globale peut être divisée en un nombre N de sous-tâches qui devront

être résolues successivement pour atteindre l'objectif. Chacune des décisions qui sont prises doivent être approuvées par l'ensemble des participants ou tranchées par l'animateur s'il y a désaccord complet.

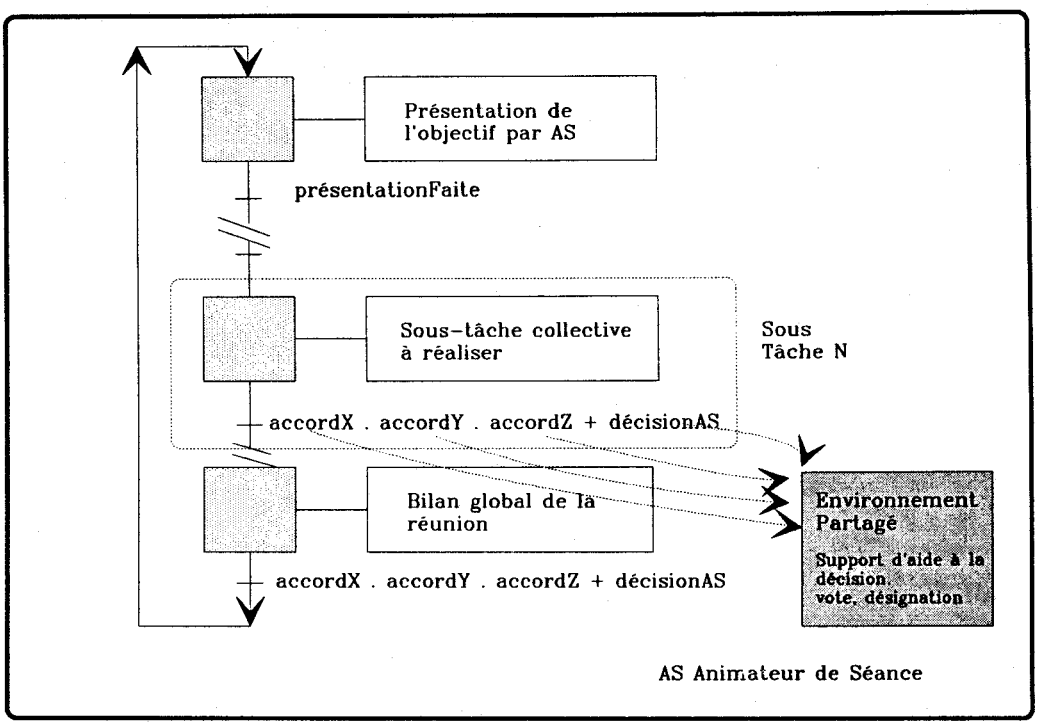


Figure 2.24. *Représentation sous forme de Grafcet des activités dans une application W.Y.S.I.W.I.S.*

Nous nous apercevons sur ce Grafcet que la transition à valider correspond au consensus général. C'est à ce niveau que notre gestionnaire de conférence doit servir pour aider les participants à parvenir à une solution. Nous voyons donc ici qu'il est possible de proposer différents types de protocoles qui différeront en fonction de l'application, des rôles et des droits de chaque participant.

Application non W.Y.S.I.W.I.S.

Dans ce type d'application, chaque participant à un rôle particulier à tenir. Les décisions qu'il prend sont personnelles, mais à partir d'un contexte général sur lequel il dispose d'un point de vue. Dans ces applications, il est nécessaire de faire des phases de synchronisation pour permettre à chacun de faire un point global sur les activités de chacun. On s'aperçoit malheureusement qu'il n'est pas possible de proposer une gestion de conférence comme nous l'avons fait précédemment, car le rôle de chacun est ici lié à l'application.

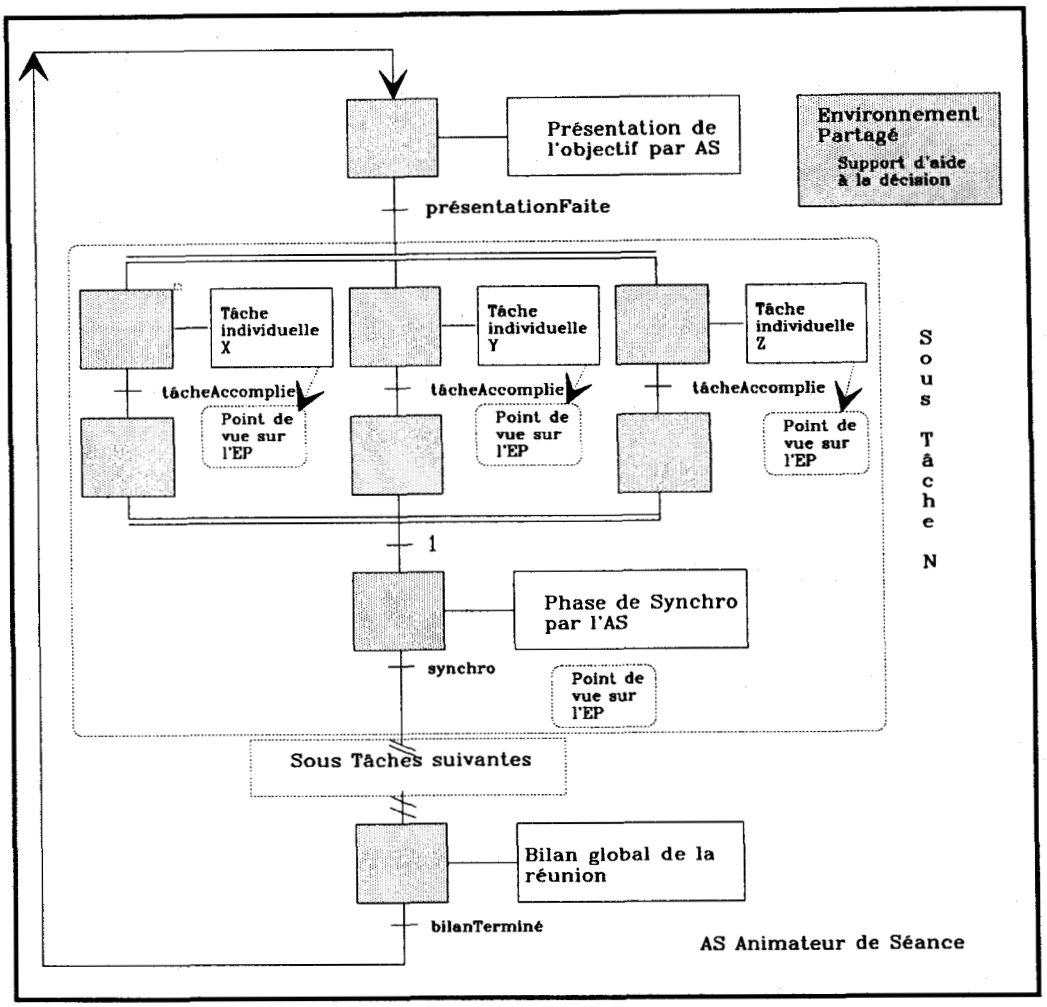


Figure 2.25. *Représentation sous forme de Grafcet des activités dans une application non W.Y.S.I.W.I.S.*

Ces deux représentations sont tout à fait générales et il est tout à fait possible de ne pas avoir d'étapes de synchronisation au cours du déroulement. Toutefois, on les retrouve en début et en fin de séance.

Cette représentation permet de mettre en évidence les différences fondamentales entre les deux types d'applications. Cela nous a également permis de montrer à quel niveau le gestionnaire de conférence se situe.

Nous allons maintenant exposer le choix que nous avons fait pour notre architecture d'accueil (gestionnaire de conférence et les applications coopératives).

2.4.2 - Le Gestionnaire de Conférence

Le gestionnaire de conférence est comparable aux applications coopératives qui ne

nécessitent pas d'espace W.Y.S.I.W.I.S.. Néanmoins, certaines données, comme la liste des participants ou le nom de la personne qui contrôle l'espace public, doivent être partagées afin qu'une cohérence soit établie sur l'ensemble du système. Sur chaque station, un modèle différent est implémenté. Les données partagées sur chaque station sont dupliquées pour les raisons suivantes.

. L'implémentation d'une application est beaucoup plus simple lorsque les données sont locales et directement accessibles. Le développement se fait d'une façon quasiment classique (mono-utilisateur) et seules les méthodes modifiant ces données sont à réécrire pour diffuser l'information aux autres stations. Ces données partagées doivent cependant recevoir un mécanisme de protection contre l'accès concurrent.

. Les nombres de données à partager dans les applications que nous avons analysées sont relativement faibles.

. Les temps de réponse sont plus courts. Nous avons minimisé les appels de procédures distantes. L'utilisation qui est faite de ces données est nettement plus fréquente que les changements de leurs valeurs.

La figure suivante représente l'implémentation des gestionnaires de conférence sur les stations.

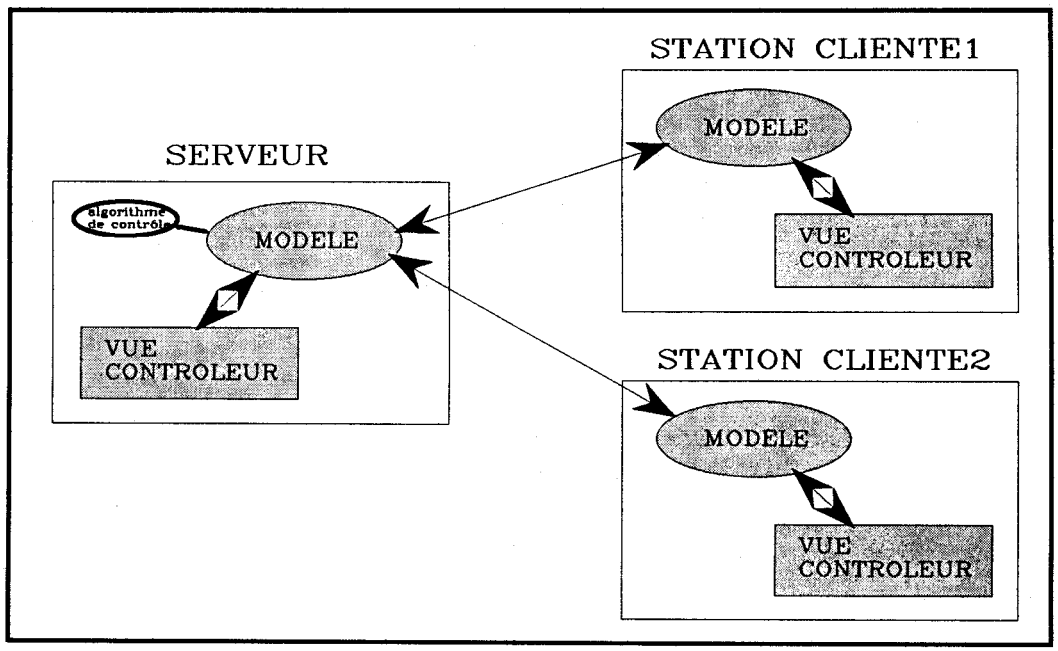


figure 2.26. *Architecture des gestionnaires de conférence. Chaque station possède chacun un modèle mais qui communique avec les autres pour assurer une cohérence globale.*

La flèche 1 symbolise l'échange d'informations entre les différents modèles. Cet échange ne se fait pas directement entre modèles, mais par l'intermédiaire d'objets particuliers appelés "exécuteur" que nous présenterons en détail un peu plus loin.

Rôle du Gestionnaire de conférence

Le rôle du gestionnaire de conférence est multiple.

. Il assure la gestion de l'espace public (attribution du contrôle aux utilisateurs).

. Il fournit des outils (différents télépointeurs, mécanisme de vôte).

. Il permet de connaître à tout instant, l'état de la conférence (participants présents, nom du participant qui contrôle, les absences, etc..)

. Il offre de nombreuses facilités au programmeur pour implémenter des applications (Facilités pour diffuser des messages aux objets "éxécuteur" des différentes stations, pour accéder au modèle de la partie applicative par des pointeurs, etc..).

Algorithme de contrôle de la conférence

Il est unique et se trouve sur le serveur. Son choix dépend fortement de l'application et en particulier du type d'activité qui va s'y dérouler.

Une dimension sur laquelle nous devons réfléchir est la granularité des objets sur lesquels le contrôle s'opére. Actuellement, celui-ci est accordé sur l'ensemble de l'espace public. Le partionner permettrait d'introduire du parallélisme. Diminuer la taille du grain, n'est pas le problème le plus compliqué à résoudre. Différentes stratégies sans réelles difficultés techniques sont envisageables.

Une fenêtre Smalltalk est généralement organisée dans une hiérarchie de sous-fenêtres dont le contrôle pourra être associé à diverses personnes. S'il s'agit d'une application à manipulation directe, chaque objet graphique pourra être la propriété momentanée d'un individu. La réelle difficulté provient de la gestion dynamique du contrôle entre les différents utilisateurs. Plus le nombre d'objets sera conséquent plus la gestion risque d'être lourde. Actuellement, le contrôle est rendu par l'utilisateur lorsqu'il le décide. A part le maître de conférence, les autres utilisateurs ne peuvent le reprendre. Ils ont toutefois la possibilité de converser oralement pour négocier les temps de contrôle.

Dans notre système, d'un simple coup d'oeil, les participants savent qui a le

contrôle, qui est le prochain à l'avoir, ceux en attente. Avec un nombre d'objets important, cette facilité ne sera pas aisée à obtenir. La confusion et le temps passé à régler les problèmes risquent d'être conséquents. Il nous faudra trouver des interfaces adaptées à ce nouveau type de gestion.

Nous avons envisagé que l'attribution pouvait ne pas être explicite mais implicite. L'utilisateur peut travailler sur tous les objets "libres" et s'attribue alors le contrôle. Après un time-out (temps d'inactivité) la perte du contrôle est automatique. Des grisés de couleurs ont été utilisés dans des éditeurs de texte coopératifs afin de signaler la présence d'un utilisateur sur une partie de texte [ELGI 91].

Les différentes idées que l'on peut avoir sur la façon de gérer l'espace public doivent être validées par une expérimentation en situation réelle. La première chose à faire est d'analyser très finement les différentes activités qui se déroulent lors d'une séance de travail. Il ne faut pas oublier que si l'on arrive sur une autorisation d'activité parallèle, le problème de la rétroaction (feedback) entre utilisateurs va augmenter (Le Cognoter de Colab offre une phase pendant laquelle chacun peut placer ses idées. En l'expérimentant, les concepteurs se sont rendus compte qu'il fallait régulièrement arrêter pour faire le point pour l'ensemble du groupe).

2.4.3 - Choix d'une Architecture d'accueil distribuée

a) Architecture centraliée

La figure suivante illustre l'architecture choisie pour l'implémentation.

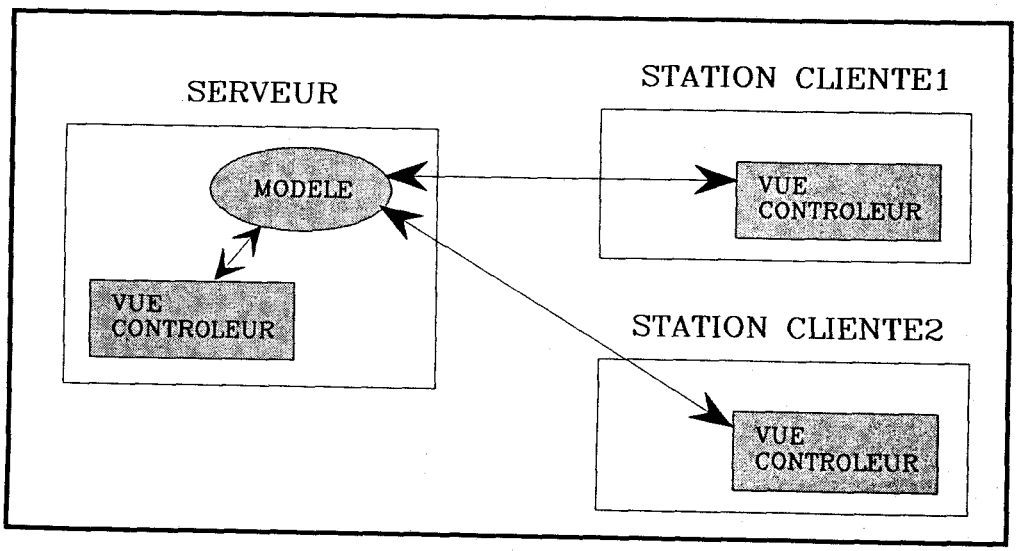


Figure 2.27. *Architecture centralisée*

L'architecture que nous avons choisie est "plutôt" centralisée. Nous expliquerons ultérieurement le mot "plutôt" car tout en ayant un modèle centralisé un partie de code est également dupliqué afin de réduire le flux d'information et d'augmenter les temps de réponse pour les mises à jour des interfaces. Globalement nous avons une architecture hybride.

En plus des avantages et inconvénients des deux architectures, les points positifs d'une architecture centralisée, qui nous ont fait opter pour elle, sont plus particulièrement liés à l'environnement de programmation Smalltalk-80. Comme nous l'avons vu, les applications interactives sont construites à partir de la triade M.V.C.. Dans celle-ci, nous centralisons uniquement le modèle de l'application, les couples vue/contrôleur restant locaux à la station. L'avantage de cela est de disposer d'un moyen qui nous permettra de répartir du code pour des tâches locales faisant ainsi diminuer le flux d'information entre les stations (Certaines de ces opérations sont par exemple la possibilité de "scroller" ou de déplacer une fenêtre).

Réaliser un modèle centralisé, se fait par l'intermédiaire du système de partage d'objets. Le modèle central (nous appellerons ainsi le modèle implémenté sur le serveur) est déclaré comme objet partagé. Des objets mandataires 'modeleProxy' peuvent alors être initialisés sur chaque station cliente. Ces 'modeleProxy' ne sont qu'une référence distante du modèle. Ils ne font rien d'autre que de renvoyer les messages qu'ils reçoivent. L'architecture de l'application peut être schématisée de la façon suivante :

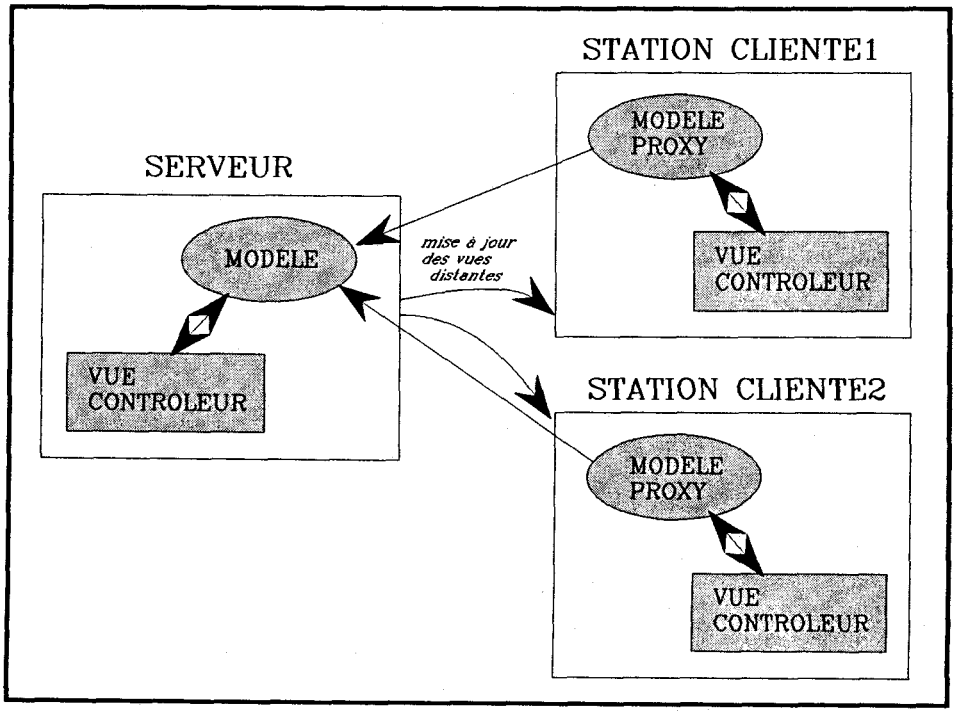


Figure 2.28. *Schéma de l'architecture avec un modèle centralisé. Sur les stations clientes, le "modèleProxy" est l'objet mandataire correspondant au modèle.*

b) Mise à jour des vues distantes

Le mécanisme de dépendance de la triade M.V.C. permet au modèle de mettre à jour automatiquement les vues lorsqu'il a été modifié. Cela est fait pour les vues locales mais n'est pas réalisé pour les vues distantes. Ce mécanisme doit donc être géré explicitement par le programmeur. Il s'agit d'étendre la notion de dépendance aux vues distantes. Pour accomplir cela, deux solutions sont envisageables.

. Gérer la mise à jour à partir du modèle,

. Gérer la mise à jour à partir des couples vue/contrôleur.

1) Gestion à partir du modèle

Cette solution a pour avantage de conserver les couples vue/contrôleur standards offerts par Smalltalk-80. L'ensemble des méthodes du modèle qui provoque une modification des vues ou qui est invoqué lorsqu'une vue a été modifiée doit être réécrit afin d'éxécuter les mises à jour des vues distantes.

L'inconvénient de cette approche est qu'il n'y a pas de caractère générique ; tout doit être écrit pour chaque nouvelle application. De plus, une bonne connaissance du système de gestion et des mécanismes de diffusion est nécessaire. Cette première solution, qui semble plus facile au prime abord, devient très vite compliquée dans certaines situations. En particulier, lors d'appel menu, il faut renvoyer le menu correspondant au statut du demandeur (contrôle ou non contrôle).

Le dernier aspect défavorable de cette solution est que le rôle des composants de la triade est faussé. En effet, ce n'est pas au modèle de gérer les mises à jour des vues mais aux vues elles-même.

2) Gestion à partir des couples vue/contrôleur

La gestion de la mise à jour des vues à partir des couples vue/contrôleur nécessite le développement de nouveaux couples spécifiques. L'avantage de cette solution est que le modèle se développe de façon classique comme dans une application mono-utilisateur. C'est cette solution que nous avons choisie en nous redéfinissant de nouvelles classes de vues et de contrôleurs. Un de ses aspects positifs est qu'elle permet une réutilisabilité des composants logiciels pour d'autres applications. Cette triade constituée de nos nouveaux couples vue/contrôleur est maintenant appelée M.V.C. Coopératif. Une présentation détaillée en est faite un peu plus loin.

La figure 2.29 illustre cette architecture.

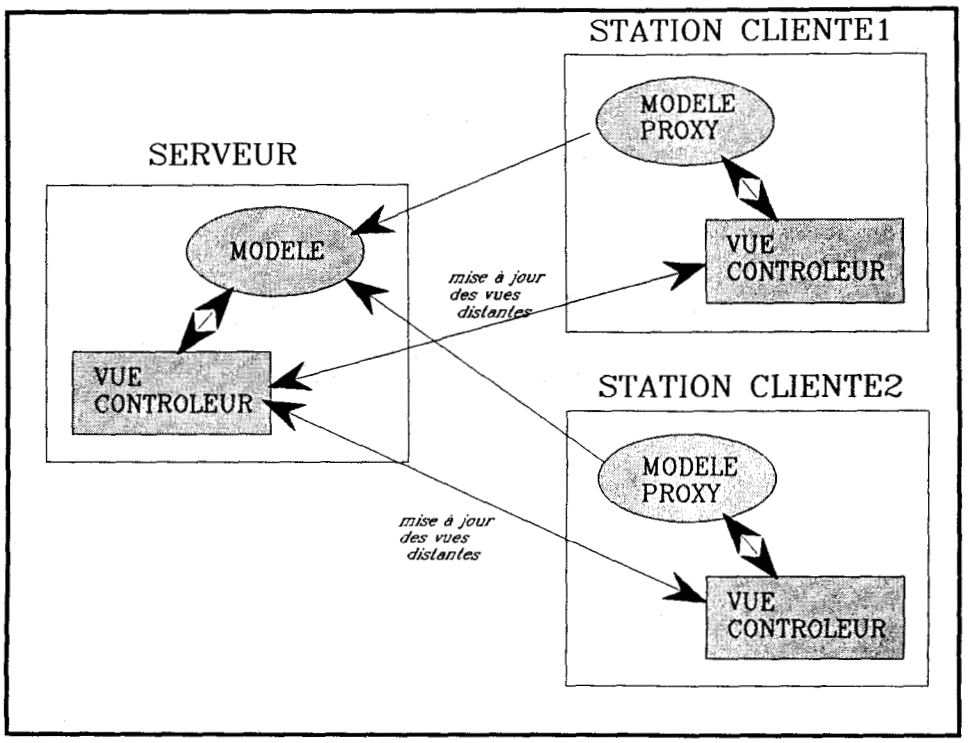


Figure 2.29. *Schéma de l'architecture avec un modèle centralisé. La mise à jour des vues distantes est gérée par des nouveaux couples vue/contrôleur spécifiques.*

Pour effectuer ces mises à jour, nous avons besoin d'envoyer des messages à des objets partagés. La première idée est d'utiliser les couples vue/contrôleur. L'inconvénient de cela est qu'il faudrait tous les partager et définir dans les classes des méthodes qui n'y ont pas leur place. Nous avons opté pour le partage d'objets spéciaux que nous appelons "éxécuteur".

c) Objet "éxécuteur"

Un objet exécuteur est un objet partagé, accessible à partir d'autres stations par l'intermédiaire de son objet mandataire.

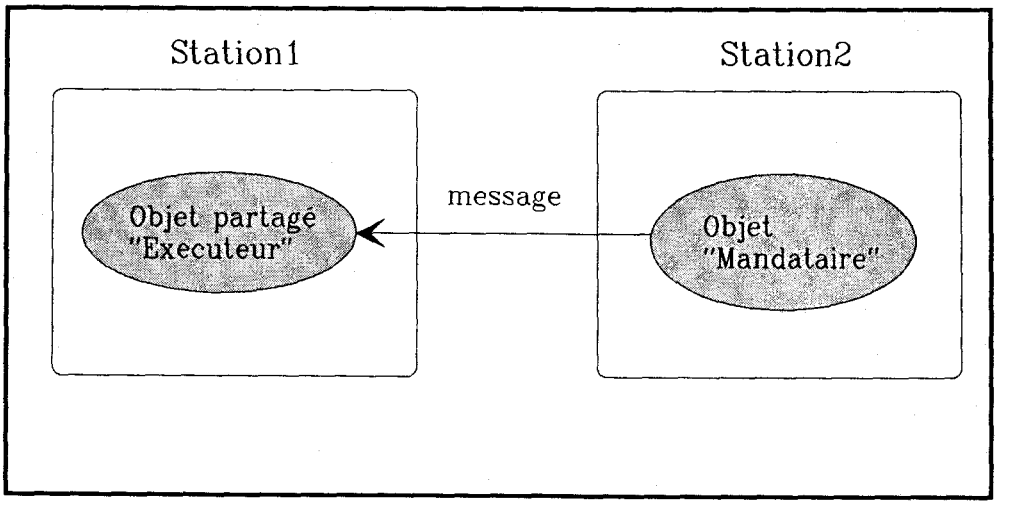


Figure 2.30. *Représentation d'un objet "exécuteur". C'est un objet partagé qui est accessible sur les autres stations par des objets mandataires.*

Nous l'avons appelé "exécuteur" car il va exécuter une action locale en fonction d'un message qu'il aura reçu d'une autre station. Cet objet va être instance d'une classe dans laquelle auront été définies des méthodes qui correspondront à des tâches bien précises. Ces méthodes pourront avoir ou non, des paramètres.

Exemple :

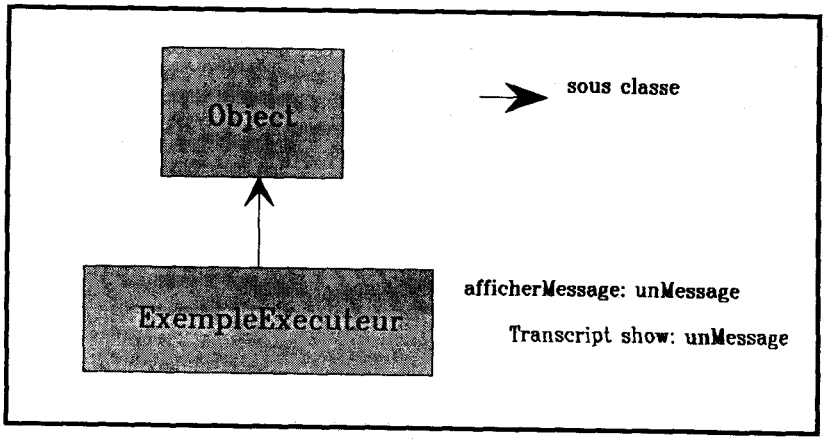


figure 2.31. *La classe ExempleExecuteur est sous-classe de la classe Objet. Elle possède un message d'instance qui affiche le message passé en paramètre dans la fenêtre Transcript.*

exécuteur : instance de la classe Exemple Exécuteur est partagé.

exécuteurMandataire : objet mandataire d'exécuteur sur une autre station.

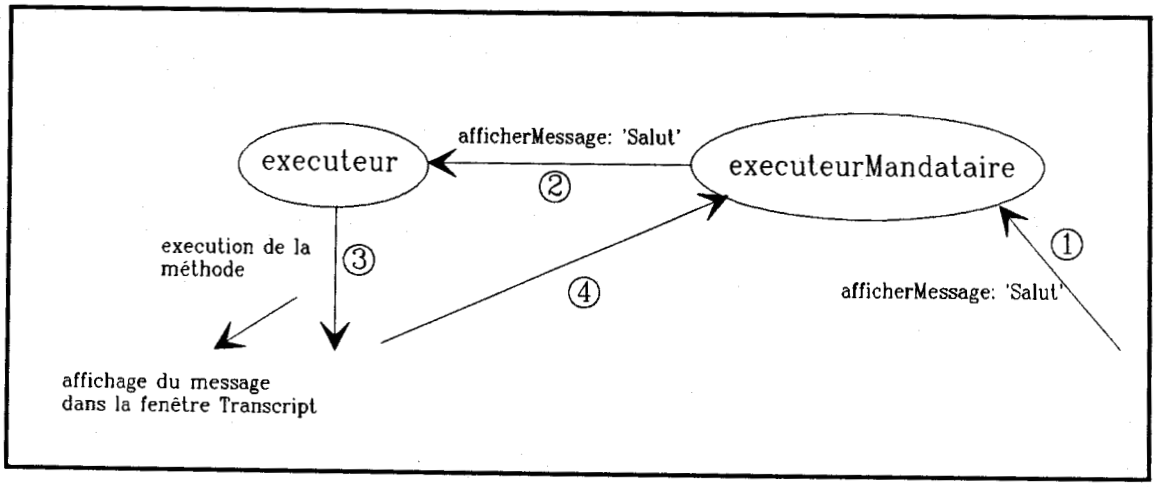


figure 2.32. *Exemple d'utilisation d'un objet "exécuteur". De la station distante, nous allons provoquer une action locale.*

1. Le message est envoyé à l'objet mandataire qui ne sait pas l'exécuter.
2. Le message est mis en forme et envoyé à l'objet.
3. Le message est reçu et exécuté.
4. Un message de retour est renvoyé au mandataire (acquittement).

En résumé, l'objet "Exécuteur" est un mécanisme qui permet d'exécuter n'importe quelle tâche prédéfinie à partir d'une autre station. Ainsi la figure 2.29 devient :

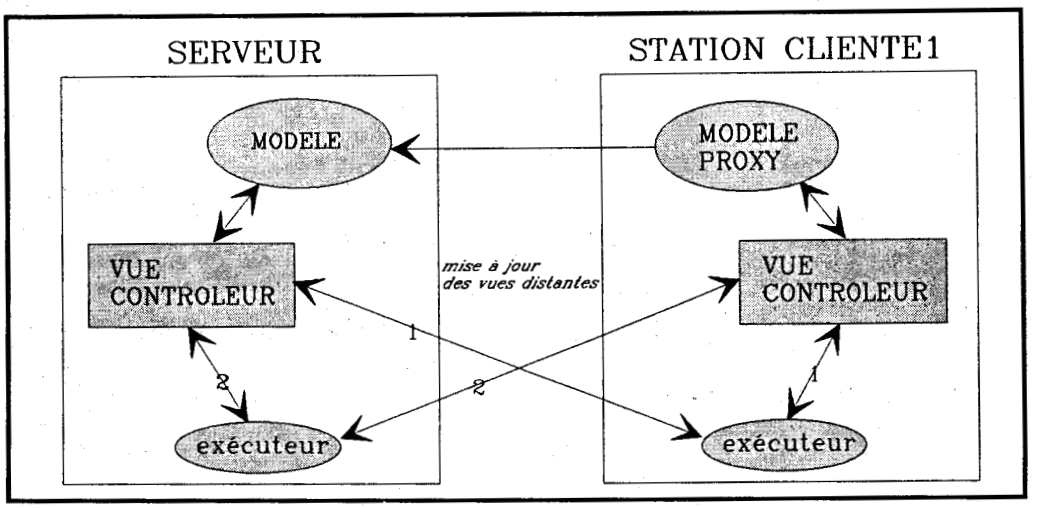


figure 2.33. *Architecture finale des applications. La mise à jour des vues est faite par l'intermédiaire d'objets "exécuteurs".*

Avant que d'entrer un peu plus en détail dans ce mécanisme, il est important de rappeler un point lié à l'architecture réseau pour la diffusion des informations entre les stations. Notre future architecture réseau sera de type étoilé, ne permettant pas toutes les communications directes entres machines (actuellement, sous réseau local Ethernet, les communications directes sont possibles mais nous n'utilisons pas cette facilité).

d) Réalisation de la diffusion

La diffusion des messages sera différente selon que ceux-ci sont envoyés d'une station cliente ou de la station centrale. Une raison à cela vient du fait qu'une station cliente accède uniquement à l'objet "exécuteur" du serveur, alors que le serveur accède à l'ensemble des "objets exécuteurs" clients. Ce choix nous permet de simplifier la phase de connexion et d'initialisation sur les stations clientes.

1) Diffusion à partir de la station centrale

Dans ce cas, la diffusion est faite directement à l'ensemble des stations clientes (simulation d'un multicast).

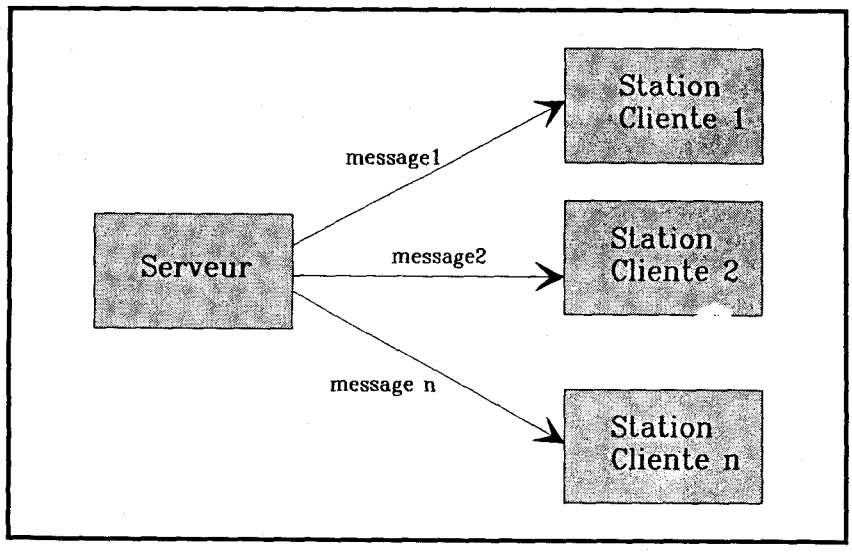


figure 2.34. *Diffusion à partir du serveur. La diffusion du message est directe sur toutes les stations clientes.*

2) Diffusion à partir d'une station cliente

Dans ce cas, le message est envoyé au serveur qui va router le message sur les autres stations clientes.

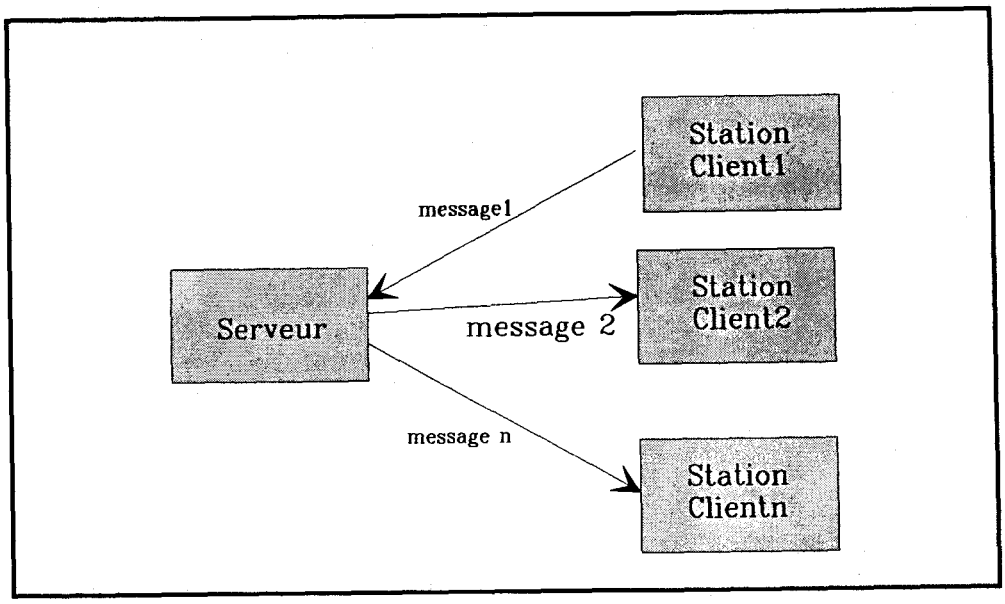


Figure 2.35. *Diffusion à partir d'une station cliente. Le message est envoyé au serveur qui va lui-même le rediffuser sur les autres stations clientes*

2.4.4 - Environnement de développement

Un aspect très important pour nous, était d'offrir un environnement de développement facilitant au maximum le travail du programmeur. Ainsi, pour l'implémentation d'applications W.Y.S.I.W.I.S., nous avons développé une boîte à outils incluant de nouveaux couples vue/contrôleur spécifiques aux applications coopératives.

La définition d'un nouveau couple vue/contrôleur se fait en résolvant les deux problèmes suivants :

. Indépendamment de la façon dont est géré le contrôle d'accès d'une application W.Y.S.I.W.I.S., une fenêtre dispose de deux types d'interactions en fonction de son statut. Il s'agit de définir les interactions (souris et clavier) autorisées lorsque le participant a le contrôle de l'espace public et celles lorsqu'il ne l'a pas.

. Il faut spécifier quelles modifications des vues entraîneront la diffusion de celles-ci aux autres stations.

Dans la présentation qui va suivre, nous allons utiliser deux exemples de couples vue/contrôleur pour illustrer les explications.

2.4.4.1 - Les interactions

Deux types d'interactions sont possibles en standard : Les interactions souris et clavier.

a) Les interactions souris

Nous rappelons que, dans l'environnement Smalltalk les trois boutons de la souris sont utilisés et ont un rôle bien précis en fonction de la fenêtre (donc du couple vue/contrôleur).

. Le bouton de gauche sert par exemple à sélectionner ou désélectionner un item dans une fenêtre liste ou à sélectionner une partie de texte dans une fenêtre texte.

. Le bouton du milieu renvoie généralement un menu propre à l'applicatif que le programmeur aura défini dans son modèle.

. Le bouton de droite renvoie un menu général à toutes les fenêtres.

Nous nous sommes particulièrement intéressés au rôle des deux premiers boutons pour chaque type de contrôleur que nous avons redéfini. Le troisième devant conserver le même comportement.

1) Bouton gauche : il faut déterminer si l'on inhibe ou non la fonction de ce bouton. La réponse n'est pas standard car elle dépend du type de contrôleur. Pour une fenêtre texte, nous l'avons gardée car des opérations de copie de l'espace public dans l'espace privé doivent être permises. Par contre, pour une fenêtre liste nous l'avons inhibée (la sélection n'est possible que par l'utilisateur qui en a le droit).

2) Bouton du milieu : La création d'une fenêtre se fait par l'instanciation d'une classe. L'un des paramètres à spécifier, si cela a été prévu dans le message de classe, est un sélecteur de message correspondant au menu. La méthode correspondante devra alors être définie dans la classe du modèle de l'application afin d'afficher le menu. Dans les couples que nous proposons, nous supposons qu'un menu différent peut être autorisé pour valider certaines actions même si le participant n'a pas le contrôle (pour permettre le parallélisme d'activités pour les utilisateurs). Les nouveaux messages de classes comprennent donc deux sélecteurs de messages pour les

menus. Le premier, pour le menu renvoyé lorsque le participant a le contrôle, et le second, lorsque le participant n'a pas le contrôle. L'exemple suivant correspond a un menu que l'on pourrait définir dans une fenêtre texte.

3) Bouton droit : Nous n'avons pas modifié ses fonctionnalités car nous souhaitons laisser à l'utilisateur la possibilité de gérer l'espace écran indépendamment du contrôle interne.

	Contrôle	Pas de Contrôle
Souris (Menu)	Copy paste cut accept	copy
Clavier	oui	non

figure 2.36. *Exemple de menu pour une fenêtre texte.*

Les nouveaux couples de vue/contrôleur gardent en grande partie les fonctionnalités des couples standards. Les classes de ces couples ont donc été définies comme sous-classes des classes déjà définies. Chaque contrôleur a accès à une variable qui lui indique son statut. Lors de l'appui sur le bouton du milieu de la souris, différentes méthodes sont activées dont la méthode nommée "yellowButtonMenu" définie dans la classe de la vue.

Cette méthode est la suivante :

```
yellowButtonMenu
        menuMsg = nil ifTrue: [^nil].
        ^model perform: menuMsg
```

figure 2.37. *Méthode classique.*

Son rôle consiste à renvoyer au modèle et à faire exécuter la méthode correspondante au sélecteur de message "menuMsg" défini lors de l'instanciation.

Nous avons réécrit cette méthode de la façon suivante :

```
yellowButtonMenu
        superView enContrôle = true
                ifTrue: [ menuMsg = nil ifTrue: [^nil].
                        ^model perform: menuMsg]
                ifFalse: [ menuPartMsg = nil ifTrue: [^nil].
                        ^model perform: menuPartMsg]
```

Figure 2.38. *Méthode modifiée pour renvoyer le menu correspondant au statut de l'utilisateur.*

La variable "enContrôle" a été rajoutée dans la classe correspondante aux vues principales des fenêtres. Sa modification est assurée par le gestionnaire de conférence. L'intérêt de l'avoir réécrite à ce niveau est que le contrôleur pointe directement sur le menu correspondant à son statut. Aucun test n'est à faire dans la méthode du menu pour déterminer qui fait appel au menu. D'une façon identique, les méthodes activées lors d'une pression sur le bouton gauche sont à redéfinir en fonction des droits que l'on accorde.

Remarque : Le test pour savoir si le participant a le contrôle ou non est déterminé à partir de la vue principale de l'application. Il s'agit d'un choix que nous avons formulé, car nous avons considéré dans un premier temps que le contrôle ou non de l'espace public était relatif à l'ensemble de l'application. Cela s'est avéré nécessaire pour l'application que nous développerons plus loin, mais nous pouvons parfaitement envisager que le contrôle puisse être affecté à une sous-fenêtre, ou à un objet manipulé dans la sous-fenêtre. Dans notre cas, nous avons réécrit la classe appelée "StandardSystemView". De plus, en réécrivant cette classe, nous pouvons redéfinir le menu général associé au bouton droit de la souris. Nous reviendrons sur cet aspect dans la conclusion.

b) Les interactions clavier

En règle générale, les interactions clavier dans l'espace public ne doivent pas être autorisées lorsque le participant n'en a pas le contrôle, car elles modifieraient le contenu des fenêtres.

Par contre, lorsqu'un utilisateur a le contrôle, les interactions clavier sont liées aux fonctionnement standard des fenêtres et aucune différence n'est visible.

Pour diffuser les mises à jour sur les autres stations, plusieurs choix restent possibles qui dépendent de la granularité choisie. La moins perturbatrice semble être une mise à jour après chaque caractère frappé, car cela permet aux autres utilisateurs de voir la construction du texte dynamiquement et ainsi, ils ne sont pas surpris par l'apparition soudaine d'un mot, d'une phrase ou d'un paragraphe.

Il peut cependant arriver que l'on souhaite formuler son texte sans que les autres en soient informés et le diffuser alors au moment voulu.

Nous n'avons pas de solution globale à donner pour choisir la granularité car cela dépend fortement des applications et de la demande des joueurs. La solution raisonnable consiste à en implémenter un maximun et à les valider par une expérimentation réelle.

2.4.4.2 - Mise à jour des vues distantes

Il s'agit dans un premier temps de spécifier les modifications de la vue que l'on souhaite diffuser aux autres utilisateurs. Aucune règle générale ne peut être donnée car cela est différent pour chaque type de vue. De plus, le choix d'une mise à jour dépendra également de la granularité choisie (par exemple pour une fenêtre texte, on peut choisir de diffuser à chaque frappe de caractère, mot, phrase ou lorsque l'utilisateur le décide). Lorsque cela est déterminé, il s'agit d'identifier et de réécrire les méthodes invoquées dans le couple vue/contrôleur lorsque les modifications choisies ont lieu.

Nous allons reprendre les deux exemples précédents pour illustrer la démarche.

1) Fenêtre liste

Il y aura mise à jour de l'ensemble des fenêtres lorsqu'il y aura modification de l'item sélectionné (désélection ou sélection).

2) Fenêtre texte

Cet exemple correspond à un cas de figure évoqué précédemment, car la diffusion dépendra tout d'abord de la granularité choisie. Ensuite, un texte affiché dépend généralement d'un contexte. C'est à dire qu'il doit évoluer en fonction de variables de son modèle (on le trouve souvent associé à une fenêtre liste). Lors de son instanciation, l'un des paramètres "unSymbole" (sélecteur de message) à spécifier dans le message de la classe correspond à une méthode du modèle qui renverra le texte à afficher en fonction du

contexte. Dans les classes des vues, une méthode nommée "update: unSymbole" est toujours définie. Elle sera déclenchée par le mécanisme de dépendance et invoquera la méthode "unSymbole" définie dans la classe du modèle . Les mises à jours ont été dans cet exemple implémentées dans la méthode "update:".

2.4.4.3 - Implémentation

Nous avons réécrit de nouvelles classes correspondant à de nouveaux couples vue/contrôleur. Une distinction est cependant à faire entre les couples utilisés sur le serveur de ceux utilisés sur les stations clientes. Cette distinction est dûe aux deux mécanismes de diffusion imposés par notre réseau. La figure suivante illustre la hiérarchie des classes client/serveur pour des couples de vue/contrôleur xte et l' e.

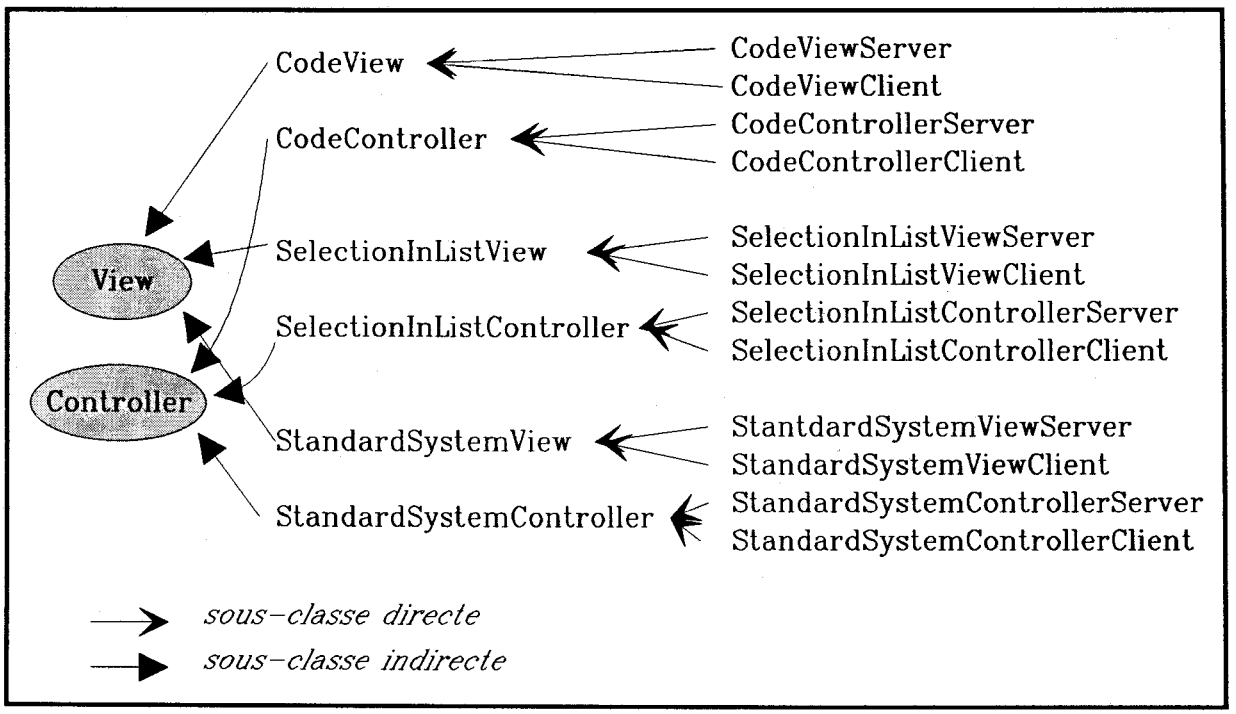


figure 2.39. *Hiérarchie des classes des nouveaux couples vue/contrôleur*

2.4.4.4 - Conclusion

Nous avons voulu offrir à un programmeur la possibilité d'utiliser pour des applications coopératives de type W.Y.S.I.W.I.S., un ensemble de vues et contrôleur gérant de façon transparente la mise à jour des vues sur l'ensemble des stations. Nous sommes tout à fait conscients que quelle que soit la richesse de cette boîte à outils nous

ne pourrons offrir l'ensemble des couples nécessaires au développement d'applications coopératives. Nous souhaitons que cette présentation offrira un guide méthodologique pour la création de nouveaux couples vue/contrôleur.

2.4.5 - Problème de la répartition

La priorité absolue est de réduire au maximun le flux d'informations entre les stations. L'architecture centralisée que nous proposons avec son modèle unique, n'est pas dans l'état actuel, entièrement satisfaisante. Comme nous l'avons indiqué, une première répartition a été faite avec les couples vue/contrôleur. Elle assure l'ensemble des opérations générales (le déplacement, le redimensionnement, le "scroll", etc...) sur les fenêtres.

En développant des applications, nous nous sommes aperçus que certaines opérations pouvaient être traitées localement, sans nuire à la cohérence globale. Si l'on considère un traitement, il peut être constitué d'une suite de phases locales et centrales. Le problème consiste donc à déterminer la décomposition fonctionnelle optimale.

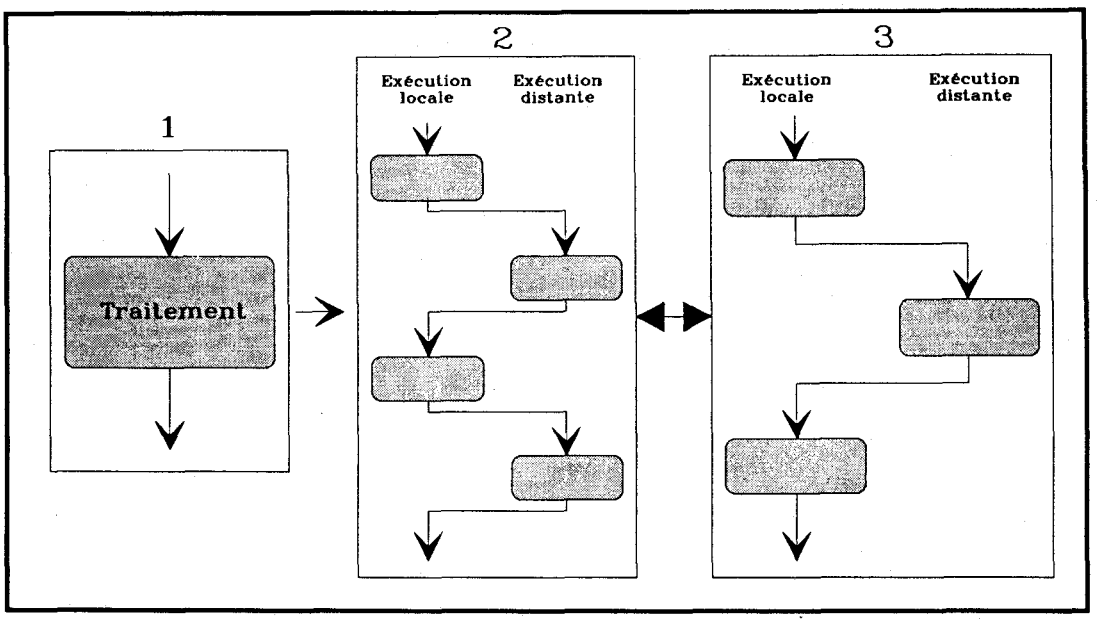


figure 2.40. *Décomposition du code. Deux exemples de décomposition. Lorsque des méthodes font appel à des exécutions distantes, il faut s'arranger pour que l'enchaînement des appels/réponses soit le plus faible possible.*

2.4.5.1 - Décomposition du code

Lorsque l'on évolue dans des environnements distribués, on se rend compte rapidement, qu'avec la puissance de plus en plus importante des stations de travail, le temps d'exécution d'une séquence de code est généralement très court par rapport au temps de transfert/récupération des données. Ce n'est pas la vitesse de transfert qui est pénalisante, mais ce sont les mécanismes de mise en forme des messages (sérialisation et désérialisation) et de synchronisation (processus). Plus le nombre de transferts sera faible, meilleur sera le temps de réponse. Nous nous sommes donc fondés sur ce critère afin de déterminer ce qui pouvait être ramené sur les stations clientes. Donner une procédure générale est délicat, mais nous pouvons l'illustrer sur deux des exemples que nous avons rencontrés dans l'application de simulation que nous avons implémentée.

Le premier exemple est relatif à une séquence qui nécessite d'abord une acquisition de données. Il s'agit de la proposition d'une valeur pour une case d'un tableur. Lorsque la case est sélectionnée, une option du menu permet d'ouvrir une boîte de dialogue afin de saisir la valeur. Si on laisse le modèle gérer seul le traitement, on arrive au cas de la figure 2.40. Le message est envoyé au modèle qui va demander, par l'intermédiaire d'un objet exécuteur sur la station cliente, l'acquisition d'une valeur et son envoi afin d'être traitée. On s'aperçoit que l'acquisition est locale à la station cliente. On va donc s'arranger pour la faire avant, et envoyer au modèle un message avec comme paramètre la valeur (figure 2.41)

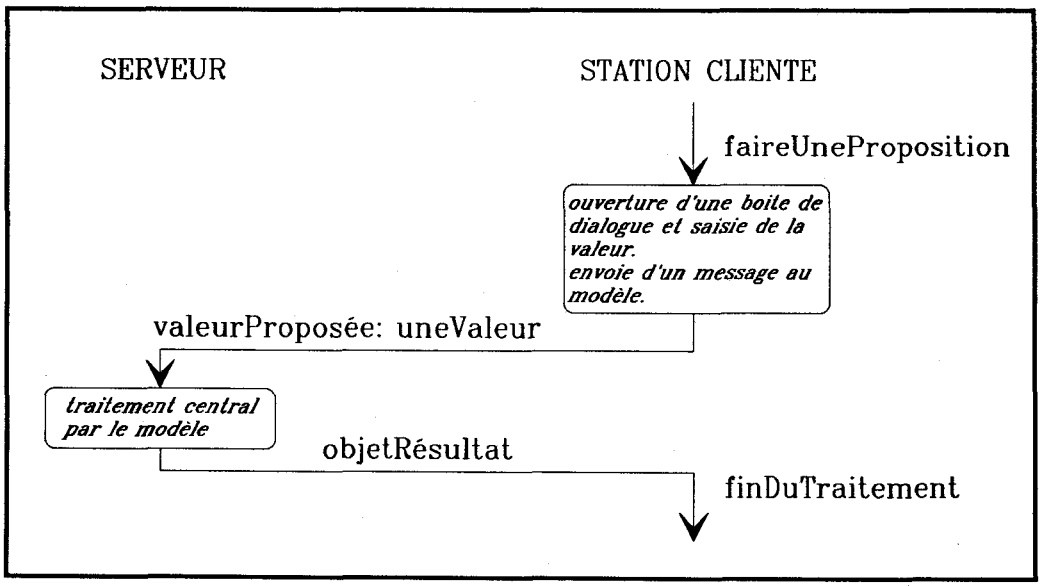


Figure 2.41. *Exemple de décomposition de code dans l'application de simulation. Avant d'avertir le modèle central, une exécution locale est effectuée.*

Le second exemple est l'ouverture d'une fenêtre à partir de l'application qui permet de consulter des statistiques, afin d'aider les utilisateurs à prendre des décisions sur les valeurs à insérer dans le tableur. Là aussi, il s'agit d'une opération locale indépendante des autres participants. Elle sera donc directement exécutée localement. Aucun message ne sera envoyé au modèle central.

2.4.5.2 - Implémentation du code

Avant de donner les détails sur la façon dont nous opérons pour implémenter localement le code, il est nécessaire de préciser le fonctionnement des menus.

Lors de la création d'un menu, les options proposées sont associées à des sélecteurs de messages dont les méthodes sont spécifiées dans le modèle. Lorsque l'on appuie sur le bouton de la souris, le contrôleur va gérer le menu qui correspond au sélecteur de messages donné lors de l'instanciation de la fenêtre. Tant que l'on garde le doigt sur le bouton, on va pouvoir se déplacer sur les différentes options affichées sans rien déclencher. Lorsque l'on relâche le bouton, le contrôleur va envoyer au modèle le sélecteur de l'option choisie afin que la méthode soit exécutée.

Dans quelques fenêtres, un artifice très pratique a été développé. Lors de la sélection d'une option, le contrôleur va d'abord vérifier si le sélecteur ne fait pas partie d'une liste avant de l'envoyer au modèle. Dans le cas où il en fait partie, la méthode définie dans la classe du contrôleur sera exécutée, et seulement après celle du modèle. La figure 2.41 illustre ces explications.

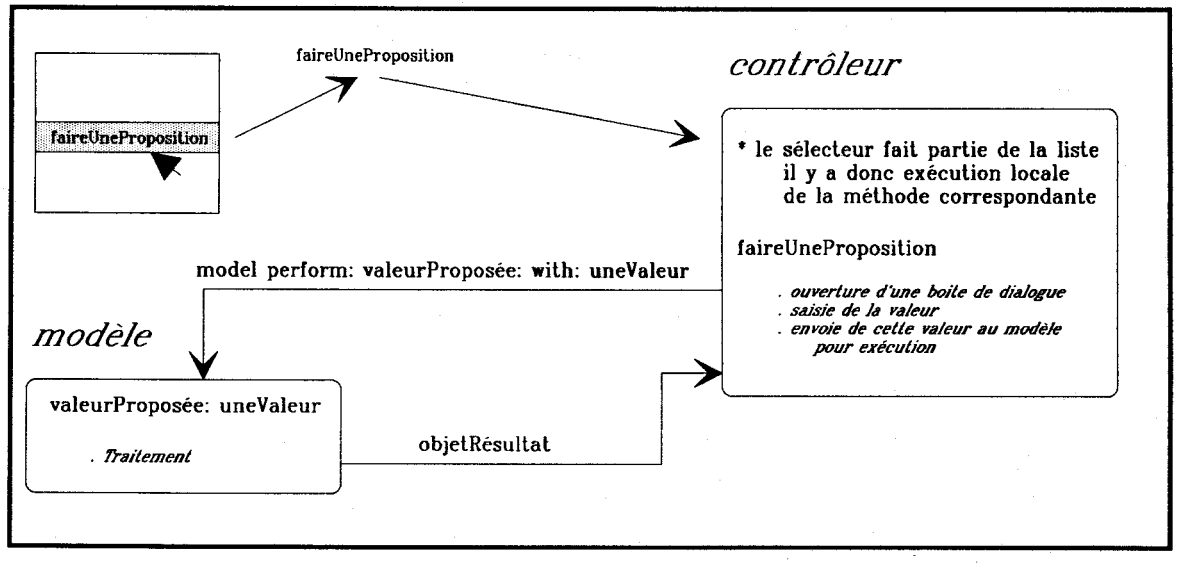


figure 2.42. *Fonctionnement d'un menu.*

La méthode que nous avons choisie pour implémenter le code dupliqué sur les stations clientes est fondée sur le mécanisme précédemment exposé.

Une autre solution consisterait à implémenter le code au niveau du modèle (implémenter les méthodes correspondant aux sélecteurs de messages dans la classe du modèle). Dans ce cas, notre modèle n'est plus tout à fait un 'proxy'. Quelques méthodes sont exécutées localement. Pour cela le modèle de l'application sera instance d'une sous-classe de la classe Proxy afin de bénéficier du mécanisme d'indirection. Cette solution nécessiterait cependant quelques modifications dans le système de partage d'objets.

2.4.5.3 - Conclusion

La conclusion que nous tirons de notre expérience dans ce domaine est qu'il faut optimiser le nombre d'appels à des procédures distantes. Globalement, c'est le temps de transfert/récupération des données qui est pénalisant dans l'ensemble du traitement. C'est donc sur ce critère qu'il faut jouer pour réduire les temps de réponse.

2.4.6 - Vue générale de l'architecture

L'architecture globale du système, avec le gestionnaire de conférence et la partie applicative, est représentée sur la figure 2.43.

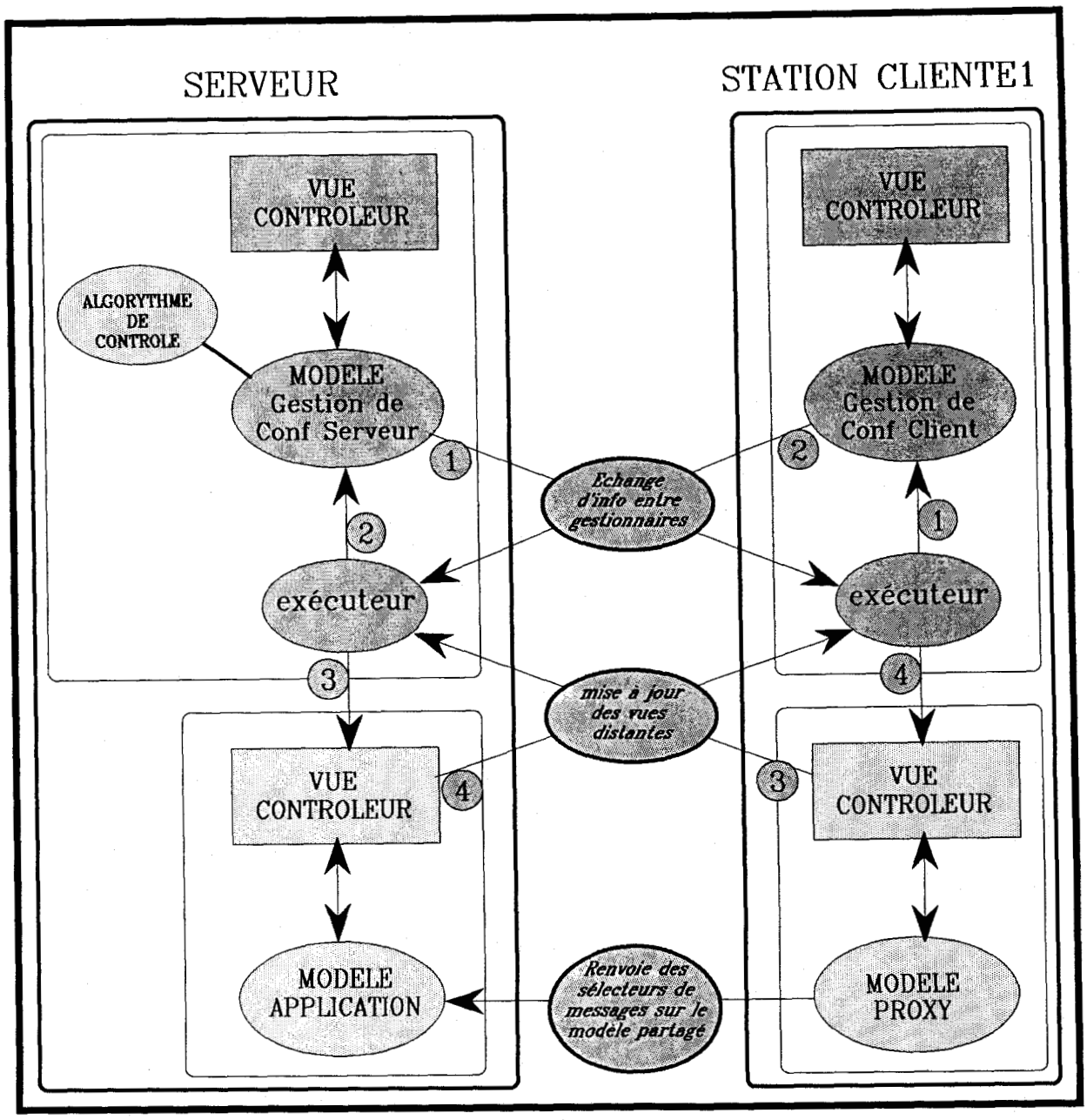


Figure 2.43. *Architecture générale du système de conférence temps-réel.*

Comme nous pouvons le voir sur le schéma, les échanges d'informations entre les différents modèles implémentés sur les stations se font toujours par l'intermédiaire d'objets "exécuteurs".

Nous pouvons également accueillir des applications coopératives qui ne soient pas W.Y.S.I.W.I.S.. Le gestionnaire va pouvoir être utilisé pour diffuser les informations aux stations et assurer une cohérence globale. L'application que nous avons implémentée et que nous allons présenter au chapître suivant est composée de phases W.Y.S.I.W.I.S. et d'autres non W.Y.S.I.W.I.S.. Un exemple de code pour une petite application coopérative est donné en annexe II afin d'illustrer le développement et l'utilisation des classes que nous offrons.

III - APPLICATION COOPERATIVE

JEU DU KANBAN

L'application que nous avons choisie d'expérimenter sur notre prototype de Téléconférence Temps-réel est un jeu à vocation pédagogique. Ce jeu appelé JEU du KANBAN est une simulation d'atelier qui permet d'apprécier les avantages et les inconvénients de deux méthodes différentes de planification. Ce jeu conçu par Michel Gref est utilisé entre autres à l'Université de Poitiers dans des enseignements de type Licence et DUT de Production.

Cette application nous a semblé particulièrement intéressante à implémenter et à expérimenter pour les raisons suivantes :

. Il s'agit d'une véritable application coopérative, qui nécessite des phases de coopération et de négociation entre les participants, pendant lesquelles les aspects "W.Y.S.I.W.I.S." et "espaces partagés" sont très importants.

. Des personnes compétentes, ayant utilisé le jeu de façon classique dans de nombreuses formations, pourront nous faire part de leurs remarques sur les aspects positifs et négatifs du système.

. Le nombre de postes nécessaires à son déroulement (5 à 6), correspond à celui que nous nous sommes fixés lors de la conception du prototype de conférence.

Il est nécessaire, pour comprendre l'application partagée que nous avons réalisée, de présenter dans un premier temps le jeu et ses règles, tel qu'il se déroule, dans les formations classiques. Ce chapître est donc divisé en 3 parties. La première partie va correspondre à la présentation du jeu. Dans une seconde partie, nous présenterons l'application partagée et les décisions d'interfaces. Nous terminerons par la description de l'architecture choisie pour son implémentation.

3.1 - Présentation du Jeu traditionnel

Nous n'allons présenter ici que le strict minimum nécessaire à la compréhension du jeu de simulation car sa description complète est donnée en annexe.

Il s'agit, comme nous l'avons dit, d'une simulation d'atelier qui permet d'apprécier les avantages et les inconvénients de deux méthodes différentes de planification :

. la méthode MRP (Manufacturing Ressource Planning),

. la méthode Kanban.

La simulation classique est prévue sur 8 semaines, divisées en 4 périodes de deux semaines, qui correspondent chacune, à une certaine forme d'organisation de l'atelier.

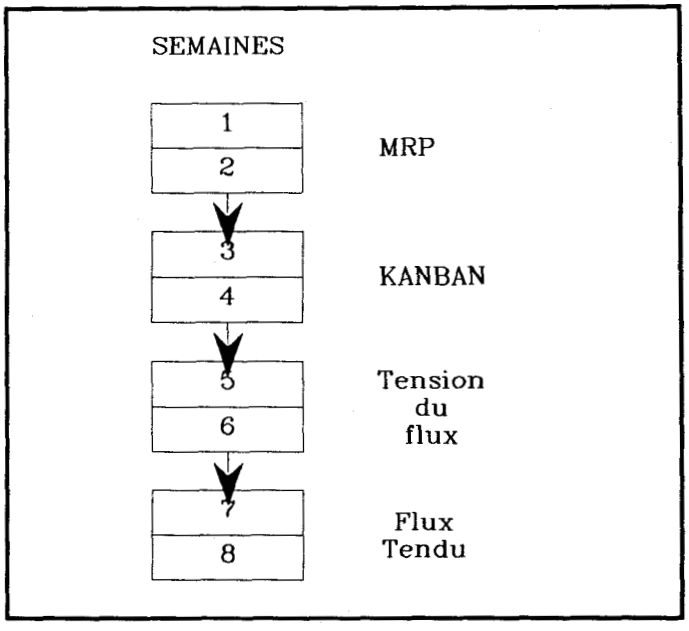


Figure 3.1. *Représentation des différentes semaines de simulation.*

3.1.1 - Présentation générale

3.1.1.1 - Données de base

Il s'agit de la simulation d'un atelier de production de réducteurs d'une société fictive appelée "Redix". L'atelier est composé de 5 postes de travail (Engrenage, Carter, Couronne PhaseA, Couronne PhaseB, Montage), et chacun d'eux est tenu par une équipe de joueurs (1 ou 2). Une sixième équipe joue le rôle de responsable de direction. L'animateur est chargé de veiller au bon déroulement du jeu et tient généralement le rôle du responsable de direction. Les produits sont représentés par des jetons, de

couleurs et de formes différentes, qui sont placés dans des containers qui circulent entre les postes.

a) Durée du Jeu

La durée normale est d'un jour pour la simulation. Celle-ci peut être suivie d'une ou plusieurs journées de réflexion en groupe pour tirer le meilleur parti des observations faites au cours du jeu. Suivant l'entreprise, il pourra s'agir d'engager une réflexion sur l'amélioration de la qualité, l'amélioration de la fiabilité des machines, etc..., ou sur la possibilité pratique de mettre en oeuvre la méthode Kanban dans un atelier.

b) Disposition des joueurs

Les joueurs prennent place autour d'une table, les postes pouvant être disposés de différentes façons. Un exemple de disposition est donné ci-dessous.

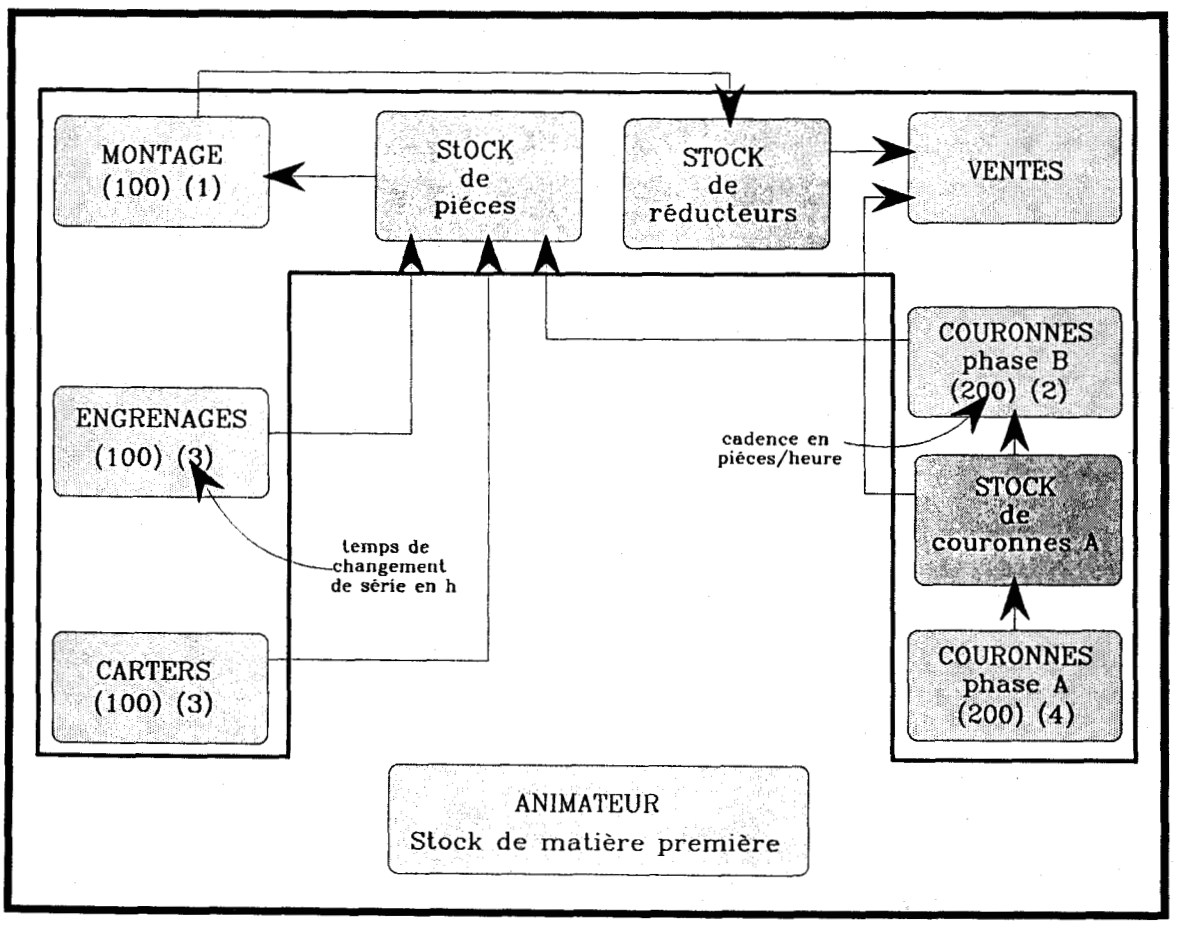


Figure 3.2. *Exemple d'une disposition des joueurs.*

c) les produits

L'atelier produit six types de réducteurs différents qui sont une combinaison de trois composants distincts : un engrenage, un carter et une couronne usinée en phase B. Les postes de fabrication correspondant à ces composants les fabriquent dans des références différentes symbolisées par des couleurs. Il y a d'autre part des ventes de pièces semi-finies, qui sont destinées à la rechange mais cela ne concerne que les couronnes-phaseA.

3.1.1.2 - L'échelle de temps

L'unité élémentaire de temps est l'heure. Il y a 8 heures de travail par jour, 5 jours par semaine. Le pas de simulation est l'heure, c'est-à-dire que le jeu avance d'heure en heure. A chaque pas, il y a soit production d'un container, si la machine est en activité, soit arrêt de la machine (panne, réglage, etc...).

La synchronisation des joueurs est assurée par l'animateur, qui affiche l'heure de la semaine à l'aide du calendrier prévu à cet effet. L'utilisation des feuilles de poste aide également à la synchronisation des joueurs.

3.1.1.3 - Le processus de fabrication

L'atelier est composé de 5 postes de travail, dont 4 postes d'usinage et 1 poste de montage. Une fois les pièces usinées, elles sont rangées dans un magasin de pièces qui est situé près de la section de montage. Les couronnes usinées en phase A sont également mises en stock, en attendant d'être prises en phase B. Le stock de produits finis est rangé dans un magasin, à proximité de la ligne de montage (figure 3.3).

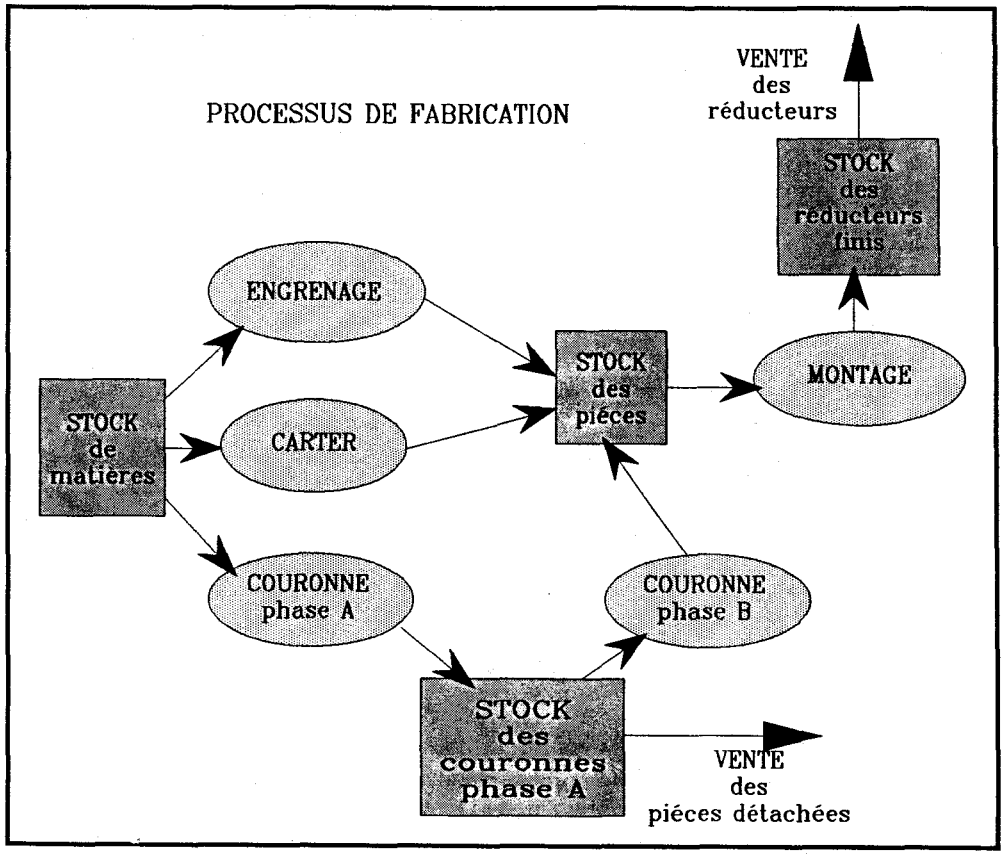


Figure 3.3. *Schéma du processus de fabrication des différents composants*

L'organisation des machines est conçue de telle façon qu'il y ait un équilibrage satisfaisant entre les capacités installées et les ventes. Ainsi, chaque poste de travail a une cadence de travail normale.

3.1.1.4 - Changement de série

Lorsqu'un poste décide de changer de série, il doit s'interrompre pendant la durée prévue et noter le changement sur la feuille de poste.

La décision de changer de série est toujours prise par le poste de travail. Mais les modalités de cette décision diffèrent suivant la méthode de programmation de la production.

Au cours de la phase I du jeu (méthode MRP), ces décisions devront être programmées en début de semaine.

Au cours des phases suivantes (méthode KANBAN), ces décisions seront prises au fur et à mesure de la fabrication.

3.1.1.5 - Incidents et informations

Le JEU DU KANBAN est destiné à simuler le fonctionnement d'un atelier dans des conditions représentatives de la réalité vécue en entreprise. Il est donc normal que des évènements variés viennent interférer avec le fonctionnement prévu.

Il y a deux types d'événements possibles :

- l'incident de fabrication ou d'approvisionnement, qui surgit de façon aléatoire.

- l'information qui évoque des questions d'ordre général à des moments prévus dans le déroulement du jeu.

Les événements sont matérialisés par des cartes.

3.1.2 - Déroulement du jeu

3.1.2.1 - Rôle des équipes

Il y a cinq postes de travail et un poste de responsable de direction.

a) Rôle des équipes de travail

- Ils programment la fabrication sur chacun des postes (quantité à faire pour la semaine, détail de chaque lancement), en tenant compte des informations disponibles sur demande auprès de la Direction.

- Ils sont confrontés à des incidents de fabrication, (qualité, pannes, fournisseurs en retard, etc...), dont ils subissent les conséquences, et qui peuvent leur imposer de modifier leur programmation. Les incidents sont représentés par des cartes tirées au hasard avec des dés.

- Ils débattent entre eux des thèmes évoqués chaque semaine dans les Cartes Information.

- A la fin de chaque phase du jeu, ils dégagent les points-clés relatifs à la méthode de planification qui vient d'être appliquée.

- A la fin du jeu, ils mesurent les résultats atteints : satisfaction des clients, efficacité industrielle, qualité.

b) Rôle du responsable de direction

- Il assure les expéditions et enregistre les manquants.

A partir de la liste des demandes, il effectue les livraisons aux clients. En pratique, cela consiste à enlever les containers, des références demandées du stock (produit fini ou pièces détachées).

- Il communique aux postes certaines informations commerciales et techniques à leur demande.

En tant que responsable commercial, il dispose de différentes informations sur les ventes.

Pour programmer leur production, les postes peuvent faire appel aux informations statistiques évoquées plus haut. Le responsable commercial communique l'information en transmettant la feuille correspondante au poste qui en fait la demande. On notera que, si un poste n'exprime pas une demande d'information, celle-ci n'a pas à lui être communiquée.

- Il déclenche le tirage des Cartes Information. Au début du jeu, l'animateur pose ces cartes sur la table. Celles-ci sont classées dans l'ordre chronologique. Leur tirage est déclenché par un code qui figure sur le dossier des expéditions.

- Il effectue les inventaires.

- Il analyse les résultats.

3.1.2.2 - Démarrage du jeu

Chaque participant reçoit un dossier contenant les informations préalablement exposées qu'il lit, avant la séance, ou en début de séance. L'animateur s'assure que les principaux points sont compris.

Il rappelle les buts de la simulation, et insiste sur le fait qu'elle doit déboucher sur une réflexion de groupe visant à améliorer le fonctionnement de l'atelier. Le jeu du KANBAN n'est pas un jeu de compétition, mais un exercice de groupe avec des objectifs partagés nécessitant une coordination. Les participants peuvent d'ailleurs changer de poste au cours des différentes phases, sans que cela nuise au jeu.

La mise en place des joueurs se fait de la façon suivante :

1. Attribution à un joueur du rôle de responsable des ventes, et remise du dossier correspondant.
2. Création des équipes de poste (1 à 2 joueurs par poste).
3. Mise en place des postes autour de la table, en commençant par placer le poste de montage, puis les autres à partir de celui-ci.
4. Attribution des feuilles de postes, et commentaire sur leur utilisation (figure 18).
5. Distribution des jetons de matière première aux postes qui en consomment (premier usinage).
6. Mise en place des containers au milieu de la table. L'animateur explique le rôle des cartes d'incidents de fabrication et d'information.

3.1.2.3 - Organisation du jeu

Le premier objectif demandé aux participants est l'établissement d'un plan général de production pour la semaine. A partir de celui-ci, un plan d'activité individuel par poste en sera déduit. Ce plan devra être respecté par les joueurs et cela quels que soient les aléas rencontrés dans le processus de fabrication (aléas introduits par les cartes incidents) lors de la première semaine de simulation.

a) Etablissement du plan de production

Pendant cette phase, l'animateur observe l'attitude des joueurs et son rôle consiste uniquement à communiquer, lorsque les participants en font la demande explicite, les informations qu'il a à sa disposition dans son dossier de direction.

Après avoir effectué la phase de démarrage présentée précédemment, l'animateur distribue un plan de production vierge et laisse les joueurs se débrouiller.

Cette phase dure en moyenne deux heures pendant lesquelles la négociation et la discussion sont intenses entre les joueurs. A partir des informations mises à leur disposition, ils vont essayer d'organiser la production en fonction des besoins de chacun.

En effet, chaque décision de production d'un produit affecte l'ensemble des postes, car les réducteurs sont un assemblage des pièces produites sur chacun d'eux. Durant cette phase, les participants discutent tous autour d'un même plan qu'ils vont mettre à jour, au fur et à mesure des décisions prises et acceptées par l'ensemble du groupe.

Concrètement, l'animateur trace le plan sur le tableau, permettant ainsi une vision convenable pour tout le monde. Il va également se charger de remplir les cases pour lesquelles les joueurs se seront mis d'accord, mais pourra aussi à tout moment céder sa place et donner la craie à un joueur qui en aura fait la demande.

On s'aperçoit donc que dans cette phase le tableau va être le support qui va focaliser l'attention du groupe et matérialiser sa progression. De plus, la possibilité de désigner une partie du plan de production est importante.

Différentes stratégies, dans la prise de décision, sont utilisées pour atteindre l'objectif. Dans la plupart des cas un "meneur de jeu" va essayer de faire un consensus parmi les joueurs sur une valeur, en dégageant les conséquences de la décision pour tous les postes. C'est à dire que les joueurs vont faire des propositions qui seront toutes évaluées afin de déterminer celle qui satisfait le plus grand nombre d'entre eux ou celle qui est le meilleur compromis.

En principe, la définition du plan débute par la validation des cases dont la valeur est non problématique. Il s'agit de celles qui correspondent aux réducteurs dont la vente est sûre (ceci à partir des informations distribuées). Elle se poursuit ensuite, avec le remplissage des cases restantes pour lesquelles les décisions sont moins évidentes et qui nécessitent alors plus de discussions. Après chaque décision, il est important que les joueurs s'assurent que la production programmée est potentiellement réalisable, et cela en tenant compte du temps de changement de série fixé pour chaque poste.

Lorsque le groupe se sera mis d'accord sur le plan de production de la première semaine, les joueurs vont en extraire le plan d'activité pour leur poste. Ce plan, qui devra être suivi scrupuleusement lors de la première phase de simulation, contient les ordres de fabrication avec la quantité d'articles à produire, l'heure de début de fabrication et la durée de la production.

Quand tous les joueurs ont réalisé leur plan, ils sont prêts à passer à la phase suivante et la simulation proprement dite peut débuter.

Il faut savoir que le jeu est conçu de façon à ce que les joueurs aient beaucoup de difficultés pour parvenir à un plan satisfaisant tout le monde. En effet dans les premières phases du jeu, l'ensemble des contraintes imposées est tel qu'il ne permet pas de satisfaire les demandes programmées. Il s'agit essentiellement de faire prendre conscience aux joueurs (élèves) qu'une gestion optimale nécessite de nombreuses conditions et de nombreux pré-requis qui seront introduits au fur et à mesure par les

cartes d'informations. Ces dernières seront aussi le point de départ de discussions sur différents concepts.

b) Phase de simulation (première semaine)

Pendant cette première phase de simulation, les décisions de changements de séries ne sont pas prises au fur et à mesure de la fabrication, mais sont programmées en début de semaine sur le plan d'activité déduit de la phase précédente. Quelles que soient les conséquences sur la production, et plus particulièrement sur le fait de ne pas pouvoir satisfaire la demande, ce plan devra être respecté.

Avant de commencer la simulation, la possibilité de constituer des stocks initiaux est également offerte aux joueurs.

Lorsque les décisions sont prises, la simulation proprement dite peut commencer. A chaque changement d'heure, le rôle de l'animateur va consister à :

. jeter le dé, une première fois, pour déterminer s'il y a lieu d'introduire une carte d'incident et, si oui, le jeter une seconde fois pour avoir la conséquence de la carte relative au poste concerné.

. lire la carte d'information programmée à l'heure courante.

. effectuer les ventes, en prélevant les jetons dans les stocks correspondant aux réducteurs finis et aux pièces détachées, à partir de son planning des ventes.

Le changement d'heure est laissé à l'appréciation de l'animateur qui va cependant s'assurer que les joueurs ont effectué la tâche correspondante à leur plan d'activité. Tout au long du jeu, des explications, des discussions, à partir des concepts proposés par les cartes incidents et d'informations, pourront avoir lieu.

Il est à noter que le comportement des joueurs dans cette première phase du jeu est relativement passif car ils suivent leur plan d'activité en subissant les conséquences des cartes d'incidents tirées. Aucune prise de décision n'est possible. Il s'agit, dans cette phase, de faire prendre conscience aux étudiants des problèmes liés au manque de flexibilité de la méthode de planification courante : en particulier, l'impossibilité de faire face à une demande spontanée de produits, des temps de changement de séries trop important, et le mauvais étalonnement des ventes sur la semaine.

Lorsque cette phase est terminée, un inventaire des stocks et une analyse des résultats sont effectués. Il s'agit pour les joueurs d'arriver aux conclusions précédemment exposées.

Le temps de déroulement de cette phase est très variable car il dépend des discussions et des questions qu'il peut y avoir dans le groupe. Sans discussion, une dizaine de minutes serait suffisante pour parvenir au résultat.

c) Simulation des semaines suivantes

A partir de la troisième semaine, la méthode de planification mise en jeu est la méthode KANBAN dont une description plus complète est donnée en annexe. Nous rappelons cependant que celle-ci est basée sur des kanbans, qui sont des étiquettes décrivant plus ou moins le produit, et qui vont circuler en circuit fermé.

En effet, un nombre fixe de kanbans est mis en jeu et dans notre cas ils seront, soit sur le planning du poste concerné, soit sur les containers placés en stock. Lorsque le produit en stock sera utilisé par le poste situé en aval de la chaîne de production, son kanban sera renvoyé au poste initial. Ainsi les opérateurs pourront, à tout instant, et par un simple coup d'oeil à leur planning kanban, déduire le nombre de produits qu'ils ont actuellement en stock.

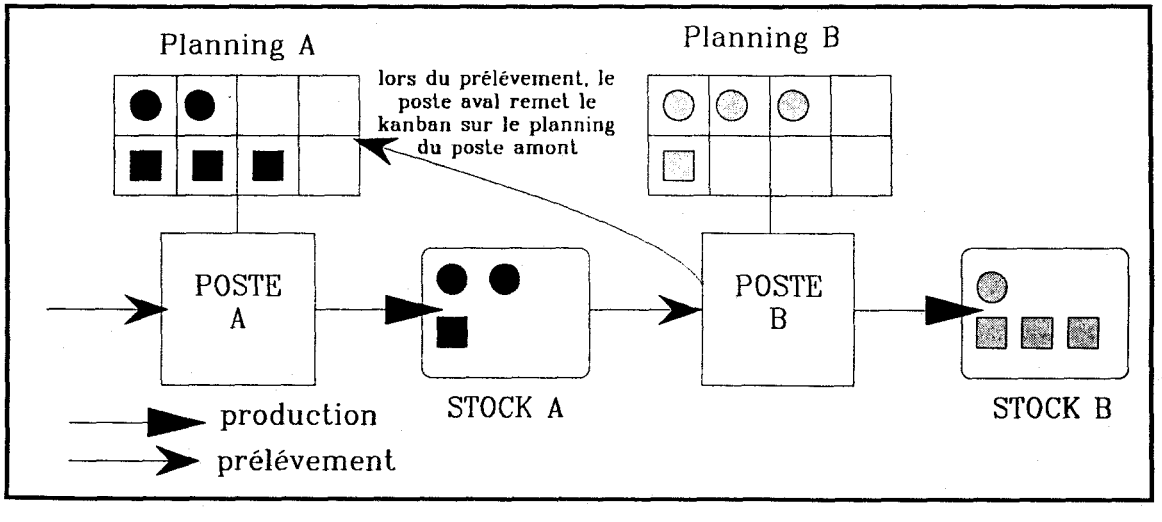


Figure 3.4. *Cycle d'utilisation d'un Kanban.*

Pour sa mise en application, un planning Kanban est attribué à chaque poste ainsi qu'un nombre de kanbans. Ce dernier est laissé au libre choix des joueurs. L'animateur pourra, au fil du jeu, affiner ce nombre avec les joueurs qui en auraient choisi un, non approprié.

Durant cette phase, et tout en respectant les délais de changement de séries, les joueurs peuvent à chaque heure prendre leurs décisions en fonction des besoins.

La figure suivante illustre le planning fourni aux postes et sur lequel les kanbans sont placés.

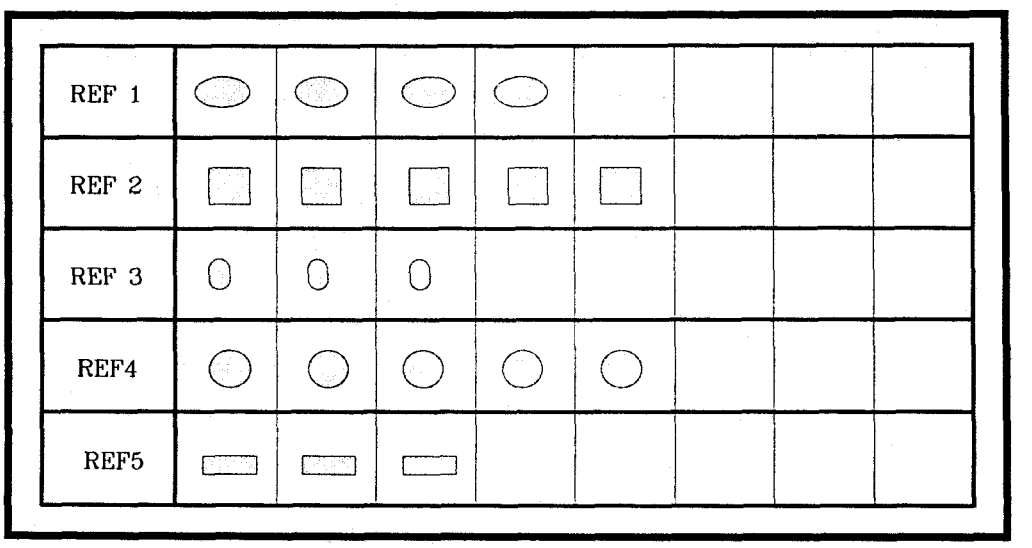


Figure 3.5. *Schéma d'un planning Kanban.*

D'une manière identique à la phase précédente, l'animateur détermine le moment du changement d'heure à partir de son observation générale du jeu.

Cette phase est plus riche en discussions car, comme les joueurs ont la possibilité de réagir aux évènements extérieurs simulés par les cartes "incidents", ils pourront et vont argumenter leurs décisions et ce, d'autant plus qu'ils ont une vision globale du jeu (sur les stocks et les plannings).

Ici encore, on va introduire, par l'intermédiaire des cartes d'informations et de certaines cartes dites de protection, l'ensemble des conditions nécessaires au bon fonctionnement de cette méthode de planification. Il va s'agir, entre autres, de réduire les temps de changements de séries, d'étaler plus régulièrement les ventes, de faire des interventions préventives sur les machines afin de réduire les probabilités de pannes, etc.

d) Conclusion

La première constatation est que l'intérêt du jeu est fortement lié aux discussions qui découlent des cartes incidents et des cartes informations. Ces dernières ont été conçues et programmées pour mettre en évidence les conditions nécessaires au bon

déroulement des méthodes (il en résulte, par exemple, qu'il est très important dans la méthode Kanban de diminuer les temps de changement de série pour les postes).

Les phases successives du jeu sont relativement indépendantes et on peut concevoir un arrêt entre celles-ci. De plus, il est tout à fait envisageable de figer le jeu au cours d'une phase de simulation (semaine), de faire un cours sur un concept exprimé par la carte incident, et de reprendre la simulation.

Ceci n'est pas vrai pour la première phase concernant la réalisation du plan de production car celle-ci demande une concertation générale et toute interruption serait néfaste au bon déroulement de la prise de décision.

Le type d'activité des joueurs évolue au fur et à mesure du déroulement du jeu. Cela peut se résumer de la façon suivante par ce tableau :

type d'activité	temps	prise de décision	activité	interruption possible	remarques
Présentation générale de la simulation	2 h	oui (attribution des postes)	synchrone	oui	. nécessité d'avoir un espace partagé . rétroprojecteur . possibilité de désignation
Plan de production	2 ou 3 h	oui collective	synchrone	non	. nécessité d'avoir un espace partagé . possibilité de désignation . prise de décision collective (consensus, vote)
Méthode de planification MRP	environ 1 à 2 h suivant discussion	non	asynchrone avec phase de synchro	oui	. partage de données communes (stock) point de vue différent possible sur celles-ci . les interactions des participants sont asynchrones mais il y a synchronisation dans le passage à l'heure suivante
Méthode de planification Kanban	environ 1 à 2 h	oui individuelle	asynchrone avec phase de synchro	oui	. partage de données . les interactions des participants sont aussi asynchrones avec des moments de synchronisation . leur attitude est active car ils peuvent prendre des décisions

Figure 3.6. *Tableau récapitulatif du type d'activité en fonction des différentes phases du jeu.*

Un problème que rencontrent les formateurs avec cette pédagogie, est la prise de notes d'information au cours du jeu. Il est en effet difficile de faire une synthèse sur le déroulement des activités. Il est encore plus compliqué de reprendre la simulation dans un état donné avec des options différentes pour expliquer une solution que les joueurs auraient pu adopter et qui aurait conduit à un résultat plus satisfaisant.

3.2 - Application Informatique

Notre objectif était d'implémenter cette application de simulation sur notre système de conférence temps réel pour permettre à un groupe de personnes de suivre la même formation, mais à distance (nous envisageons également qu'ils puissent se trouver dans une même pièce et que les stations servent de support).

Nous avons appréhendé le problème en respectant le plus fidèlement possible le scénario classique du jeu, mais en lui apportant des fonctionnalités supplémentaires liées à l'environnement informatique. L'interface résultant de cette approche reprend en grande partie les "formes" papier du jeu classique.

Les différentes phases du jeu et les règles restent identiques à celles exposées précédemment.

3.2.1 - Phase d'introduction et de présentation du jeu

3.2.1.1 - Présentation de l'interface réalisée

Nous commençons par la phase de présentation qui est très importante car, comme nous l'avons vu, il s'agit pour le formateur d'expliquer les objectifs et les règles du jeu. Il doit également attribuer le rôle des joueurs.

Pour cela, il utilise classiquement le tableau et le rétro-projecteur qui sont pour lui le moyen de focaliser l'attention du groupe et de signaler les points particuliers par une désignation appropriée.

Ce type d'activité cadre tout à fait avec une phase de travail W.Y.S.I.W.I.S. puisque tous les joueurs doivent voir la même chose. Nous avons pris l'idée de transparents que l'on présente les uns après les autres. L'espace partagé correspond donc à une fenêtre dans laquelle seront affichés des plans successifs commandés par l'animateur qui s'en servira comme support aux explications fournies. La figure suivante illustre deux de ces plans.

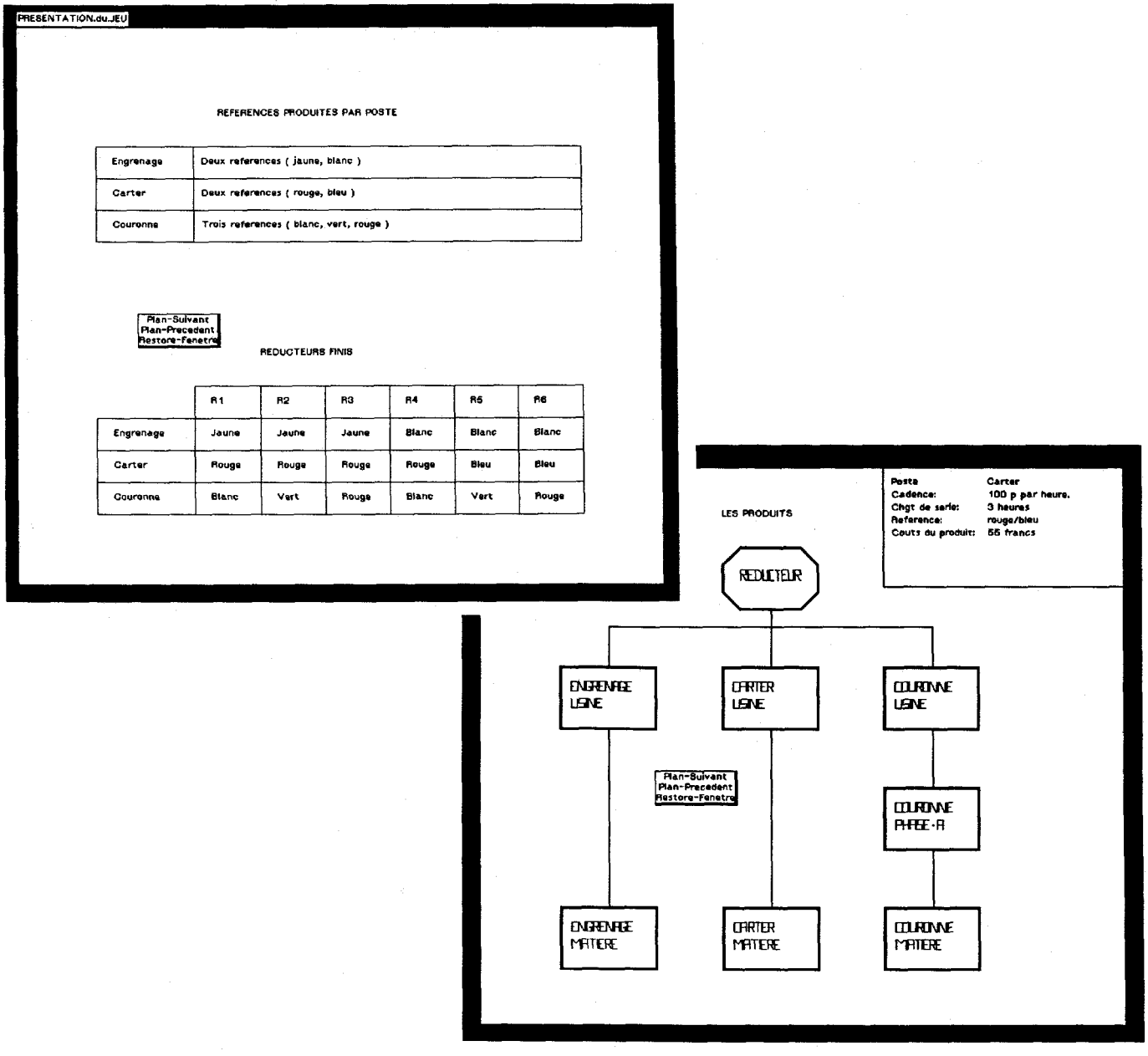


Figure 3.7. *Deux exemples de plans présentés aux joueurs pendant cette première phase.*

3.2.1.2 - Utilisation

L'animateur va, à partir du gestionnaire de conférence, déclencher sur l'ensemble des stations clientes, l'ouverture d'une fenêtre partagée dans laquelle sera déjà affiché le premier plan. Il va ensuite enchaîner les plans au moyen d'un menu dans lequel il choisira une option parmi les suivantes : plan suivant, plan précédent, premier, dernier. Ce menu évolue en fonction du contexte, ainsi, certaines fonctions ne sont pas toujours disponibles, comme par exemple l'option plan précédent lorsque l'on est sur le premier. Pendant cette phase, il dispose des télépointeurs du gestionnaire de conférence pour

préciser certains endroits particuliers de l'espace public. Par analogie, nous pouvons dire que le déroulement de cette phase avec l'enchaînement de ces plans correspond à l'utilisation d'un rétro-projecteur avec les commandes avant et arrière.

L'un des moments importants de cette présentation est l'attribution des rôles aux joueurs. Pour satisfaire cela, nous avons représenté sur l'écran le processus de fabrication avec les différents postes de fabrication ainsi que la liste des joueurs connectés. Cette dernière est acquise à partir du gestionnaire de conférence.

Nous avons souhaité rendre cette phase la plus pratique possible en choisissant de manipuler directement les noms des joueurs plutôt que de les rentrer par l'intermédiaire de boîtes de dialogue. La métaphore que nous pouvons en donner correspond au dessin des postes sur un tableau sur lequel on va coller des étiquettes comportant le nom des joueurs. La figure 3.8. présente ce plan.

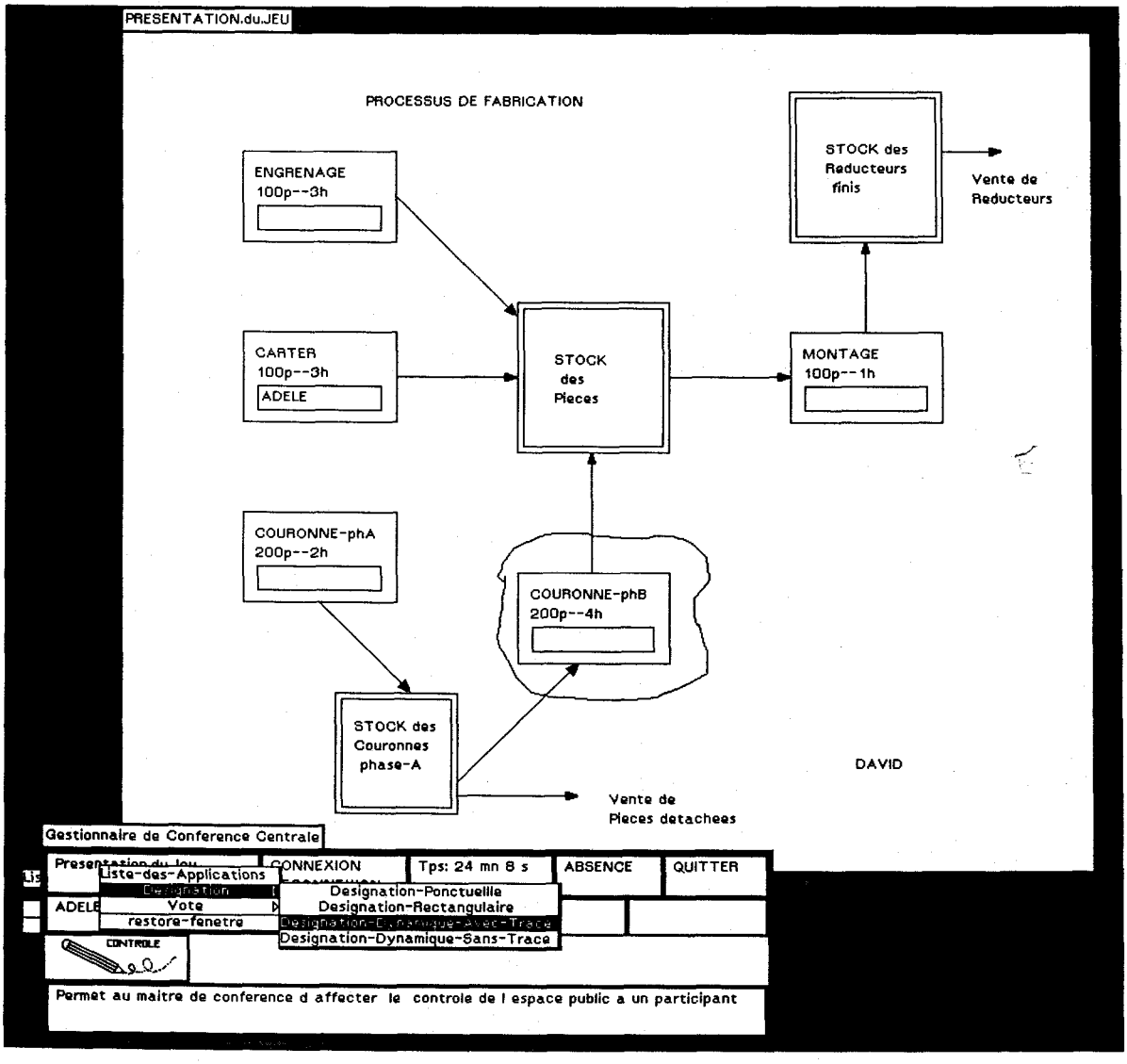


Figure 3.8. *Copie écran du plan utilisé pour l'attribution du rôle de chaque joueur (ou équipe).*

Pour affecter un poste à un joueur, l'animateur va, avec sa souris, cliquer sur le nom du joueur pour le sélectionner et le déplacer jusqu'au poste de production qu'il a choisi. Lorsque la sélection est faite, le nom est attaché au curseur de sa souris et remplace le curseur normal. Pour valider l'attribution du poste, il suffit de placer le nom à l'intérieur d'une case prévue à cet effet et d'appuyer sur le bouton de la souris. Cette attribution n'a rien de définitive car il peut toujours "reprendre" le nom et renouveler l'opération.

Sur les postes clients, les joueurs ne voient pas le déplacement dynamique du nom, mais uniquement la validation. C'est à dire qu'à la sélection, le nom disparaît de leur écran et qu'il réapparaît lors de l'affectation. Nous avons fait cela dans le souci de diminuer le flux d'informations sur le réseau. Cependant, il est tout à fait possible de réaliser l'affichage dynamique si le besoin s'en faisait vraiment sentir. L'opération est identique au télécurseur implémenté sur le gestionnaire de conférence.

Lorsque l'ensemble des postes a été pourvu et qu'il n'y a plus d'objection parmi les étudiants, l'animateur va valider l'attribution de ces rôles. Elle sera définitive et permettra de lancer automatiquement, pour les phases de simulation suivantes, les applications correspondant aux rôles des joueurs. En effet, les interfaces utilisateurs ont été personnalisées pour chacun des postes.

3.2.1.3 - Conclusion

Cette application est typique des applications W.Y.S.I.W.I.S. car elle nécessite un espace de travail public partagé par les joueurs et des références identiques pour l'ensemble du groupe.

Le rôle de l'animateur est en relation directe avec le gestionnaire de conférence implémenté. En effet, il est le seul, à un instant donné, à pouvoir modifier l'espace public. Il n'est pas envisageable que les joueurs puissent modifier chacun dans leur coin les plans à la vitesse qu'ils souhaitent. Pareillement à des transparents classiques, ces plans ne sont qu'un support de cours et doivent être accompagnés d'explications orales.

Les joueurs peuvent toujours intervenir pour demander à l'animateur de revenir sur un plan afin de préciser certaines choses. Si l'un des étudiants a besoin de désigner une partie de l'écran ou désire revenir lui-même sur certains plans, le contrôle peut éventuellement lui être donné par l'animateur, le temps qu'il s'explique.

3.2.2 - Définition du plan de production

3.2.2.1 - Présentation

Dans cette phase les joueurs ont besoin de partager un espace de travail commun qui, en l'occurrence, est le plan de production. Ce dernier doit être vu par tout le monde et les modifications effectuées doivent être diffusées immédiatement.

D'autre part, comme nous l'avons précédemment expliqué, l'animateur est la personne qui remplissait les cases du tableau en fonction des indications des joueurs mais qui pouvait également passer la craie à un joueur qui en faisait la demande.

L'interface que nous avons développée s'inspire fortement du tableau (sur feuilles papiers) qui est normalement distribué aux joueurs. Le résultat de son implémentation est montré par la figure suivante.

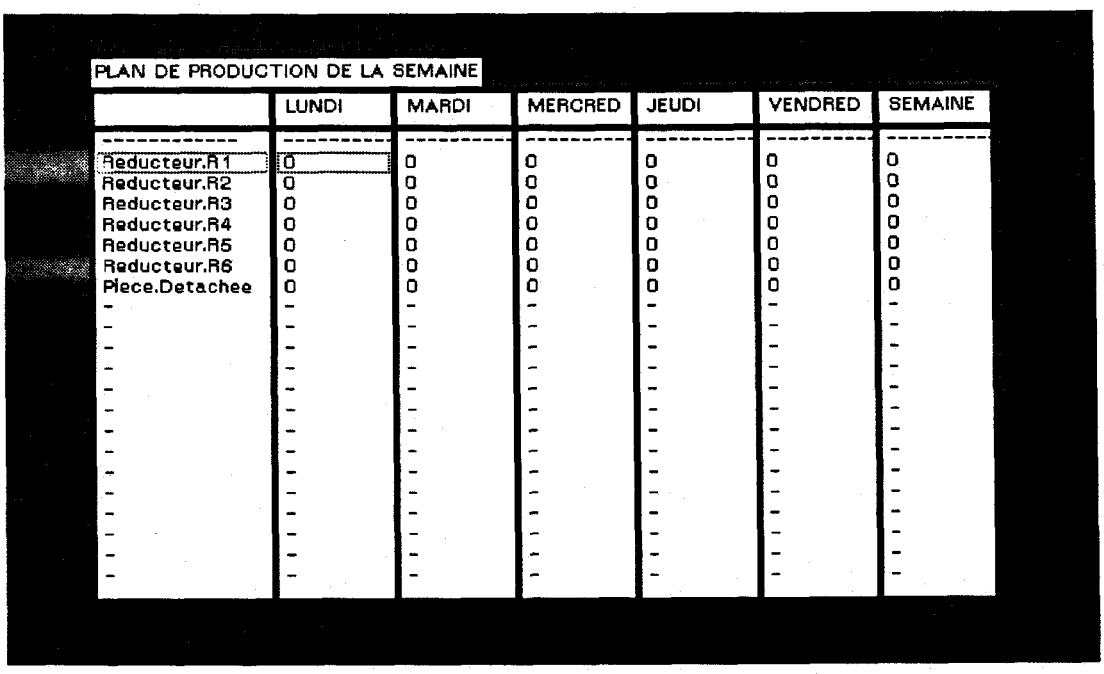


Figure 3.9. *Plan de production.*

Ces deux figures mettent en évidence les deux zones du tableau. En effet, ce dernier est un tableur dans lequel il est possible d'effectuer des opérations de calcul automatique lorsque l'un des champs a été modifié. La première zone correspond aux cases que les joueurs vont devoir remplir ; il s'agit là de leur objectif général. La seconde est destinée à l'affichage de la production induite par les valeurs programmées. Dans la version classique, les calculs s'effectuent à la main à partir de feuilles spéciales et il s'agit tout simplement d'apporter ici, un "plus" informatique.

L'animateur peut décider de montrer ou de cacher cette seconde zone. Il dispose d'une option menu qui lui permet un affichage partiel ou total de ce tableau. Il semble que débuter par l'affichage des seules valeurs de productions des réducteurs soit moins perturbateur. Cependant, l'affichage du tableau en entier permet une prise de décision plus rapide car les conséquences directes pour les postes de production sont visibles sur un simple coup d'oeil.

L'animateur dispose d'informations techniques et commerciales qui doivent être divulguées si les participants en font la demande expresse. Nous les avons représentées sous forme d'histogrammes accessibles dans une fenêtre par menus. La figure suivante montre la représentation de ces informations.

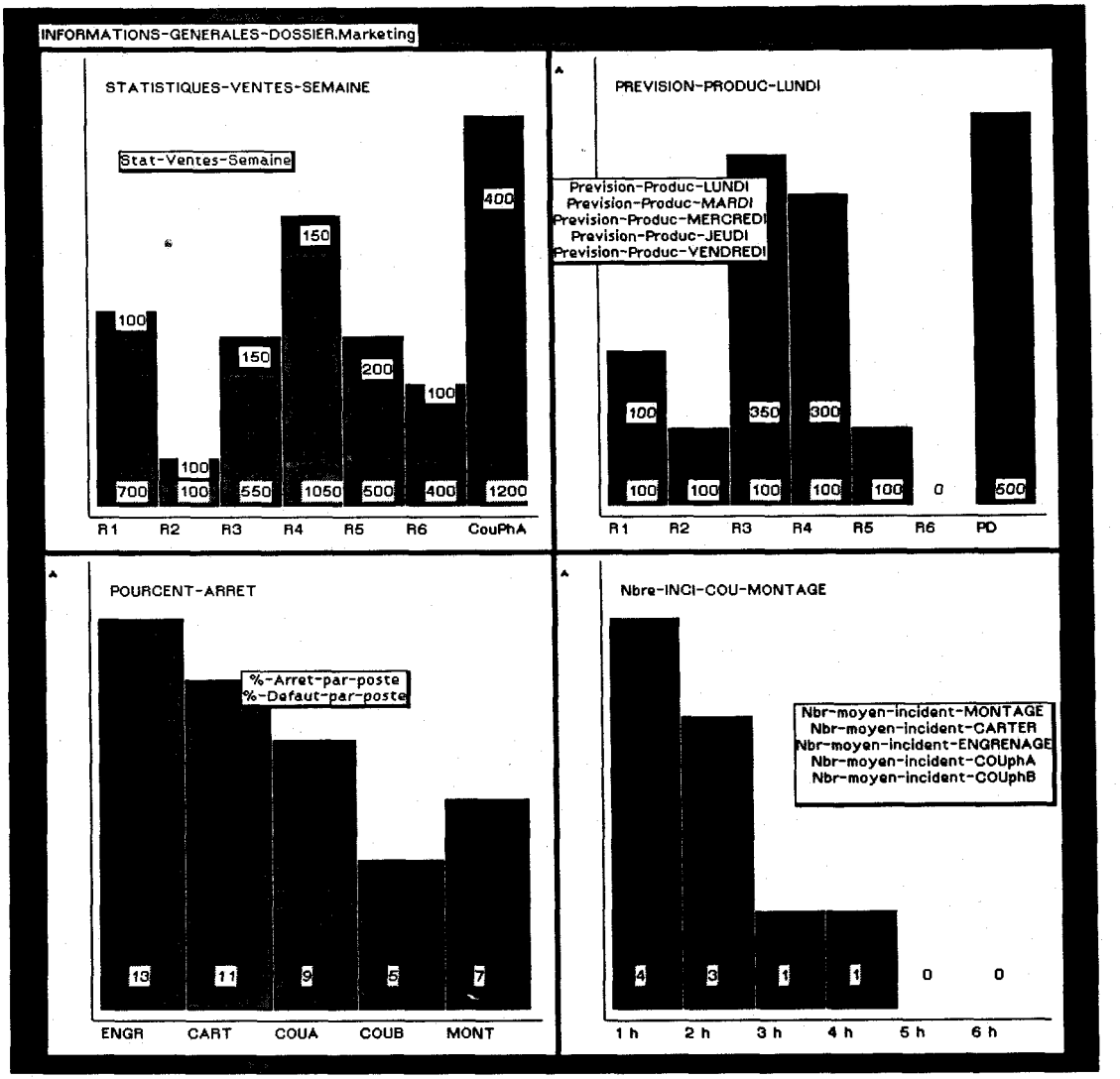


Figure 3.10. *Exemple d'une copie d'écran relative aux informations fournies aux joueurs lorsqu'ils en font la demande explicite.*

3.2.2.2 - Utilisation

Au démarrage, un tableau vierge est ouvert sur l'ensemble des stations des participants.

L'animateur va collecter pour une case, les valeurs proposées par les autres joueurs. Il pourra ensuite les évaluer successivement et les commenter ou les faire commenter par les joueurs. Par un affichage total du tableau, les conséquences directes sur les postes de production pourront être appréciées. C'est-à-dire que la visualisation du nombre de pièces correspondant aux réducteurs programmés sera faite.

Comme nous l'avons expliqué, le menu de l'animateur et ceux des joueurs sont différents. L'animateur est le seul à pouvoir évaluer et placer une des valeurs de la liste. Cette liste est donc constituée des propositions faites par les joueurs. Si les participants n'arrivent pas à se mettre d'accord, ils peuvent utiliser le système de vote mis à leur disposition pour trancher.

Il arrive qu'une personne veuille disposer des fonctionnalités pour assurer le rôle de "meneur de jeu". L'animateur et le joueur peuvent par l'intermédiaire du gestionnaire de conférence effectuer ce changement.

Lorsque l'animateur (ou le meneur de jeu) demande l'avis des autres pour une valeur particulière, le reste du groupe sait toujours sur quelle case la discussion porte car elle est alors représentée en vidéo inverse. Pour cela aussi, il n'y a que la seule personne contrôlant l'espace public qui peut, par sa souris, modifier la case sélectionnée.

Lorsque l'un des participants demande à l'animateur s'il dispose d'informations complémentaires pour prendre ses décisions, ce dernier pourra les lui donner en débloquant une option menu. Leur menu comportera alors une option supplémentaire leur permettant d'effectuer l'ouverture de la fenêtre d'information présentée par la figure 3.11. Il ne s'agit pas dans ce cas d'une ouverture générale et identique pour tout le monde, mais d'une possibilité de consultation indépendante pour les joueurs. En temps normal, cette information n'est pas diffusée en une seule fois mais de façon successive en fonction des demandes. Comme nous avons représenté cette information dans une seule fenêtre, tout est ici accessible en une seule fois. Pour ne donner que l'information demandée, il aurait fallu non pas une fenêtre avec quatre sous-fenêtres, mais quatre fenêtres différentes dont l'ouverture aurait pu être déclenchée à des instants différents.

Les figures suivantes sont des exemples de ce que l'on peut voir sur des écrans de participants.

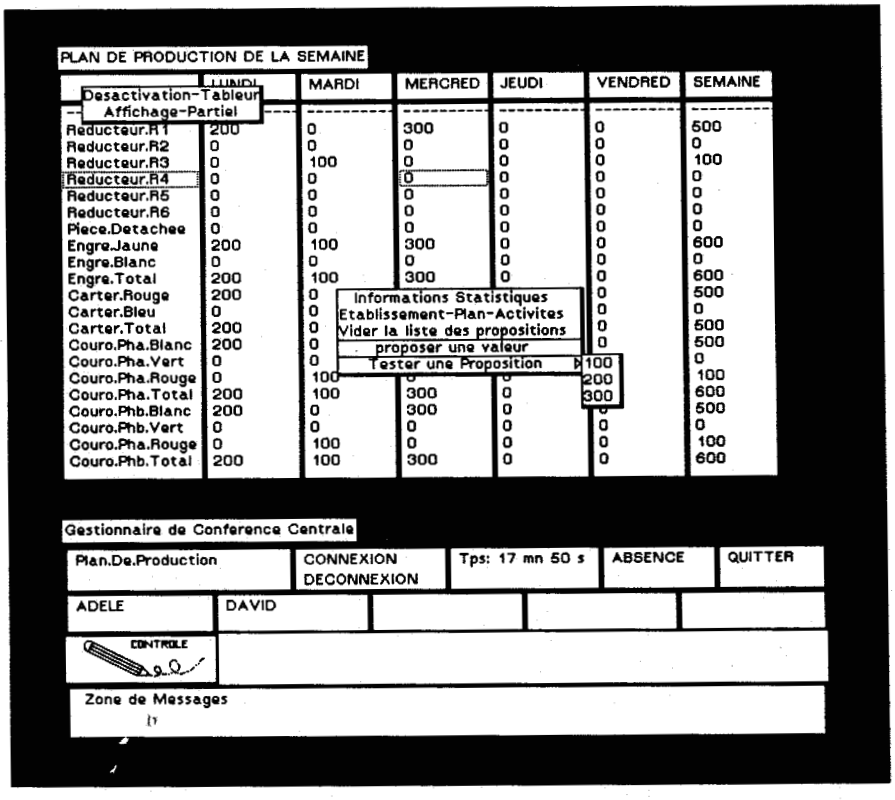


Figure 3.11. *Exemple de l'écran de l'animateur pendant la phase du jeu.*

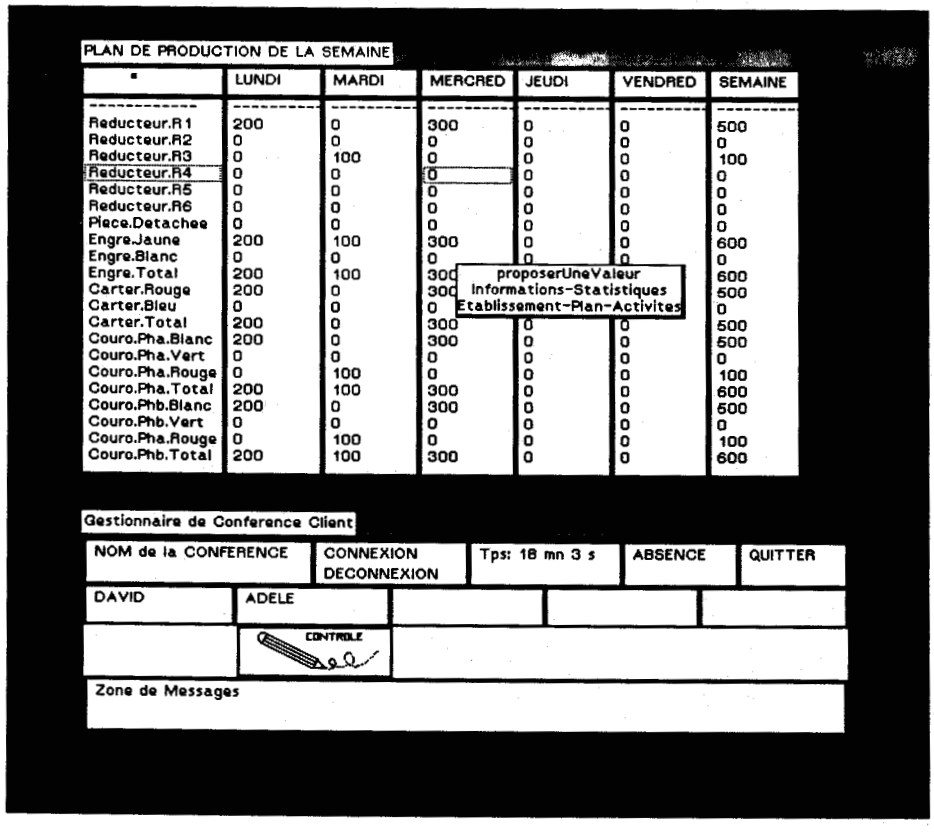


Figure 3.12. *Exemple de l'écran d'un des joueurs pendant la phase du jeu (pas de possibilité pour lui de modifier le tableau).*

Lorsque ce tableau est complet, il faut que chaque joueur en déduise le plan d'activité de son poste de production. Pour qu'il puisse le concevoir, l'animateur va de nouveau modifier leur menu de façon à pouvoir accéder à un éditeur spécifique et adapté aux références de chacun des postes.

Cet éditeur se présente sous la forme suivante :

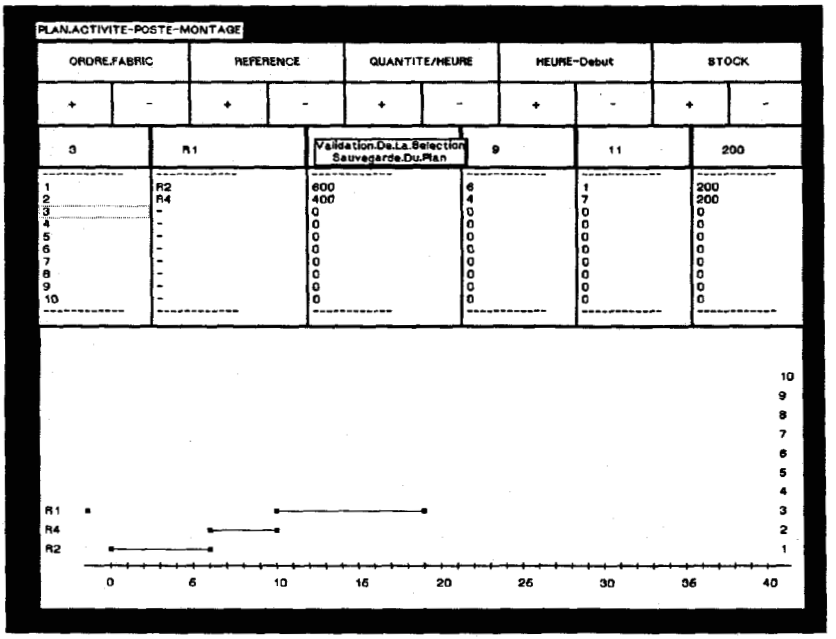


Figure 3.12. *Editeur spécifique du plan d'activité.*

Les joueurs n'ont pas à entrer de valeurs par l'intermédiaire du clavier, mais simplement par la sélection de cases, avec leur souris. S'ils souhaitent définir la quantité de produits pour une référence donnée, il leur suffit de l'entrer en cliquant sur les cases "plus" et "moins". En un mot, on choisit le paramètre que l'on veut modifier et on le spécifie par l'utilisation de compteur.

Pour une meilleure compréhension, nous avons choisi de représenter les informations sous deux formes différentes : d'une façon classique, par un tableau, et d'une façon graphique, par un planning. Ces informations sont mises à jour dynamiquement sur ces deux média. Le graphique est très pratique, car il permet de ne pas se tromper et évite en particulier les erreurs de chevauchement de production. Comme nous pouvons le voir, entre l'heure de début et de fin de production, un problème peut facilement survenir.

Lorsque le participant juge que son plan d'activité est satisfaisant, il le sauvegarde pour l'utiliser dans la phase de simulation suivante. S'il s'avère que son plan n'est pas

bon, il peut réouvrir son éditeur, le modifier, et le sauvegarder. Lors de la sauvegarde, un petit tableau récapitulatif est ouvert pour vérification, avec l'ensemble des paramètres que le participant a rentré.

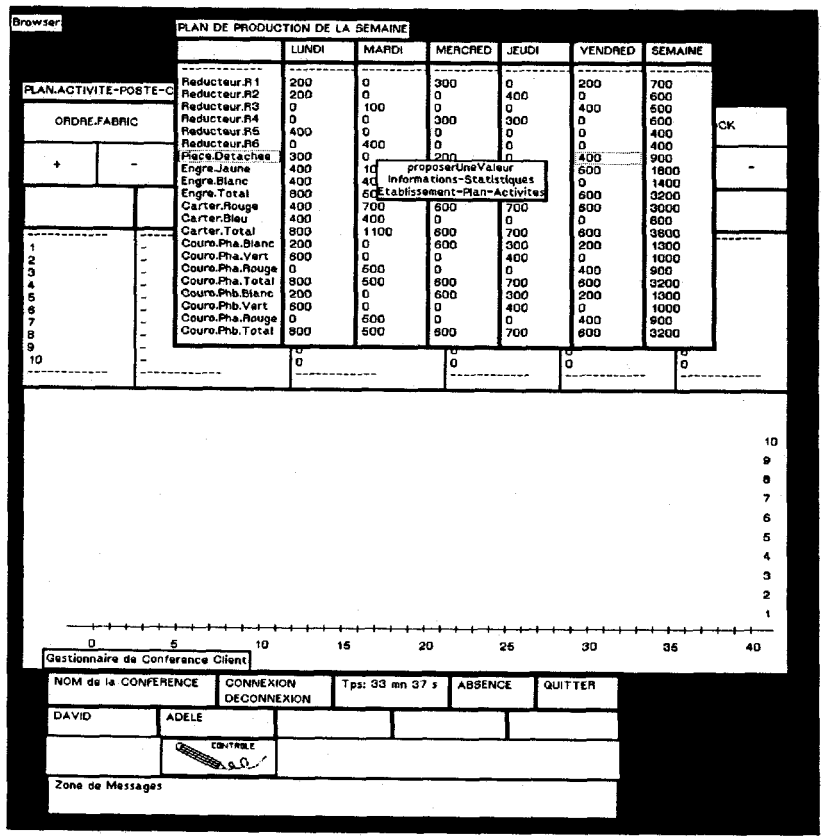


Figure 3.13. *Copie de l'écran d'un des joueurs au moment de la réalisation du plan d'activité.*

3.2.3 - Méthode de planification MRP

3.2.3.1 - Les besoins

Cette phase de simulation se déroule à partir des plans d'activités que les joueurs ont établis et qu'ils doivent respecter quels que soient les aléas introduits par les cartes incidents.

Cette phase est différente des deux précédentes, car il n'y a pas besoin ici d'un espace public de type W.Y.S.I.W.I.S.. En effet, bien que les participants partagent certaines données, les décisions qu'ils prennent le sont individuellement. Il n'y a donc plus, à ce niveau du jeu, de décision collective sur la base d'un support visuel commun.

La collaboration de chacun est tout de même primordiale car l'objectif ne peut être atteint que si tout le monde réalise la tâche qui lui a été assignée. Par exemple, si l'un des joueurs ne valide pas une action, l'animateur ne pourra pas passer à l'heure suivante.

Le choix d'un espace de travail W.Y.S.I.W.I.S. ou non est un problème délicat à formaliser car il dépend fortement des rôles, des tâches à remplir et plus particulièrement des activités de chacun induites dans l'application. Après analyse de cette phase du jeu, il est apparu qu'un point de vue sur l'ensemble des données partagées était suffisant pour parvenir au résultat. Ainsi, bien que la décision finale sur la production de leur poste leur appartiennent, les joueurs peuvent continuer à discuter entre eux et à donner leur avis sur la conduite à tenir. Ils auront par l'intermédiaire de cet environnement partagé, la possibilité de voir l'évolution de l'ensemble du jeu.

Les données partagées sont les trois stocks de pièces, l'heure et les cartes d'incidents et d'informations.

Sur l'interface utilisateur, différentes informations doivent être accessibles.

. On doit tout d'abord y trouver une "feuille d'activité" pour rendre compte des actions accomplies à chaque heure.

. Pour les discussions, une vue sur l'état des stocks est nécessaire.

. A chaque changement d'heure, un tirage aléatoire est déclenché et une carte incident est affichée en fonction du résultat sur les différents postes. L'affichage des cartes d'informations doit être également accessible.

. Le participant doit pouvoir lire son plan d'activité afin de le suivre.

L'interface que nous avons réalisée est la suivante :

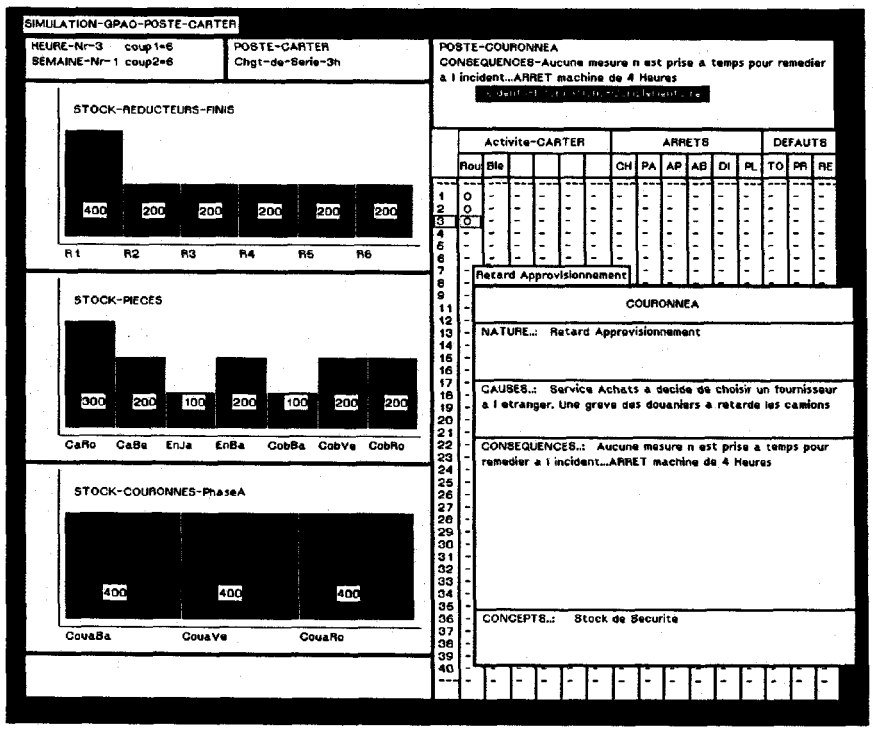


Figure 3.14. *Exemples de l'interface du poste Carter pendant la phase MRP. La figure de droite montre les différentes informations accessibles dans d'autres fenêtres par le joueur.*

Cette interface est divisée en 6 zones :

1. La première zone indique l'heure et la semaine courante, ainsi que le résultat du tirage aléatoire permettant de déterminer la carte incident.

2. La seconde affiche le nom du poste, ainsi que le temps de changement de série entre deux références. Diverses informations générales y sont accessibles par des menus.

3. Dans cette sous-fenêtre, on retrouve un résumé de la carte incident et de la carte information. Des options menu permettent d'ouvrir une fenêtre complète.

4. Elle affiche, à partir des données, l'état des stocks sous forme de barres. Une opération du menu permet d'ouvrir une fenêtre affichant l'ensemble des produits avec leur coût.

5. Cette zone permet l'affichage de messages d'information du système pour guider l'utilisateur lors de manipulations incorrectes.

6. Cette dernière zone représente la feuille d'activité personnalisée de chaque poste.

3.2.3.2 - Utilisation

Le stock de pièces initial, que les joueurs ont défini dans la phase précédente, est affiché au démarrage. Nous avons fait le choix, pour cette méthode de planification de ne pas manipuler directement les jetons, mais de valider les opérations à partir de la feuille d'activité. C'est à dire que le joueur va valider une option de son menu après avoir sélectionné la case voulue. La zone d'affichage des stocks courants n'est plus qu'une représentation des données et des valeurs physiques.

Le système implémenté permet d'effectuer certaines opérations de vérification sur cette feuille. Il ne peut, par exemple, y avoir qu'une case de validée par ligne et cette validation doit être obligatoirement effectuée sur la ligne correspond à l'heure courante. Un contrôle est également effectué sur la cohérence d'une action. Si le poste "montage", fabrique un réducteur, dont au moins une pièce n'est pas disponible, un message dans la zone 6 sera affiché et l'action ne sera pas validée.

3.2.3.3 - Activité

Les cinq équipes vont à chaque changement d'heure, valider leur choix et modifier l'état des stocks d'une façon asynchrone. Les nouvelles valeurs des stocks seront diffusées immédiatement à l'ensemble des joueurs.

A chaque validation d'une équipe, l'animateur sera averti et il passera à l'heure suivante lorsque tous les joueurs auront accompli leur tâche. Le passage à l'heure suivante déclenchera automatiquement le tirage aléatoire d'un dé, et d'un second si nécessaire, pour déterminer la conséquence.

Ne disposant pas pour l'instant d'un nombre de machines suffisant, nous avons implémenté l'interface de l'animateur sur l'un des postes de production. Ainsi, sur l'une des stations, un utilisateur joue deux rôles.

A la fin de cette phase de simulation, l'animateur fait le bilan des stocks, etc, d'une manière identique au jeu classique. Comme toujours, la richesse du jeu découle des discussions qui se sont enchaînées tout au long de la simulation.

3.2.4 - Méthode de planification KANBAN

3.2.4.1 - Les besoins

A partir de cette phase, les joueurs ont la possibilité de choisir les changements de séries à chaque heure en fonction de l'état de leur planning. Le type d'informations à distribuer reste identique à celui de la phase précédente, mais un changement a été fait dans sa représentation et dans sa manipulation. Nous ne représentons plus directement les "jetons", mais les kanbans.

Par exemple, l'interface du poste carter pour cette phase de simulation est la suivante :

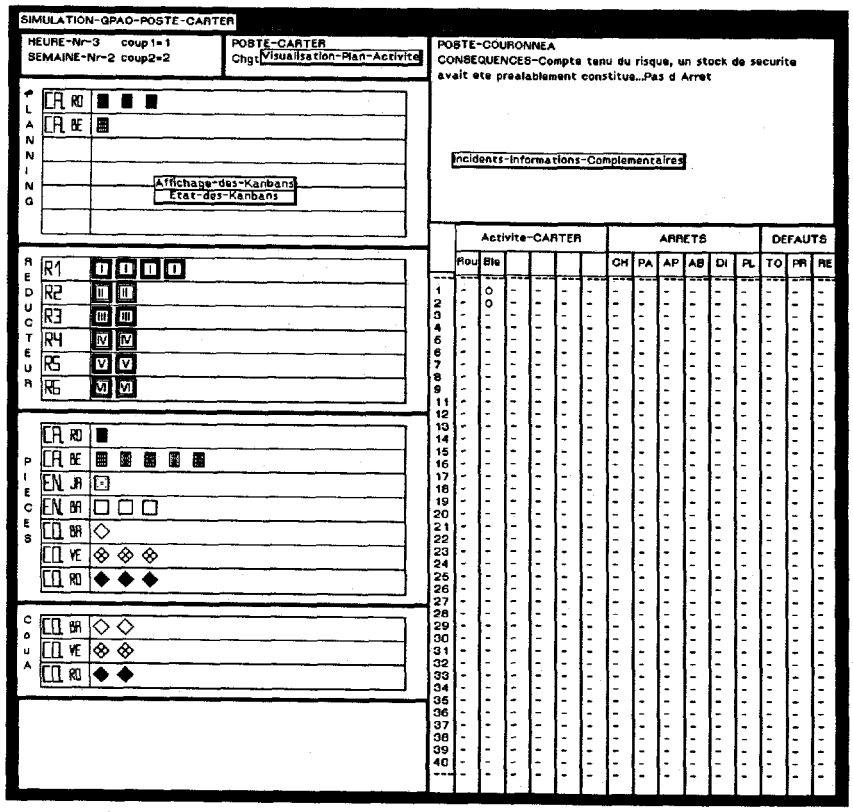


Figure 3.15. *Exemples de l'interface du poste Carter pendant la phase KANBAN. La figure de droite montre les différentes informations accessibles dans d'autres fenêtres par le joueur.*

L'interface est toujours divisée en 6 zones avec des fonctionnalités qui restent identiques à celles de l'étape précédente. Cependant, dans la dernière zone a été ajouté le planning Kanban du poste concerné.

3.2.4.2 - Utilisation

Dans le souci d'étudier différents types d'interactions, nous avons modifié par rapport à l'interface utilisée dans l'étape précédente, la validation des tâches effectuées à chaque heure.

Lorsque le joueur d'un poste veut produire une référence, il ne va plus la valider sur sa feuille d'activité, mais va le faire directement sur les plannings au moyen de sa souris. Il lui suffit de se placer sur le dernier Kanban de la référence choisie, de le sélectionner, en "cliquant" dessus avec le bouton gauche de la souris, et de l'amener à la place correspondante dans le stock. Si la place de destination n'est pas correcte, le kanban retourne à sa position initiale et l'opération n'est pas validée (un message dans la zone de message avertira le joueur). Dans le cas favorable, sa feuille d'activité sera alors mise à jour automatiquement et l'information sur l'état des stocks diffusée à l'ensemble des postes.

Les activités correspondant aux défauts et aux arrêts doivent toujours être validées directement sur la feuille d'activité, comme pourrait le faire l'ouvrier, derrière sa machine.

Le poste montage est particulier dans le jeu car il faut pour valider un Kanban correspondant à un réducteur que les pièces le constituant soient disponibles. Au cas où une ou plusieurs pièces manqueraient, la validation ne pourrait être faite et un message indiquant que les pièces ne sont pas en stock sera affiché.

L'animateur n'a pas de planning dans son interface, mais il replacera automatiquement les kanbans sur les plannings des postes montage et couronne phaseA, lors des ventes de réducteurs et de pièces détachées.

3.2.4.3 - Activité

L'activité est plus intense du fait que les décisions sont désormais possibles et peuvent être discutées par le groupe. La phase de synchronisation pour le passage à l'heure suivante est toujours effectuée par l'animateur qui s'assure que l'ensemble des joueurs a joué. Ici aussi, nous avons, pour chaque poste, une interface différente avec des points de vue différents sur des données partagées. L'ensemble complet des fonctionnalités offertes pour établir la conclusion de la semaine est à la disposition du groupe.

3.2.5 - Conclusion

Sur les quatre phases de cette simulation, deux sont caractéristiques des applications W.Y.S.I.W.I.S.. La gestion des activités des participants est directement liée au gestionnaire de conférence du système. Il existe alors les deux types d'inter-actions liées au statut de l'utilisateur (contrôleur ou non de l'espace publique).

Le gestionnaire, que nous avons implémenté, s'est avéré tout à fait adapté aux rôles des différents intervenants dans la simulation. Pendant les phases W.Y.S.I.W.I.S., autoriser des activités parallèles n'aurait eu aucun sens, de même qu'attribuer un contrôle partiel sur des sous-ensembles du tableau. En effet, nous aurions pu nous arranger pour que chaque participant soit chargé de réaliser le plan sur une colonne, c'est à dire, pour un jour de la semaine, mais le résultat n'aurait certainement pas été probant.

L'une des règles que l'on tire de cela, est qu'il ne faut pas essayer d'offrir aux utilisateurs des conditions de travail inhabituelles, c'est à dire que l'on ne retrouve pas dans des réunions classiques (sans ordinateur).

3.3 - Architecture de l'application

L'architecture globale de l'application se décompose en deux modèles différents.

Le premier correspond à des applications coopératives W.Y.S.I.W.I.S. qui nécessitent un espace de travail public, comme dans les deux premières phases de la simulation. Ce type d'application implique un partage complet des données qui ne se résume pas simplement à une gestion des accès concurrents, mais à un accord préalable entre les participants sur les valeurs à affecter. C'est le choix de ces valeurs qui nécessite la coordination entre les différents individus impliqués dans la décision. A partir de nos expériences, nous avons remarqué qu'il fallait qu'une seule personne puisse à un instant donné, modifier les données en accord avec le groupe. Le gestionnaire de conférence, que nous avons implémenté, propose ce protocole de gestion de l'espace public.

Le second cas de figure correspond à des applications coopératives qui partagent un contexte, mais qui ne nécessitent pas pour autant une interface W.Y.S.I.W.I.S.. Les décisions de modification des données qui sont soit partagées (plusieurs personnes peuvent les modifier et elles ont donc un mécanisme de protection) soit réservées (une seule personne y a accès) sont prises individuellement. Avec ce type d'application, les participants ont chacun un rôle qui est souvent différent de celui des autres. Le gestionnaire de conférence n'est pas adapté à cette distribution des rôles car ceux-ci seront toujours différents suivant les applications. On ne pourra donc pas en déduire ici un comporteùent ni même un processus de décision.

3.3.1 - Architecture du plan de production

Comme nous venons de l'expliquer, le tableau que les participants doivent remplir va être le support qui sera partagé par le groupe.

3.3.1.1 - Besoins

Nous souhaitions offrir des facilités supplémentaires pour réaliser le plan de production. Cela s'est traduit dans l'environnement informatique par des fonctionnalités communes au tableur. Nous en avons donc développé un, en Smalltalk, qui s'utilise de façon tout à fait classique.

Une différence existe cependant : la sélection d'une case ne se fait qu'au moyen de la souris et se matérialise par un affichage en vidéo inverse. L'initialisation, ou la

modification d'une case qui peut être, soit une valeur, soit une formule, se fait par l'intermédiaire de boîtes de dialogue.

La première opération à réaliser a consisté à mettre à jour la boîte à outils car nous ne disposions pas, pour réaliser ce tableur partageable, de couples vues/contrôleurs qui satisfassent nos besoins. Il est constitué, en partie, de fenêtres listes particulières, dépendantes les unes des autres, qui n'autorisent la sélection d'un item que dans une seule d'entre elles. La sélection d'un nouveau item désélectionne automatiquement le précédent.

La démarche utilisée pour la "coopératisation" de cette vue correspond à celle exposée dans le chapitre précédent. Il s'agit de réécrire les méthodes qui en modifient "l'apparence".

Il y a modification, entraînant une diffusion de l'information, lorsque :

. une case est sélectionnée ou désélectionnée. La sélection doit apparaître aussitôt sur l'ensemble des stations ;

. une valeur est modifiée (lorsqu'une valeur est changée, elle entraîne la mise à jour des cases dont les formules y font référence, la diffusion est effectuée lorsque les valeurs ont été recalculées).

L'utilisation collective de ce tableur est liée au gestionnaire de conférence, c'est-à-dire qu'une personne, à un instant, donné possède des droits différents des autres joueurs. Ces droits, au niveau de l'application, sont liés au statut de "meneur de jeu temporaire". Avec le schéma que nous proposons, son rôle consiste à obtenir le consensus des joueurs sur une valeur particulière. Il sera alors le seul autorisé à valider et à évaluer les valeurs que les joueurs auront proposées. En termes d'implémentation, ces droits particuliers, en fonction des rôles, deviennent des options menu différentes.

Rôle	Meneur de Jeu	Les Joueurs
menu (bouton du milieu)	proposer une valeur évaluer une valeur	proposer une valeur
bouton gauche	sélection d'une case possible	sélection d'une case impossible

Les menus sont un peu plus compliqués, car ils évoluent en fonction de divers paramètres. Lorsque l'animateur l'autorise, les joueurs ont des options supplémentaires qui leur permet d'ouvrir une fenêtre avec les statistiques des ventes et des pannes.

3.3.1.2 - Modèle de l'architecture

Le schéma de développement de l'application est toujours construit à partir de la triade M.V.C.. Le modèle de l'application est implémenté sur le serveur et il est partagé par l'ensemble des stations qui disposent d'un objet mandataire. La mise à jour des vues est faite par les couples vue/contrôleur.

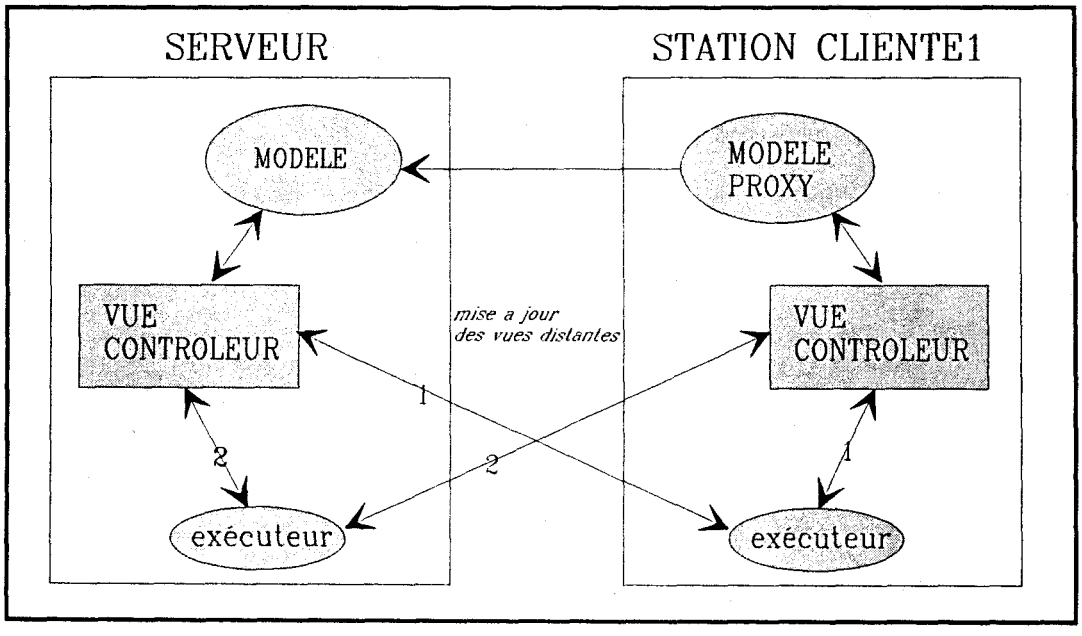


Figure 3.16. *Architecture de l'application*

La figure 3.16 représente l'architecture globale de l'application. Les mises à jour des vues sont toujours assurées par des objets "exécuteurs" référencés par le gestionnaire de conférence. La diffusion des messages se fait toujours par l'intermédiaire du serveur qui se charge de la transmission.

Nous nous sommes intéressés au problème de la répartition de code entre le serveur, qui assure la cohérence, et les stations clientes qui pourraient en traiter une partie. L'un des exemples que nous avons abordé dans le chapître précédent correspond à l'ouverture des fenêtres statistiques, lorsque l'option menu est disponible. C'est aussi le cas de l'éditeur spécifique, prévu pour la définition du plan d'activité.

Pour ces deux fonctions, nous utilisons un mécanisme qui permet d'exécuter le message correspondant à l'option menu sélectionnée, par le contrôleur, et non par le modèle. Il s'agit en quelque sorte d'un court-circuit qui évite ainsi l'envoi d'un message au modèle et un autre en retour à l'objet "exécuteur" local. Dans ce cas, le modèle n'en sera pas averti mais cela ne pose aucun problème de cohérence, car ces deux options sont utilisables d'une façon asynchrone par les étudiants.

Un autre exemple, que nous avons expliqué précédemment, concerne la proposition d'une valeur pour une case du tableur. Ici aussi, nous pouvons gagner un appel/réponse, en traitant d'abord localement la méthode, et en envoyant seulement après, la valeur acquise. La cohérence sera toujours faite par le modèle partagé.

3.3.1.3 - Conclusion

Nous ne disposions pas dans notre boîte à outils d'un couple vue et contrôleur qui satisfasse nos besoins. La première opération a alors consisté à implémenter ce couple avec les mises à jour distantes que nous souhaitions en fonction des actions utilisateurs. A partir de cela, l'implémentation du plan de production ne pose plus de problème car elle permet de travailler d'une façon classique.

3.3.2 - Architecture des phases de simulation

3.3.2.1 - Besoin

Pendant le déroulement des phases de simulation, les joueurs sont responsables de leur poste. Ils ont chacun une tâche à remplir, et des décisions personnelles à prendre. Pour cela, ils ont besoin d'avoir une vision globale de l'environnement partagé. Il s'agit plus particulièrement dans notre cas, d'avoir une vue sur l'état des stocks et de voir en temps-réel son évolution en fonction des productions et des prélèvements de chacun.

Les jeux de rôle des participants dans ces deux phases ne sont pas adaptés au gestionnaire de conférence. Il faut donc implémenter ces fonctionnalités au niveau applicatif.

Chaque station possède maintenant son propre modèle, mais partage des données communes. Sur chaque station, on trouve les valeurs dupliquées de ces stocks qui sont mises à jour dès qu'un joueur les modifie. Pour réaliser les mises à jour, nous utilisons le gestionnaire de conférence et en particulier les objets que nous avons appelés "exécuteurs".

3.3.2.2 - Modèle de l'architecture

Le schéma de l'architecture est le suivant :

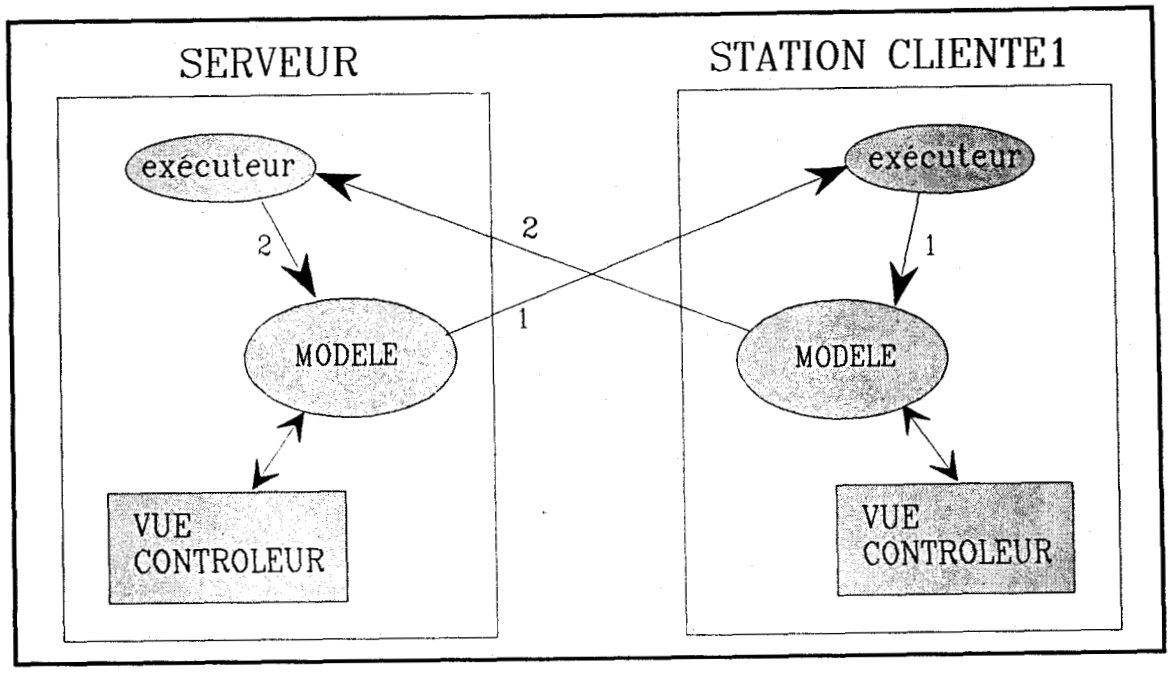


Figure 3.17. *Schéma de l'architecture. La diffusion des messages se fait par l'intermédiaire des objets "exécuteurs".*

Les communications ne sont pas directes entre les modèles mais transitent par les différents objets "exécuteurs" des stations.

Dans cette application, des messages entraînant une diffusion sont échangés dans les occasions suivantes :

. lorsqu'un joueur modifie les stocks.

. lorsque l'animateur passe à l'heure suivante (entraînant le tirage d'une carte incident).

Le gestionnaire de conférence implémenté sur le serveur possède entre autres une variable d'instance qui pointe sur la liste des clients connectés (liste de noms). Une station cliente a une de ses variables d'instance qui pointe sur le nom du serveur. Les objets "exécuteurs" sont des objets partagés dont on peut avoir l'accès par leur identificateur et ce par l'intermédiaire du "service de nom". Au moment de l'initialisation nous avons choisi, leur nom comme identificateur des objets "exécuteurs" sur chaque station.

Ainsi, accéder et diffuser des messages aux objets "exécuteurs" sur toutes les stations devient simple. L'exemple suivant correspond au code que l'on doit utiliser pour envoyer un message à toutes les stations clientes.

> listeDesClientsConnectés do: [:nomClient | (Serveur name: nomClient) messageAEnvoyer: unParamètre]
>
> listeDesClientsConnectés est une variable d'instance du gestionnaire qui pointe sur la liste des personnes connectées
>
> (Serveur name: nomClient) va renvoyer le proxy correspondant à l'identificateur nomClient
>
> Ainsi cette méthode va successivement envoyer le message "messageAEnvoyer:" avec le paramètre "unParamètre" à toutes les stations dont le nom figure dans la liste.

Nous nous apercevons donc que, tout en n'étant pas adapté au rôle des participants, le gestionnaire de conférence permet cependant de résoudre les problèmes de diffusion.

Dans le souci de diminuer le flux d'informations entre les stations, il est important d'étudier la nature des messages échangés.

En effet, ces derniers peuvent être unaires (un sélecteur, mais pas d'argument) ou composés d'un sélecteur et d'un ou plusieurs arguments. La sérialisation nécessaire des messages avec le B.O.S. (Binary Object Storage), pour la diffusion sur le réseau, concerne le sélecteur et les arguments. Le sélecteur du message n'est qu'une chaîne de caractères (plus précisément, un symbole), mais les arguments peuvent être n'importe quel objet. Pour minimiser la taille du message, il n'est pas intéressant de réduire la taille du sélecteur, mais il est plus avantageux d'étudier les objets que l'on va passer en paramètres.

Dans les deux phases de simulation, l'animateur déclenche automatiquement, à chaque changement d'heure, le tirage aléatoire d'une carte d'incident. Il faut donc diffuser à chaque joueur, les informations suivantes :

. Le passage à l'heure suivante.

. La carte incident tirée avec l'indice de conséquence.

. La carte d'information.

Ces deux types de cartes sont définis comme des objets "carteIncident" ou "carteInformation" respectivement instances des classes CarteIncident et CarteInformation. Elles seront ensuite utilisées pour un affichage dans des fenêtres adaptées. L'ouverture de celles-ci se fait par un message de classe auquel on passe un

objet "carte" comme paramètre.

Nous avons sauvegardé une vingtaine de ces cartes dans un dictionnaire référencé par une variable globale. L'intérêt d'un objet "dictionnaire" est qu'il permet d'accéder à l'objet par, ce que l'on appelle, une clé (généralement un symbole).

Pour diminuer la taille des messages, il faut choisir ce qui peut être dupliqué ou non. Dans ce cas particulier, si rien n'est dupliqué, le message à envoyer aux joueurs sera le suivant :

```
incrémentationHeure: uneHeure carteIncident: uneCarteIncident conséquence: unNombre
```

On s'aperçoit alors que l'on est obligé de passer l'objet "carte" comme paramètre.

Il peut être plus avantageux de donner uniquement la clé d'accès, si les données ont été dupliquées sur toutes les stations. Dans ce cas, nous pouvons également gagner le paramètre "uneHeure" car nous déduisons qu'à chaque réception de ce message, l'incrémentation est d'une unité. Le message devient alors :

```
incrémentationHeure: cléDeLaCarte conséquence: unNombre
```

Il est important de bien étudier ce qui peut être dupliqué ou non. Une duplication importante des données est néfaste sur le plan de la confidentialité, mais permet un allègement de la bande passante en ne transmettant que les identificateurs et non les objets. Pour que la gestion des ces données ne soit pas trop lourde, il ne faut pas que leurs modifications soient fréquentes.

A travers cet exemple nous nous apercevons qu'il y a lieu d'établir à nouveau un compromis entre ce qui doit être dupliqué, et ce qui ne doit pas l'être. Il faut également étudier l'enchaînement des appels de procédures distantes afin de les réduire au maximum.

3.3.2.3 - Conclusion

Dans cette phase de simulation, l'utilisation d'espace partagé de type W.Y.S.I.W.I.S. n'a pas été nécessaire. Les joueurs disposent sur leur station d'une interface adaptée au poste et au rôle qu'ils occupent dans le jeu.

L'échange d'informations entre les différentes stations permet cependant de garder la cohérence globale de l'application. Le problème du contenu des messages doit être particulièrement étudié car les performances des mises à jour de l'interface utilisateur en dépendent.

3.3.3 - Conclusion sur l'architecture de l'application

Cette application coopérative que nous avons étudiée et implémentée est constituée de quatre phases différentes.

Les deux premières nécessitent des espaces de travail publics de type W.Y.S.I.W.I.S. alors que les deux autres n'utilisent qu'un partage de données.

Ce qui différencie les deux types d'interfaces est explicable en partie par les tâches assignées aux joueurs. Dans le premier cas, ils ont tous la même tâche à accomplir. Leurs décisions sont collectives, et ils doivent parvenir à un accord entre eux pour modifier l'espace de travail public (le tableau du plan de production). Le gestionnaire de conférence, que nous avons implémenté, permet de gérer les protocoles d'interactions entre les joueurs, d'une façon satisfaisante.

Pour les phases de simulation, chacun des joueurs remplit une tâche particulière. Bien que l'ensemble de ces tâches doit être coordonné pour parvenir à l'objectif fixé, les décisions sont prises individuellement. Le gestionnaire de conférence proposé n'est pas adapté à ce type de coopération. En effet, les rôles des joueurs sont toujours différents et sont liés à l'applicatif. C'est donc au niveau de l'application que les protocoles doivent être implémentés par des fonctionnalités accessibles ou non en fonction du contexte global.

Quel que soit le modèle architectural proposé, la diminution du trafic sur le réseau est un des problèmes fondamentaux du Groupware. En effet, de ce problème dépend la vitesse de mise à jour des interfaces car plus les temps de réponse seront courts, plus le travail du groupe sera efficace et cohérent.

Pour résoudre cela, une étude de la répartition du code est nécessaire. Il faut que l'exécution des méthodes entraîne un minimum d'appel/réponse de procédures distantes. En mettant les applications au point, on s'aperçoit rapidement que ce n'est pas le temps de traitement qui est pénalisant, mais le temps de mise en forme et de récupération des messages.

Un des points positifs de Smalltalk est que le modèle de développement des applications interactives proposé permet d'emblée, un partage de code efficace. Par exemple, les opérations générales sur les fenêtres sont gérées localement par les contrôleurs.

Ce qui peut faire diminuer également les temps de mises à jour est la taille des messages que l'on va envoyer. Jouer sur le nombre de caractères du sélecteur de message n'est pas réellement bénéfique, mais réduire la taille des objets passés en paramètre, l'est. Réaliser cela oblige alors une duplication des données avec les problèmes que cela pose.

Pour construire le tableur partageable, nous avons dû définir de nouveaux couples vue/contrôleur qui n'étaient pas disponibles dans notre boîte à outils. Leur développement coopératif s'est fait de la même façon que ceux proposés dans la boîte à outils qui servent de modèle.

IV - BILAN ET PERSPECTIVES

Ce dernier chapitre va être consacré à la critique du système de conférence temps-réel que nous avons développé et à un premier bilan de notre expérience dans le domaine du groupware. A travers cela, nous dégagerons les axes de recherche, qui nous semblent nécessaires au développement des systèmes C.S.C.W..

Nous allons aborder ce chapitre par une analyse de l'architecture en la comparant avec la tendance actuelle d'utilisation de X Windows. Nous allons ensuite revenir sur les différents types d'activité que les participants peuvent avoir lors du déroulement d'une application coopérative ou plus généralement, d'une tâche collective. En effet, le type d'activité est l'élément fondamental qui va contraindre à l'utilisation des différents concepts du groupware que nous avons introduits dans le premier chapitre (relâchement du W.Y.S.I.W.I.S., architecture, gestionnaire de conférence et parallélisme d'action entre utilisateurs).

4.1 - Architecture

Notre système de conférence est conçu de manière à pouvoir y insérer des applications coopératives qui pourront être implémentées en bénéficiant de facilités de développement. Au départ, nous avions choisi d'avoir des modèles applicatifs, centralisés sur le serveur. Les applications clientes ne disposaient que d'un modèle "proxy" chargé de renvoyer, pour traitement, les messages sur le serveur. Les messages étaient exécutés et le résultat renvoyé au "proxy" qui était en attente. Nous nous sommes expliqués sur ce choix au cours du second chapitre mais nous ajouterons cependant qu'il permettait aussi de proposer un schéma simple de développement pour des applications coopératives.

L'un des constats que nous avons fait en implémentant l'application décrite au chapitre III, est qu'il fallait réduire au maximum l'envoi de messages à des objets distants car c'est l'un des facteurs les plus pénalisants pour les temps de réponse. Pour remédier à cela, nous avons réparti du code sur les stations clientes en implémentant des méthodes dans les classes des contrôleurs (il s'agit d'un mécanisme que l'on retrouve déjà implémenté pour certains types de fenêtres).

Implémenter les méthodes dans la classe du modèle ne serait pas plus compliqué et cela permettrait de garder une distinction plus nette sur le rôle de chaque élément de la triade M.V.C.. Les modèles des applications clientes ne seraient plus alors uniquement de simples "proxy" mais des objets qui répondraient à un certain nombre de messages et qui renverraient sur le modèle serveur ceux qu'ils ne savent pas exécuter.

La figure suivante montre l'organisation des classes que l'on pourrait avoir pour bénéficier de ce mécanisme.

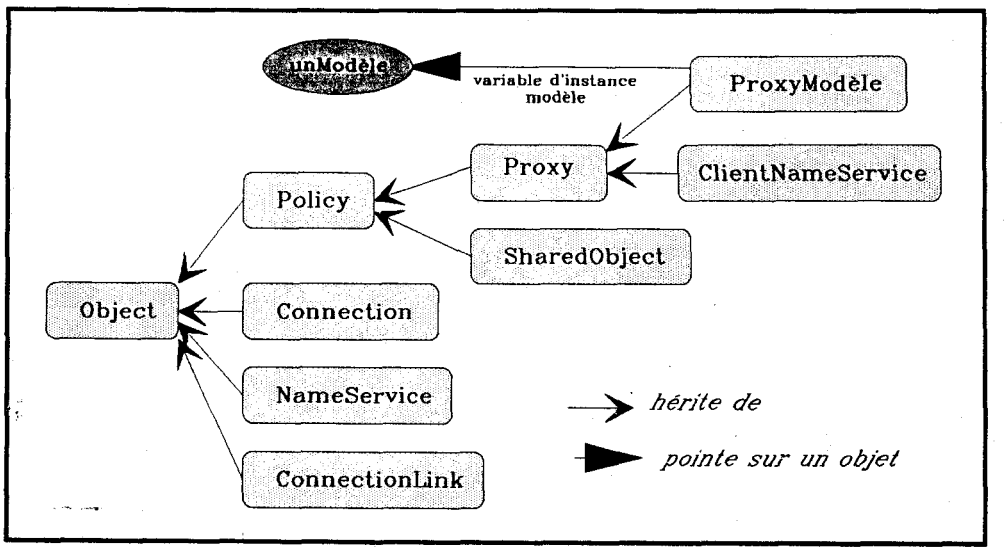


Figure 4.1. *Nouvelle organisation des classes. La variable d'instance "model" pointe sur le modèle qui contient les méthodes implémentées localement. L'intérêt de cela est que l'on peut avoir comme modèle n'importe quelle instance d'une classe. Nous n'avons pas à réécrire les méthodes du modèle dans des classes, sous-classes de Proxy.*

Ainsi, pour notre système de conférence temps-réel, nous avons progressivement migré vers une architecture hybride suivant un modèle client/serveur. Cette solution s'est avérée bonne car les temps de réponse pour les mises à jours des fenêtres sont tout à fait corrects et permettent un travail coopératif agréable. Nous devons cependant signaler qu'à ce stade du projet, nous n'avons pas encore fait d'évaluation sur la charge réseau induite par notre système. Ce travail sera nécessaire pour le passage du système sur le réseau R.N.I.S. car nous devrons vérifier, par exemple, si la longueur des trames échangées joue sur les temps de réponse.

Dans notre environnement, le développement d'une application coopérative de type W.Y.S.I.W.I.S. commence par une définition des couples vue/contrôleur nécessaires à la réalisation de son interface. Nous en avons développé quelques-uns dans une boîte à outils : ils sont à la disposition des programmeurs mais servent, avant tout, d'exemple pour la définition de nouveaux couples qui correspondraient mieux aux besoins de l'applicatif. Il faut, pour chacun d'eux, définir les interactions autorisées lorsque l'utilisateur aura ou n'aura pas le contrôle de l'espace public. Il faut également choisir ce qui devra être mis à jour sur l'ensemble des stations lorsque l'espace public sera modifié.

Lorsque ces éléments de base sont disponibles, le modèle de l'application se développe d'une façon classique mais en deux temps. Il faut tout d'abord écrire les

méthodes en ne tenant pas compte des stations distantes. Les mises à jour et les tests pour savoir, par exemple, quel menu doit être renvoyé, ne sont plus à faire car ils sont gérés par les couples vue/contrôleur. La seconde étape va consister à étudier la façon de réduire les appels/réponses entre les stations clientes et le serveur. La majorité des modifications va être liée aux méthodes invoquées lors de la sélection d'un "item menu". En effet, il est possible d'exécuter certaines opérations en local avant de renvoyer les messages sur le modèle. Cette technique permet également de réaliser des tâches sans que le modèle en soit informé. C'est à ce niveau que les modèles des stations clientes, totalement proxy, deviennent peu à peu, des modèles qui gèrent une partie du code applicatif.

Pour les applications coopératives qui ne nécessitent d'espaces de type W.Y.S.I.W.I.S., le problème doit être traité globalement dès le départ. Le rôle de chaque participant sera lié à l'applicatif. Ce dernier nécessitera un développement spécifique sur chacune des stations. La transmission d'informations entre les stations va concerner les données partagées. Elle se fera par l'intermédiaire des objets "exécuteurs" localisés sur toutes les stations et accessibles à partir des gestionnaires de conférence. Il n'y aura pas forcement de diffusion à l'ensemble des stations car nous pouvons introduire la notion de point de vue sur le contexte partagé.

Nous allons maintenant parler de l'architecture sous X Windows, qui devient un standard incontournable adopté par la majorité des utilisateurs et des constructeurs. D'ailleurs, l'un des travaux en cours de notre équipe de recherche est de porter le système de conférence ainsi que l'application sous la nouvelle version 4.0 de Smalltalk qui s'appuie maintenant sur X Windows.

Présentation de X Windows

X Windows, plus communément appelé X, peut être défini comme un système de multi-fenêtrage pour des environnements répartis et résulte des travaux du MIT (Massachusetts Institut of Technology) en collaboration avec différents industriels [DUBO 90].

X Windows est d'abord destiné à créer des interfaces utilisateurs évoluées en proposant une uniformisation de celles-ci. Il offre aussi la possibilité de faire dialoguer des machines distantes : il permet par exemple d'exécuter des applications sur une machine et de visualiser ou d'interagir sur une autre.

S'agissant d'un système multi-fenêtrage, X Windows contient des couches logicielles gérant d'une part l'interactivité d'un poste de travail et d'autre part l'affichage

graphique s'effectuant sur des écrans bitmap.

X Windows utilise une architecture Client/Serveur. Le client est un programme d'application qui va solliciter, auprès d'un serveur, des services à travers un réseau. Le serveur est un programme qui contrôle et pilote l'interface de visualisation en recevant et en exécutant les ordres d'un client. La figure suivante illustre l'architecture de X Windows.

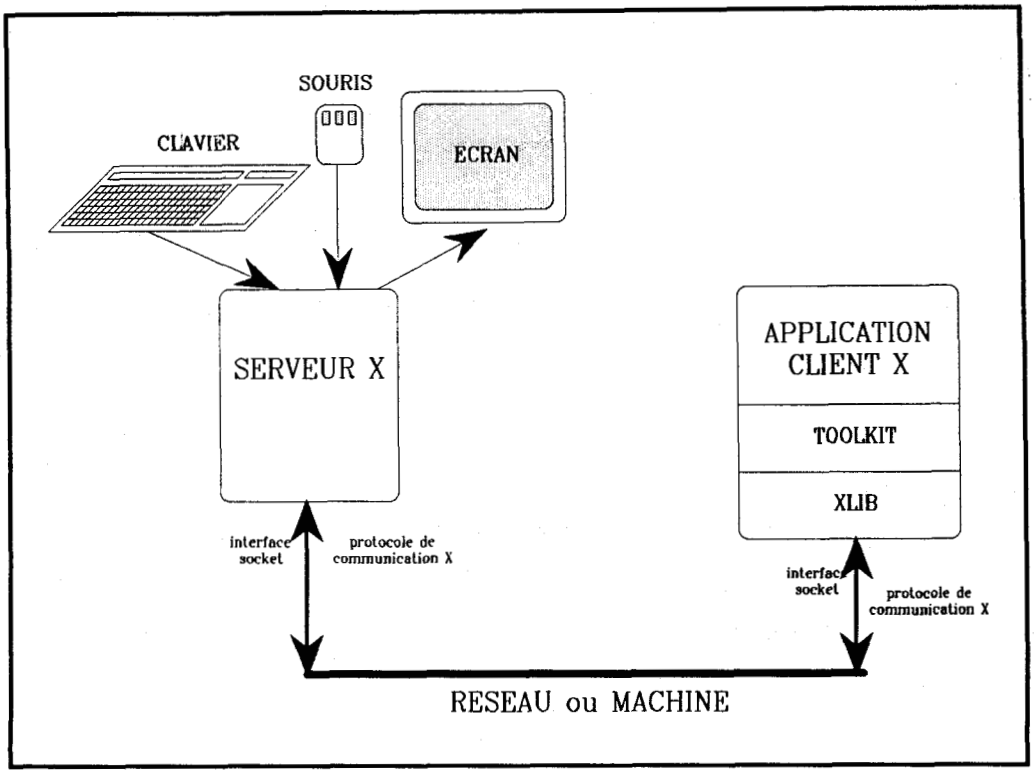


Figure 4.2. *Architecture de X Windows.*

X Lib est une librairie de fonctions d'assez bas niveau qui permettent de développer des applications mais d'une façon qui reste assez compliquée.

Les Toolkits sont des utilitaires qui peuvent être des interfaces de programmation comme OPENLOOK et MOTIF ou des générateurs d'interfaces comme GUIDE (Sun).

Echange des informations entre clients et serveurs

Les communications entre les serveurs et les clients utilisent le X Protocol qui s'appuie sur l'interface "socket" (il s'agit des "sockets" UNIX lorsque le client et le serveur sont sur une même machine et des "sockets" R.P.C. lorsqu'ils sont sur deux machines différentes).

Les terminaux X

Les terminaux X supportent le programme Serveur, qui est chargé en local de gérer les interactions souris/clavier, les sorties sur l'écran Bitmap et les protocoles réseaux pour communiquer avec le programme client.

Son rôle va consister à renvoyer sur le programme client les requêtes qui seront alors placées dans une file d'attente pour être traitées et à attendre les messages de retour.

On devine donc que l'un des inconvénients de ce type d'architecture est qu'il va entraîner une utilisation intensive du réseau avec pour conséquence une dégradation importante des performances. Les tâches de sélection souris, de déplacement, de redimensionnement des fenêtres et de passage incessant d'une fenêtre à une autre donnent lieu à l'émission de nombreuses trames qui peuvent alors en générer d'autres dues à la réémission (lors de la collusion avec par exemple des messages de transfert de fichiers).

L'avantage des terminaux X est leur prix tout à fait abordable. L'architecture, qu'ils permettent de mettre en place, est basée sur des machines dédiées à des traitements spécifiques.

Deux tendances : Coopératisation d'une application existante ou développement d'une application spécifique

Il existe, à l'heure actuelle, deux tendances parmi les concepteurs de Groupwares : il y a ceux qui "coopératisent" des applications existantes mono-utilisateur et ceux qui en développent des spécifiques. Ces deux techniques ont chacune des avantages et des inconvénients.

Comme nous l'avons expliqué dans la première partie, une application coopérative ne se résume pas à une application mono-utilisateur à laquelle on rajoute de multiples entrées/sorties. Une réelle application coopérative doit être conçue dès le départ pour une utilisation en groupe.

Proposer des systèmes ou des boîtes à outils qui permettent de coopératiser des applications est intéressant car il n'est alors pas nécessaire de les réécrire et que les applications potentiellement adaptables sont extrêmement importantes. Bien que les applications obtenues soient parfois mal adaptées aux réels besoins du groupe et des individus, elles ont l'avantage d'exister. En effet, il n'est pas facile de réécrire

l'équivalent des logiciels de bureautique et autres que l'on peut trouver actuellement sur le marché.

L'architecture de X Windows permet aisément de "coopératiser" des applications en y apportant très peu de modifications. Nous allons brièvement présenter les travaux de différentes équipes de recherche qui ont choisi cette approche pour développer des systèmes de C.S.C.W.. Il faut souligner que dans cette approche, deux tendances se dégagent : il y a ceux qui proposent des boîtes à outils permettant de construire les applications partagées et ceux qui offrent des systèmes qui peuvent intégrer des applications standards.

Une équipe de recherche italienne [BONA 91] s'est basée sur X Windows pour proposer un "conférence toolkit" qui permet l'intégration d'applications standards dans un environnement de conférence ou le développement d'applications de groupe spécifiques.

Les applications sont considérées comme étant partageables si elles sont décomposables en deux parties ayant chacun un rôle bien précis.

. L'interface utilisateur

. Le noyau applicatif

Conférence toolkit offre tout un ensemble de composants dont les principaux sont les suivants :

. Un gestionnaire de conférence.

. Un "switcher" qui multiplexe et démultiplexe les streams de données entre les différents utilisateurs pour l'intégration d'applications existantes sur le système de conférence.

. Un "Bridge Manager" qui est chargé de router les données sur les stations des utilisateurs qui en ont les droits. Il permet, par là même, une sélection dans la diffusion.

A partir de ces éléments, le partage d'une application pourra être construit d'une façon tout à fait classique sous X Windows (Cette approche est en quelque sorte comparable à l'approche du terminal virtuel) avec un client (programme de l'application) et des serveurs (interface). Au client sera associé un "switcher" pour gérer les multiples entrées et sorties. Il s'agit dans ce cas d'un mode "centralisé".

Dans l'utilisation d'une architecture sous X Windows nous citerons aussi le système de conférence MERMAID [WASA 90], Rendezvous [PAHI 90] et shared X [GAGU 89]. MERMAID est un système de conférence bureautique et Rendezvous est une architecture conçue pour créer des applications multi-utilisateurs synchrones. Les travaux de DEC/Karsruhe avec Xshared et de HP avec sharedX vont également dans ce sens avec une approche plus conservatrice.

Comme nous l'avons précédemment dit, les tâches de sélection souris, de déplacement et de passage incessant d'une fenêtre à l'autre génèrent un flux important d'informations entre le serveur et le client. Avec une application de groupe, nous aurons autant de serveurs qu'il y aura d'individus connectés et par conséquent un flux amplifié. Si nous nous en tenons à l'objectif que nous nous sommes fixés, c'est à dire de porter le système sur le réseau R.N.I.S., cette approche n'apparait pas raisonnable car le réseau ne sera pas en mesure de supporter cette charge. Comme les temps de réponse et de mise à jours ne pourront pas être courts, cela ne pourra pas satisfaire les participants.

La seconde approche qui consiste à écrire une application spécifique est, pour nous, la solution qui s'impose. Il est alors possible de faire diminuer dans des proportions très importantes, le flux d'informations entres les stations. Dans ce cas, il n'est plus possible d'utiliser des terminaux X mais il faut, au contraire, disposer de stations X qui permettront l'implémentation d'un programme client. Les différents types d'architecture (centralisée, dupliquée et hybride) sont permis et le choix est laissé au programmeur.

L'avantage d'écrire une application spécifique est qu'il est alors possible de répartir du code. Cela permet d'utiliser les ressources locales de traitement et de lancer sur les stations clientes des programmes clients (du point de vue de X) qui échangeront entre eux des informations de "haut niveau".

Cette approche a également été abordée par l'équipe de recherche italienne. Pour cela, ils ont introduit des composants appelés de "présentation" qui sont locaux aux stations et qui ne renvoient sur un client que les événements significatifs. Le bénéfice de cette opération est une diminution très importante du flux d'informations mais le prix à payer est une modification importante des applications qui suppose un accès aux codes sources.

La polémique qui peut avoir lieu sur le choix de "coopératiser" une application existante ou d'en développer une spécifiquement n'a pas grand intérêt car les deux solutions sont complémentaires. La première solution permet de disposer d'un ensemble très important d'applications et même si le résultat n'est pas parfait pour

l'utilisation de groupe, elles ont le mérite d'exister. Dans l'état actuel, expérimenter ces applications "coopératisées" permet d'identifier des problèmes qui pourront être résolus dans les applications développées spécifiquement. La seconde est effectivement beaucoup plus performante mais il faut tout réécrire ou disposer des codes sources pour les modifier.

4.2 - Différents types d'activités

Lorsqu'un objectif commun est donné à un groupe constitué de plusieurs personnes, son aboutissement nécessite la réalisation de ce que nous appelerons une "tâche commune". Cette tâche peut éventuellement être constituée de plusieurs tâches de moindre importance.

Deux cas de figures peuvent se présenter pour l'exécution d'une de ces tâches :

. Il est possible de fractionner la tâche commune en un nombre N de sous-tâches, qui seront attribuées à différents individus ou à différents sous-groupes. Leur réalisation sera alors indépendante et la prise de décision incombera uniquement à la personne ou au sous-groupe concerné qui pourra cependant tenir compte d'un environnement partagé par le groupe. Des phases de synchronisation entre les différentes personnes sont alors nécessaires.

. Il est impossible de subdiviser la tâche commune car la prise de décision dépend entièrement du groupe. Il faut alors, pour parvenir au résultat, une coordination (collaboration intense) entre toutes les personnes concernées. Un exemple rencontré au cours de notre étude est la réalisation du plan de production. En effet, il ne serait pas raisonnable de demander aux cinq équipes de remplir respectivement une partie de tableau sans tenir compte de l'avis des autres joueurs.

Comme dans le cas de notre application de simulation, les tâches collectives sont généralement constituées d'un ensemble de sous-tâches appartenant aux deux cas de figure précédemment cités.

C'est ce que nous avons par ailleurs rencontré lors de la mise en oeuvre de la méthode kanban dans notre application de simulation. Effectivement, les décisions sont prises individuellement par chaque équipe mais elles tiennent compte de l'état des stocks qui évoluent en fonction des productions et des prélèvements de chacune d'elles.

Le type d'application qui nous intéresse tout particulièrement est celui qui nécessite une coopération et une négociation du groupe ou du sous-groupe. Pour que les discussions et les prises de décisions soient possibles, il faut alors qu'un support visuel commun soit offert. Il servira à argumenter, à attirer l'attention sur un point particulier ou encore à matérialiser la progression du groupe.

Quelle que soit l'application ou la tâche, les objets ou les données ne pourront être modifiés à un instant donné que par une seule personne à la fois. Le problème ne se pose pas d'un point de vue informatique puisque toutes les techniques présentées pour gérer l'accès concurrent sont utilisables mais d'un point de vue social.

4.3 - Le gestionnaire de conférence

Le système de conférence temps-réel que nous avons développé a été spécialement étudié pour des applications coopératives où les individus ont une même tâche à réaliser. Il permet également l'intégration d'applications coopératives de la seconde catégorie. Cependant dans ce cas, le protocole de gestion de conférence n'est plus adapté à l'application. En effet il n'est plus alors possible de "factoriser" ou "d'identifier" un scénario puisque celui-ci variera avec les diverses applications. Le rôle des participants est alors ramené au niveau de son applicatif et il est lié aux fonctionnalités qui lui seront offertes. Notre architecture d'accueil permet néanmoins de récupérer les mécanismes de diffusion des messages ainsi que les objets "exécuteurs" utiles à la réalisation d'actions commandées à distance.

Jusqu'à maintenant, nous avons étudié et implémenté sur notre système qu'un seul algorithme de contrôle correspondant à la situation suivante :

. Le maître de conférence a des droits supérieurs aux autres et peut par exemple reprendre le contrôle à l'un des participants à tout instant ou encore l'affecter à n'importe qui, même si celui-ci n'en a pas fait la demande.

. L'attribution du contrôle de l'espace public aux participants est fonction de l'ordre dans lequel les demandes ont été reçues.

Avant de poursuivre, nous allons présenter rapidement les solutions qu'ont adoptées d'autres équipes de recherche pour résoudre le problème de l'algorithme de contrôle.

i) - Une équipe de chercheurs japonais, qui travaille sur un système de conférence distribué appelé MERMAID [KASH 90] a implémenté trois algorithmes différents :

. Le premier correspondant au notre, accorde le contrôle aux participants en fonction de l'ordre des requêtes mais ici chaque participant a les mêmes droits y compris le maître de conférence.

. Avec le second, le contrôle est attribué aux participants par le maître de conférence.

. Le troisième est particulier car c'est le participant qui a le contrôle, qui choisit celui qui va l'avoir.

Le choix de l'algorithme de contrôle est alors proposé aux

participants en début de séance par l'animateur.

ii) - DistEdit [KNPR 90] est un "toolkit" qui fournit un ensemble de primitives pouvant être utilisées pour construire des applications supportant la collaboration de plusieurs personnes dans des environnements distribués. Pour expérimenter leur système Knister et Piakash ont modifié deux éditeurs : MicroEmacs et GNUEmacs. Cette modification permet aux utilisateurs de faire des changements d'édition dans leur environnement sous le regard des autres. Cependant, là encore, les modifications ne peuvent être faites que par une personne à la fois.

Les participants peuvent avoir deux statuts distincts qui évoluent au cours du temps ; "observateur" ou "maître" (le maître étant le seul à pouvoir modifier le texte pendant que les observateurs regardent. Le changement de statut se fait par l'intermédiaire de "switchs" disposés sur les fenêtres des participants ; Le maître qui clique sur le switch redevient observateur et l'observateur qui clique sur le switch devient maître si le poste est disponible. On peut également forcer un changement de statut au cas où, par exemple, le maître oublierait de rendre le contrôle avant de quitter son bureau.

iii) - Les travaux de S Greenberg [GREE 91] sont également très intéressants dans ce domaine. Il propose plusieurs algorithmes de contrôle de l'espace public qui sont en partie ceux que nous avons présentés précédemment. Son prototype SHARE est un Groupware personnalisable en ce sens qu'il offre la possibilité aux utilisateurs de choisir le type d'algorithme qui pourra les aider au mieux dans leurs interactions avec l'espace partagé. Il a introduit dans son système, un algorithme de contrôle qui permet aux participants de reprendre le contrôle de l'espace public à tout instant.

Le tableau ci-dessous représente les différents types de protocole que l'on rencontre avec les systèmes sur lesquels ils sont implémentés.

Description du protocole	Système
Tous les participants peuvent interagir à n'importe quel moment, ils ont ainsi tous accès à l'espace public	Timbuktu Share Cognoter
Chaque participant peut reprendre le contrôle de l'espace public à tout instant	CaptureLab Dialogo Share
Le participant qui contrôle l'espace public doit le rendre explicitement pour que les autres participants puissent le prendre	Cantata Share DistEdit Notre système
Le contrôle est attribué en fonction de l'ordre des demandes (file d'attente) La restitution de l'espace public doit se faire explicitement par la personne qui en a le contrôle	Cantata Vconf Mermaid Notre système
Un participant supervise toutes les activités. Il décide qui aura le contrôle et à quel moment en fonction des demandes qui lui sont faites	RtCal Share Mermaid Notre système
Le contrôle de l'espace public est à la disposition des autres participants dès l'instant où la personne qui le contrôle a une interruption significative dans ses activités	Emce Share

A travers nos travaux et ceux d'autres équipes nous nous apercevons qu'il n'existe pas de nombreuses façons de gérer l'espace public. Ce que l'on pourra tout de même critiquer à ces protocoles de gestion est qu'ils ont tous une certaine rigidité dans le passage du contrôle, phénomène que l'on ne retrouve pas dans les réunions classiques.

En effet au cours d'une discussion, les personnes savent implicitement à quel moment elles doivent prendre la parole car il existe tout un ensemble d'indices comportementaux qui leur indique le moment opportun. Il apparait donc très intéressant et très important d'expérimenter des protocoles moins contraignants dans le passage du contrôle de l'espace public entre les participants. On pourrait par exemple imaginer de le rendre automatiquement après un certain temps d'inactivité.

D'autre part dans une conversation classique, il est possible de deviner à l'avance le moment où la personne va s'arrêter de parler (au ton, au contenu même de son discours...).

Ceci nous amène à penser qu'il serait intéressant d'étudier ce problème et voire dans quelle mesure on pourrait assouplir ce passage en utilisant, par exemple, un artifice visuel qui signalerait au groupe que le contrôle va être bientôt relâché. Ce serait d'autant plus bénéfique que cela permettrait au suivant de la liste de se préparer et de ne pas être surpris par une attribution soudaine du contrôle.

Un autre point, qui n'a pas été abordé, est l'introduction d'un niveau de priorité différent entre les participants. Dans ce cas, l'ordre des requêtes ne serait plus le seul élément déterminant dans l'attribution du contrôle.

On se rend donc parfaitement compte que le problème du passage du contrôle entre les utilisateurs reste primordial et ce, d'autant plus, qu'il fait surtout intervenir des problèmes d'ordre social et non d'ordre technique. Nous pensons que chaque application nécessite un protocole spécifique qui devra, entre autres, tenir compte des utilisateurs et en particulier des relations qu'ils entretiennent (hiérarchie). Il ne faudrait surtout pas qu'une trop grande rigidité du protocole inhibe les interventions des participants, en particulier dans les réunions de type "brainstorming".

Dans ces systèmes, il faut que chacun connaisse l'état de la conférence à tout instant. Savoir quel participant contrôle, quelle sera la prochaine personne à contrôler l'espace public, font partie des informations à afficher impérativement à tout moment. Diviser l'espace public en plusieurs sous-ensembles contrôlables par des personnes différentes ne fait qu'amplifier la difficulté à satisfaire une rétro-action des activités de chacun.

4.4 - Le parallélisme d'action entre les utilisateurs

Dans certains cas, le groupware permet d'effectuer des tâches d'une façon parallèle augmentant ainsi la productivité du groupe. Nous avons vu qu'il y avait deux types de tâches coopératives : celles qu'on ne peut pas diviser car elles nécessitent un accord et une coordination du groupe ou d'un sous-groupe, et celles que l'on peut diviser et donc confier à différentes personnes.

Les tâches coopératives qui nécessitent l'accord d'au moins deux personnes permettent, néanmoins, certaines activités parallèles. En effet, ceux qui ne disposent pas du contrôle ont toujours la possibilité de travailler dans leur espace privé. Ils peuvent préparer des informations qu'ils ramèneront en temps voulu dans l'espace public pour les soumettre à l'avis général. Ils peuvent aussi utiliser les fonctionnalités offertes dans l'espace public lorsqu'ils n'ont pas le contrôle. Il s'agit essentiellement de fonctions qui ne modifient pas l'espace public comme par exemple copier un texte. Ils récupèrent alors des informations de l'espace public pour les travailler dans leur environnement privé.

Jusqu'à maintenant, l'échange d'informations entre l'espace public et l'espace privé se limite uniquement, aux opérations simples de copier/coller. Les travaux d'Olson [OLMA 90] dans ce domaine portent sur l'étude de nouveaux types de transfert d'informations entre les espaces privés et publics. Il envisage la possibilité d'avoir des transferts progressifs entre les espaces privés et publics. S'il reprend la notion de point de vue, on peut également admettre que le passage d'informations entre les deux espaces soit régit par différentes règles de composition.

S'il est possible de diviser une tâche coopérative en sous-tâches, qui pourront être alors confiées à des individus où à des sous-groupes différents, le parallélisme sera effectif.

Cette répartition des tâches va alors obligatoirement entraîner un fractionnement de l'espace public. Ce fractionnement pourra se faire de trois façons différentes.

. Fractionnement de Surface : L'espace public associé à une ou plusieurs fenêtres peut être divisé en plusieurs sous-fenêtres qui seront sous le contrôle d'utilisateurs différents.

. Fractionnement Structurel : L'espace public est constitué de sous-espaces dépendants les uns des autres. Ils sont associés à des fenêtres différentes.

. Fractionnement Sémantique : L'espace public est constitué de sous-

espaces indépendants les uns des autres (cela permet d'introduire des notions de points de vue différents).

Quelque soit le fractionnement appliqué à l'espace public, les différentes parties de cet espace seront sous le contrôle d'une seule personne ou d'un seul sous-groupe à un instant donné. La gestion de la conférence va alors devenir beaucoup plus complexe puisqu'il faudra représenter et faire évoluer les personnes qui contrôlent ces sous-ensembles. Un exemple de représentation est proposé à la figure suivante :

	utilisateur x	utilisateur y	utilisateur z	utilisateur u	utilisateur v
sous espace 1			C	P	
sous espace 2		C			
sous espace 3	P			C	
sous espace 4				P	C

C utilisateur qui a le contrôle de l'espace public
P prochain utilisateur qui aura le contrôle

figure 4.3. *Exemple de répartition des sous-espaces publics entres les utilisateurs. Le grisé correspond aux utilisateurs qui peuvent intervenir sur le sous-espace concerné.*

Répartir l'espace public en sous-espaces permet d'augmenter la production des participants mais il faut avoir à l'esprit que plus le nombre de sous-espaces sera élevé, plus la gestion et la représentation seront difficiles.

Cette répartition des tâches au sein du groupe pourra également avoir pour conséquence, un relâchement de la dimension "groupe" du WYSIWIS. En effet, les mises à jours des vues d'un sous-espace public pourront être faites uniquement pour le sous-groupe concerné.

Actuellement, la plus grande partie des recherches porte essentiellement sur une catégorie spécifique du groupware : celle des éditeurs de groupe. Nous citerons pour exemple les prototypes GROVE [ELGI 90], CoAuthor [HAJA 91] et le Collaborative Annotor [KOLI 91]. Ces prototypes permettent effectivement d'avoir des activités parallèles mais uniquement dans certaines phases qui sont essentiellement les phases de

production (génération d'idées ou de textes...). Il faut également souligner que ces systèmes sont conçus pour une utilisation asynchrone.

Cependant, lorsque la participation d'un groupe ou d'un sous-groupe est nécessaire à la prise de décision, on revient alors aux stratégies précédemment citées, où un seul individu a le contrôle de l'espace public à un moment donné. Il est à noter que pour certains prototypes, la possibilité de travailler de façon asynchrone est également permise. Une gestion en fonction du droit d'accès aux documents est alors réalisée. Ce type de travail permet de préparer des documents avant une réunion et de les terminer lorsque la conférence en temps réel est finie.

En conclusion, nous pouvons dire que les activités parallèles permettent un gain de temps effectif à la seule condition qu'elles soient bien structurées. Là encore, le problème n'est pas d'ordre technique mais plutôt d'ordre cognitif et social. De plus, il faut alors une gestion et une représentation très complexes des activités et des droits de chacun des participants. La répartition des tâches nécessite aussi un choix judicieux de la granularité des sous-espaces publics. A savoir, choisira t-on pour un texte le chapitre, le paragraphe ou encore la phrase ?

4.5 - Les relâchements du W.Y.S.I.W.I.S.

Les relâchements du W.Y.S.I.W.I.S. [STBO 87] peuvent opérer dans quatre dimensions différentes : espace, groupe, temps, congruence.

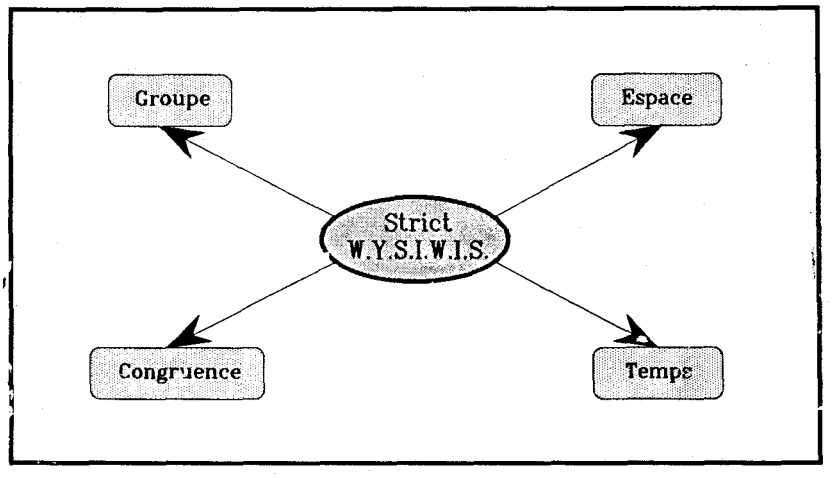


Figure 4.4. *Les quatre dimensions du W.Y.S.I.W.I.S.*

Les raisons pour lesquelles on relâche le strict W.Y.S.I.W.I.S. sont de deux types :

. Par choix délibéré pour augmenter le confort des utilisateurs, on joue alors sur les trois dimensions suivantes : espace, groupe, congruence.

. A cause de contraintes indépendantes de la volonté des concepteurs, c'est alors la dimension temps qui intervient.

La dimension espace

Relâcher la dimension espace concerne essentiellement les positions et le dimensionnement des fenêtres. Sur notre prototype les utilisateurs ont la possibilité de positionner les fenêtres de l'espace public là où ils le désirent mais ils ne peuvent pas les redimensionner et ce en raison de l'utilisation des télépointeurs.

En effet, l'utilisation d'un télépointeur nécessite une transmission des coordonnées de sa position d'origine aux différentes stations. Sa position est généralement transmise en coordonnées relatives (par rapport à la fenêtre active) pour être indépendant de la position de la fenêtre active dans l'écran.

Lorsque l'on modifie la taille des fenêtres, les coordonnées reçues ne correspondent plus et doivent alors subir un calcul pour être adaptées Pour des fenêtres de type bitmap le problème est simple à résoudre puisqu'il suffit d'effectuer un simple rapport entre la taille de la fenêtre de référence et celle où doit avoir lieu l'affichage.

Mais, pour des fenêtres textes où le reformatage du texte est automatique en cas de modification des dimensions de la fenêtre, le problème s'avère plus complexe puisqu'il faut alors déterminer la position des caractères les plus proches du télépointeur. La résolution demeure simple lors de l'utilisation de télépointeur ponctuel mais les algorithmes se compliquent rapidement avec les télépointeurs dynamiques.

Une autre façon de relâcher la dimension espace est de mettre les fenêtres sous forme d'icônes. Les fenêtres restent actives et les mises à jours sont réalisées sans que l'utilisateur le voit.

Que les activités soient individuelles ou collectives, la relâche de cette dimension ne pose pas de problème de cohérence globale dans la conférence car elle ne modifie pas profondément l'espace public. Il s'agit essentiellement d'une modification par rapport à l'environnement privé.

La dimension groupe

La relâche de cette dimension est directement associée au partitionnement de l'espace public en sous-espaces réservés à une ou plusieurs personnes. Dans notre prototype, nous n'avons pas de relâche de cette dimension dans la mesure où nous n'en avions pas l'utilité car l'application de simulation nécessite pendant les phases de coordination une participation du groupe en entier.

Pour relâcher cette dimension, il faudrait modifier la fenêtre de gestion de conférence pour y introduire l'ensemble des sous-espaces publics disponibles avec les utilisateurs concernés et impliqués (le nom des utilisateurs qui peuvent y accéder, le nom de celui qui contrôle cet espace et éventuellement le nom du prochain qui va en avoir le contrôle). Nous arriverions à une représentation sous forme matricielle comme l'exemple sur la figure 4.3. Il est alors nécessaire, si le groupe est plus important, d'étudier de nouvelles interfaces avec des métaphores adéquates. La métaphore souvent utilisée est celle de la "room". Les différentes parties de l'espace de travail (public et privé) sont associées à des pièces. Par exemple, on accède à un espace (fenêtre) par une porte (icône).

La dimension congruence

Cette dimension est celle qui mérite peut-être le plus de réflexion de la part des équipes de recherche sur le groupware. Relâcher la dimension congruence permet d'avoir une représentation différente d'un même objet mais aussi de bénéficier de mécanismes de filtrage des informations en fonction du statut des utilisateurs.

Relâcher cette dimension dans le cas où les participants ont une même tâche à réaliser n'est pas souhaitable car les références verbales souvent utilisées pour les prises de décision deviendraient compliquées. Par contre, pour les applications coopératives où les participants ont une tâche individuelle à accomplir, cette possibilité devient alors très intéressante.

Tout en gardant un contexte global partagé entre utilisateurs, cela leur permet en plus d'en avoir un, personnalisé à leur besoin. Ils ont ainsi les informations représentées sous la forme qu'ils désirent. Par exemple, les valeurs d'un tableau peuvent être affichées sous forme d'histogrammes ou sous formes de courbes. C'est aussi le moyen, comme nous l'avons dit, de filtrer certaines informations et de ne laisser ainsi aux personnes que les informations dont elles ont besoin pour prendre leur décision. C'est également un moyen de garder une confidentialité sur certaines données (citons par exemple certaines dates de rendez-vous d'un chef de service dans le cas d'une utilisation d'un agenda collectif).

La dimension temps

Tout le monde sera d'accord pour admettre que cette dimension doit être relâchée au minimum pour conserver au groupware une aisance d'utilisation.

La relâche de cette dimension n'est généralement pas souhaitée mais subie par les concepteurs de groupware qui chercheront dans leur architecture et dans leur matériel (puissance des stations, codec) le moyen de la minimiser. En effet, le temps de mise à jour est fortement lié au type d'architecture et aux bandes passantes des réseaux utilisés. D'autre part, pour certains média, un léger délai de transmission ne nuit pas à la compréhension mais pour d'autres et en particulier pour la voix, les décalages ne sont pas acceptables.

Dans notre système nous avons choisi une architecture hybride (centralisée avec une partie dupliquée) parce qu'elle nous donnait les meilleurs résultats pour mettre le plus rapidement possible à jours, les informations. Nous avons travaillé sur le type de messages à échanger et la répartition de certaines données pour faire diminuer le flux d'informations à transmettre. Nous avons également échantillonné, pendant les phases d'utilisation du télépointeur dynamique, les coordonnées que l' on transmettait aux autres stations. Les mouvements du télécurseur apparaissent légèrement saccadés lorsque le déplacement de la souris se fait très rapidement mais cela ne gène en rien la désignation et la discussion.

Notre conclusion sur les différentes relâches des dimensions du strict W.Y.S.I.W.I.S. est la suivante :

La dimension temps est celle qu'il faut garder la plus proche du strict W.Y.S.I.W.I.S. c'est à dire avoir les délais les plus courts possibles de façon à avoir un dynamisme de groupe important. Les trois autres relâches dépendent du type d'application coopérative et sont schématisées par le tableau suivant :

Relâchement de la dimension du WYSIWIS	**Tâches collectives non partageables (Prise de décisions)**	**Tâches collectives partageables**
ESPACE	- position - dimension (si problème de télépointage résolu)	- position - dimension - mise sous forme d'icône
GROUPE	Non	Oui
CONGRUENCE	Oui mais très délicate et dépend fortement de l'application	- point de vue différent - filtrage
TEMPS	La plus faible possible pour des raisons ergonomiques	La plus faible possible. Cependant pour certaines tâches des petits délais peuvent être acceptés(*).

(*) Il faut signaler que certains considèrent qu'en relâchant beaucoup cette dimension, on arrive aux systèmes asynchrones.

Figure 4.3. *Les quatre dimensions du W.Y.S.I.W.I.S. avec leurs relâches possibles en fonction du type d'applications.*

Il est à noter que la dimension espace est l'une des plus faciles à satisfaire lorsqu'on évolue dans des environnements multi-fenêtres.

4.6 - Conclusion

Comme nous venons de le voir, de nombreux problèmes restent "ouverts" dans le domaine du Groupware. Leurs résolutions passent par une expérimentation qui permet de les identifier plus sûrement.

Nous pensons qu'actuellement, le plus gros handicap du Groupware est l'absence d'U.I.M.S. coopératifs. Pour évoluer, le Groupware devra bénéficier d'interfaces spécialement adaptées à ses spécificités. Il faudra que ces U.I.M.S. soient d'emblée conçus pour une utilisation en groupe et ne soient pas de simples extensions d'U.I.M.S. mono-utilisateur associés à des multiplexeurs et démultiplexeurs d'entrées/sorties.

L'une des premières évolutions de notre système est son "portage" sous X Windows. Cependant, par rapport aux objectifs que nous nous sommes fixés (utilisation du réseau R.N.I.S.), il nous parait cependant pas envisageable de concevoir les applications de groupe à partir de la "coopératisation" d'applications mono-utilisateur existantes. Nous retiendrons éventuellement cette solution pour faire des expériences sous un réseau local. Nous pensons qu'une réelle application coopérative doit être conçue de prime abord pour un groupe. L'intégration du son et de la vidéo fait aussi partie des améliorations que nous apporterons lorsque nous disposerons des matériels appropriés.

CONCLUSION GENERALE

Notre contribution à l'étude du Travail Coopératif Supporté par Ordinateur et à ses applications, notamment à une nouvelle forme d'enseignement, est relatif à trois points :

- Conception d'une plate-forme d'accueil pour des applications coopératives Temps-réel par la définition d'une architecture hybride proche d'un mode Client/Serveur.

- Mise en évidence de mécanismes de répartition et contrôle de la conférence et implémentation des concepts W.Y.S.I.W.I.S., Télépointeurs, Feed-back multi-utilisateur.

- Conception d'une application spécifique dans le domaine de la gestion de production. L'objectif de cette application étant d'une part, de valoriser l'usage de notre plate-forme dans un enseignement coopératif et d'autre part, de démontrer le caractère adaptatif et ouvert des solutions proposées.

Nous avons le souci de mener systématiquement une double démarche : l'une relative à la faisabilité et l'implémentation dans un environnement moderne de prototypage, l'autre étant relative à une conceptualisation et une comparaison avec d'autres démarches analogues.

Il faut préciser qu'au départ, notre équipe n'avait aucune expérience préalable dans ce domaine. La voie de recherche que nous avons initialisée est aujourd'hui intégrée dans un projet plus vaste de communication pour l'enseignement à distance fédérant les systèmes de communications synchrones et asynchrones [DEVI 90]. Notre travail sera donc prolongé non seulement au sein du laboratoire TRIGONE, mais également au sein du projet CO-LEARN retenu à l'appel d'offre DELTA de la Communauté Economique Européenne.

A court terme, la poursuite des travaux relatifs à la Téléconférence Temps Réel vont se poursuivre dans les directions suivantes :

- intégration des dimensions multimédia,

- conception d'applications génériques supportant les activités coordonnées et parallèles,

- partage de l'application sur d'autres plate-formes standards (X11, Windows) supportant le réseau R.N.I.S..

Ce travail a déjà été amorcé dans le D.E.A. de Pascal Croisy. Nous pensons que nos études pourront également avoir un impact sur la conception des futurs systèmes de gestion d'interfaces Homme-Machine afin de prendre systématiquement en compte le potentiel multi-utilisateur. Par ailleurs, le laboratoire d'Automatique doit poursuivre les travaux relatifs aux applications dans le domaine de la Productique en menant, d'une part, une évaluation systématique du système proposé dans une situation d'enseignement réel, et d'autre part, en élargissant les domaines d'applications, en particulier dans la simulation distribuée.

Le domaine que nous avons eu à aborder est très enrichissant car, au delà de la technique et de la science informatique, il fait appel à des apports de sciences fondamentales telles que la psychologie cognitive, les sciences sociales, voire même, de la philosophie. Ces apports ne peuvent être obtenus que dans un travail collectif et multi-disciplinaire.

REFERENCES BIBLIOGRAPHIQUES

[APKO 86] Applegate L.M., Konsynski B.R., Nunamaker J.F., A group decision support system for idea generation and issue analysis in organization planning, Proceedings of the Conference on Computer-Supported Cooperative Work, Austin, Texas, 3-5 Décembre 1986, pp. 16-34.

[BEAU 91] M. Beaudoin Lafon, Groupware ? Etat de l'art, Groupware et interface, Conférence Ganymède réalisée au Laboratoire d'Informatique Fondamentale de Lille en Mars 1991.

[BENN 87] Bennet John K., The design and Implementation of Distributed Smalltalk, OOPSLA 87 Proceedings, 1987, pp. 318-330.

[BOMA 91] Bonfiglio A., Malatesta G., Tisato F., Conference Toolkit : A Framework for Real-Time Conferencing, Computer Supported Cooperative Work, North-Holland, 1991, pp. 63-77.

[BRAD 86] Brad J. Cox, Ph.D., Object-Oriented Programming - An Evolutionary Approach, Addison-Wesley Publishing Company, 1986.

[BUSH 45] Bush Vannevar, As We May Think, Issue of the Atlantic Monthly, N° 176, (Report in Greif 1988), July 1945, pp. 101-108.

[CHCH 91] Chin Roger S., Chanson Samuel T., Distributed Object-Based Programming Systems, ACM Computing Surveys, Vol. 23, N° 1, March 1991, pp. 91-124.

[COUT 90] Coutaz J., Interface homme ordinateur : conception et réalisation, Dunod Informatique, 1990.

[CRMI 90] Crowley Terrence, Milazzo Paul, Baker Ellie, Forsdick Harry, Tomlinson Raymond, MMConf : An Infrastructure for Building Shared Multimedia Applications, Proceedings CSCW 90, ACM Conf., October 1990, pp. 329-342.

[CULL 87] Mc Cullough Paul, Transparent forwarding : first steps, OOPSLA 87 Proceedings, 1987, pp. 331-341.

[DECL 91] Derycke A., Clément D., Ladesou C. Viéville C., Computer Mediated Conferences for Collaborative Learning : towards hypermedium, National Education and Computer Conference, Phoenix, Juin 1991, pp. 226-234.

[DECO 86] Decouchant D., Design of a distributed Object Manager for the Smalltalk-80 System, OOPSLA 86 Proceedings, 1986, pp. 444-452.

[DEUT 89] Deutsch L. Peter, The Past, Present, and Future of Smalltalk, ECOOP'89, 1989, pp. 72-87.

[DEVI 90] Derycke A., Viéville C., Vilers P., Cooperation and Communication in Open Learning : The CoCoNut Project, Fifth World Conference on Computer in Education, IFIP, Sydney, Australie, 9-13 juillet 1990, pp. 957-962.

[DUBO 90] Dumeur R., Le Borgne J.A., Bassini M., PratiX - Utiliser et programmer X, Editions C2V, 1990.

[DUGE 90] Dugerdil Ph., Programmation par objets, Collection informatique, Eds. Dunod, 1990.

[ELGI 90] Ellis Clarence A., Gibbs Simon J. and Rein Gail L., Design and Use of a Group Editor, North-Holland, 1990, pp. 13-28.

[ELGI 91] Ellis C.A., Gibbs S.J., Reln G.L., Groupware, some issues and experiences, Communications of the ACM, Vol. 34, N° 1, January 1991, pp. 39-56.

[ENEN 68] Engelbart D.C., English W.K., A Research Center for Augmenting Human Intellect, Proc. Fall Joint Computing Conf., Thompson Book Co., Washington DC, (Report in Greif 1988), Décembre 1968, pp. 395-410.

[ENGL 84] Englebart D.C., Collaboration support provisions in AUGMENT, OAC 84 digest, Proceedings of the 1984 AFIPS Office Automation Conference, Los Angeles, Californie, AFIPS, Reston, Va., (Report in Greif 1988), 20-22 Fevrier 1984, pp. 51-58.

[GAGU 89] Greenberg S., Sharing views and interactions with single-user applications, In COIS'90 : Proceedings of the Conference on Office Information Systems, Boston, April, 1990.

[GARI 80] Garibaldi A.M., Affective contributions of cooperative and group goal structures, Journal of Educational Psychology, Vol. 71 (6) - 1979, pp. 788-794.

[GIBB 89] Gibbs S.J., LIZA : An Extensible Groupware Toolkit, MCC, Software Technolgoy Program, Austin, Texas, CHI, May 1989, pp. 29-35.

[GORO 83] Goldberg A., Robson D., Smalltalk-80 : the langage and its implementation, Addison Wesley, 1983.

[GREE 91] Greenberg S., Personalizable groupware : Accommodating individual roles and group differences, Proceedings of the Second European Conference on Computer-Supported Cooperative Work, Amsterdam, The Netherlands, September 25-27, North-Holland, 1991, pp. 17-32.

[GRSA 86] Greif Irene, Sarin Sunil, Data Sharing in Group Work, Computer-Supported Cooperative Work : A book of Reading, Austin, Texas, December 1986, pp. 477-508.

[GREI 88] Greif I., Computer supported cooperative work : a book of reading, 783 p., Morgan Kaufman Publishers, 1988.

[GUNI 88] Guan Sheng, Nievergelt Jay, Abdel-Wahab Hussein M., Shared workspaces for Group Collaborations : an Experiment Using Internet and Unix Interprocess Communications, IEEE Communications Magazine, November 1988, pp. 10-16.

[HAJA 91] Hahn Udo, Jarke Matthias, Eherer Stefan, Kreplin Klaus, CoAUTHOR - A Hypermedia Group Authoring Environment, CSCW, North-Holland, 1991, pp. 79-100.

[HILT 86] Hiltz S.R., The virtual classroom : using computer mediated communications for University, Journal of Communication, 36 (2), 1986, pp. 95-104.

[HITU 78] Hiltz S.R., Turoff M., The network nation : Human communication via computer, Wiley, 1978.

[HITU 85] Hiltz S.R., Turoff M., Structuring computer mediated communication systems to avoid information overload, Communication of A.C.M., Vol. 28, N° 7, July 1985, pp. 680-689.

[KARS 87] Karsenty Solange, Graffiti : un outil interactif et graphique pour la construction d'interfaces homme-machine adaptables, Thèse de 3ème cycle, Université d'Orsay, 17 Décembre 1987.

[KASI 82] Kasik David J., A User Interface Management System, In Computer Graphics, July 1982, pp. 99-106.

[KNPR 90] Knister Michael J., Prakash Atul, DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors, Software Systems Research Laboratory, Department of EECS, U. of Michigan, Ann Arbor, MI 48109, October 1990, pp. 343-355.

[KOLI 90] Koszarek J.L., Lindstrom T.L., Ensor J.R., Ahuja S.R., A Multi-User Document Review Tool, Multi-User Interfaces and Applications, IFIP, (in Verrijn/Stuart 1990), 1990, pp. 207-214.

[KORA 90] De Koven C., Radhakrishnan T., An Experiment in Distributed Group Problem Solving, Multi-user Interfaces and Applications, Ifip, (in Verrijn/Stuart 1990), 1990, pp. 61-76.

[KRKI 88] Kraemer K., King J.L., Computer-based Systems for Cooperative work and group decision making, ACM Computing Survey, Vol. 20, N° 2, June 1988, pp. 211-226.

[LANT 86] Lantz Keith, An Experiment in Integrated Multimedia Conferencing, Proceedings of the Conference on Computer-Supported Cooperative Work, (In Greif 1988), December 1986, pp. 267-275.

[MADS 89] Madsen Christian Murmann, Using Persistent Objects to Implement an Environment for Cooperative Work, Tools'89, 29 Septembre 1989, pp. 243-252.

[MEYE 90] Meyer Bertrand, Concept et programmation par objets - Pour du logiciel de qualité, Traduit de l'américain et mis à jour par Robert Mahl, InterEditions, 1990.

[OLMA 90] Olson Judith S., Mack Lisbeth A., Wellner Pierre, Olson Gary M., Concurrent Editing : The Groups Interface, Human-Computer Interaction - Interact'90, IFIP, North-Holland, 1990, pp. 835-840.

[PAHI 90] Patterson J., Hill R., Rohall S., Rendez-vous : An architecture for synchronous multi-user applications, Proceeding of the CSCW 90, ACM Conference, October 1990, pp. 317-328.

[PFPE 79] Pferd W., Peralta A. and Prendergast F.X., Interactive graphics teleconferencing, Computer 12, November 1979, pp. 62-72.

[POAC 85] Poggio A., Aceves J.J. Garcia Luna, Craighill E.J., Moran D., Aguilar L., Worthington D., Hight J., CCWS : A Computer-Based, Multimedia Information System, IEEE Computer, December 1985, pp. 92-103.

[RINA 89] Richy Hélène, Nanard Jocelyne, Spécification et Contrôle Orienté Objet de l'Accès Concurrent à des Documents Structurés, Woodman'89, 1989, pp. 152-164.

[SAGR 85] Sarin S., Greif I., Computer based real-time conferencing systems, IEEE Computer, December 1985, pp. 33-45.

[SCBL 88] Schelvis Marcel, Bledoey Eddy, The implementation of a distributed Smalltalk, OOPSLA 88 Proceedings, 1988, pp. 212-232.

[SCHE 86] Scheifler Robert W. and Gettys Jim, The X Window System, ACM Transactions on Graphics, Vol. 5 n° 2, April 1986, pp. 79-109.

[SCHU 86] Schmucker K.J., MacApp : An Application Framework, Byte, August 1986, pp. 189-192.

[SHAN 89] Shan Yen-Ping, An event-driven Model-View-Controller framework for Smalltalk, In OOPSLA'89 : Object Oriented Programming, Systems and Applications, 1989, pp. 347-352.

[SHAN 90] Shan Yen-Ping, Mode offers direct manipulation for Smalltalk, IEEE Software, vol. 7, n° 3, 1990, 36.

[SHAN 91] Shan Yen-Ping, An Object-Oriented Framework for Direct-Manipulation User Interfaces, pp. 1-19

[SLCA 84] Sluizer S., Cashman P.M., XCP : An Experimental Tool for Supporting Office Procedures, Proc. First Int'l Conf. Office Automation, IEEE-CS Press, Silver Spring, Md., Décembre 1984, pp. 73-80.

[STBO 87] Stefik M., Bobrow D.G., Foster G., Lanning S., Tatar D., WYSIWIS Revised : Early Experiences with Mutliuser Interfaces, ACM Transactions on Office Information Systems, Vol. 5, N° 2, April 1987, pp. 147-167.

[STFO 87] Stefik Mark, Foster Gregg, Bobrow Daniel G., Kahn Kenneth, Lanning Stan, Suchman Lucy, Beyond the Chalkboard : Computer Support for Collaboration and Problem Solving in Meetings, Communications of the ACM, 30-1 January 1987, pp. 32-47.

[TESL 81] Tesler Lary, The Smalltalk environment, Byte, August 1981, pp. 90-147.

[TIMB 90] Timbuktu, Guide de l'utilisateur Timbuktu, Farallon, Farallon Computing Inc., 1990.

[VEGD 86] Vegdahl Steven R., Moving Structures between Smalltalk Images, OOPSLA 86 Proceedings, September 1986, pp. 466-471.

[VIDEa90] Viéville C., Derycke A., Poisson D., Global Network, Local and Regional, for Open Learning with CAI, Proceeding of the seventh ICTE, Seventh International Conference on Technology and Education, Mars 1990, pp. 250-252.

[VIDEb90] Viéville C., Derycke A., Vilers P., Architecture of a collaborative system using Smalltalk and Unix, European Unix User Conference Proceeding, Spring 90, Munich RFA, 25-27 avril 1990, pp. 89-98.

[VIDEc90] Vilers P., Derycke A., Viéville C., Is Smalltalk a good tool for CSCW systems ?, Tools 90, Poster, Paris, Juin 1990.

[VIDE 91] Viéville C., Derycke A., Clément D., Demerval R., Ladesou C., To promote Cooperative Education in Open Learning by a Dedicated Computer Mediated Conference and Groupware. The 12th Educational Computing Organization of Ontario and 8th International Conference on Technology and Education joint Conference, mai 1991, Toronto, Canada, pp. 609-611.

[VIPO 90] Viéville C., Poisson D., Derycke A., Global Network, local and regional, for Open Learning with C.A.I., Seventh International Conference on Technology and Education, Bruxelles, Belgique, 20-22 mars 1990, pp. 250-253.

[VIVI 92] Vilers P., Viéville C., Cantegrit B., Système de Téléconférence en Temps Réel Multimédia, Conférence et Exposition sur l'Automatisation industrielle, Montréal, Québec, Canada, 1-3 Juin 1992, (à paraître).

[WASA 90] Watabe Kazuo, Sakata Shiro, Maeno Kazutoshi, Fukuoka Hideyuki, Ohmori Toyoko, Distributed Multiparty Desktop Conferencing System : MERMAID, Proceedings CSCW 90, ACM Conférence, October 1990, pp. 27-38.

GLOSSAIRE

W.Y.S.I.W.I.S. (What You See Is What I See)

Ce concept a été introduit la première fois dans le projet Colab de Xerox. Il permet d'imager la possibilité, pour des participants à une conférence temps réel, de travailler autour d'un espace public commun au groupe. L'image généralement proposée est celle du tableau noir souvent utilisé dans des réunions classiques. Cet espace public est donc visible de tous et accessible par un protocole (il peut éventuellement y en avoir plusieurs) prédéfini. Lorsque l'espace public est strictement identique sur toutes les stations, on parlera plus précisément de strict W.Y.S.I.W.I.S.. Dans la pratique, on lui préférera une version relâchée. Le relâchement du W.Y.S.I.W.I.S. pourra se faire suivant quatre dimensions qui sont : Espace, Groupe, Congruence et Temps.

C.S.C.W. (Computer Supported Cooperative Work)

Les systèmes C.S.C.W. sont conçus pour permettre des communications d'homme à homme par l'intermédiaire d'ordinateurs connectés en réseau. Ces systèmes englobent un large éventail d'outils qui diffèrent en fonction du type d'interactions (synchrones ou asynchrones) et en fonction de la localisation des participants (réunis dans une même salle ou dispersés géographiquement).

GROUPWARE (traduit en français par le mot Collecticiel)

Sous-ensemble des systèmes C.S.C.W., les Groupwares sont plus spécialement conçus pour une utilisation par des personnes qui ont un objectif (ou tâche) commun à accomplir.

TELEPOINTEUR

Le télépointeur est une extension de la souris utilisée localement sur les stations de travail. Il permet de faire des désignations distantes utiles pour faire référence à la conversation courante. Différents modes de désignation existent pour s'adapter au mieux aux styles des références.

ESPACE PUBLIC ou PARTAGE

Il correspond à l'espace de travail du groupe ou d'un sous-groupe s'il y a eu des relâches du W.Y.S.I.W.I.S.. A travers cet espace, les participants pourront suivre l'activité des autres et pourront échanger des informations. Cet espace étant souvent un compromis entre les besoins du groupe et de l'individu, différentes techniques d'interfaces (exemple les hypertextes) et des métaphores sous-jacentes (Rooms) ont été adoptées pour l'optimiser au mieux.

Les interactions possibles dans cet espace doivent être coordonnées entre les différents participants afin d'assurer une dynamique et une productivité importante dans le groupe ainsi qu'une cohérence globale du système. Il faut également que les protocoles soient adaptés aux styles des réunions.

ESPACE PRIVE

Par opposition à l'espace public, cet espace est strictement réservé et accessible par le participant propriétaire. Aucune interaction ne peut y être faite par quelqu'un d'autre.

Des informations peuvent être évidemment transférées d'un espace à l'autre par des mécanismes du type copier/coller.

ALGORITHME DE CONTROLE (Floor-algorithm)

L'algorithme de contrôle de la conférence est la traduction du protocole de gestion de cette dernière. Plusieurs d'entre eux peuvent être implémentés pour des styles de réunions différents.

INTERACTIONS SYNCHRONES ET ASYNCHRONES

Les interactions synchrones sont possibles lorsque les participants sont connectés ensemble à un instant donné. On parle également, dans ce cas, d'interaction temps réel. Les interactions asynchrones ont lieu lorsque les participants ne sont pas ou ne peuvent pas être réunis ensemble à un instant donné (système de messagerie, etc).

RETRO-ACTION (FEEDBACK)

La rétro-action est un des éléments fondamental des interfaces multi-utilisateurs. Dans notre cas, elle cumule celle de toute interface mono-utilisateur et celle du rendu de l'activité du groupe. Là encore, des compromis doivent être faits pour ne pas

occasionner une gêne trop importante mais pour permettre cependant, un suivi correct de l'activité du groupe. On ne peut pas, par exemple, permettre l'affichage du curseur des souris de chacun. En effet, le clignotement de tous ces curseurs devient rapidement perturbateur. On préférera utiliser des télépointeurs dans des occasions bien particulières. La rétro-action de l'activité des autres participants pourra aussi se faire à travers des filtres. C'est-à-dire que des points de vue différents pourront être adaptés à chaque individu.

MAITRE DE CONFERENCE

Le maître de conférence est la personne qui a en charge la gestion de la conférence. Il est généralement chargé d'ouvrir et de fermer cette dernière. C'est lui qui décidera du début, lorsque toutes les personnes prévues seront connectées ou lorsqu'un délais d'attente sera passé. Il dispose aussi de fonctionnalités supérieures aux autres participants pour pouvoir intervenir si des conflits surviennent. Le rôle qu'il jouera dans la conférence sera fonction du type de protocole (ou d'algorithme) choisi.

COOPERATISATION D'UNE APPLICATION (ou collectivisation)

Contrairement au développement spécifique d'une application multi-utilisateurs spécifique, cette technique permet de récupérer une application mono-utilisateur existante.

B.O.S.	Binary Object Storage
C.S.C.W.	Computer Supported Cooperative Work
E.A.D.	Enseignement A Distance
E.M.A.	Electronic Mail Association
G.D.S.S.	Group Decision Support System
M.R.P.	Manufacturing Ressource Planning
M.V.C.	Modèle Vue Contrôleur
N.F.S.	Network File System
R.N.I.S.	Réseau Numérique à Intégration de Services
R.P.C.	Remote Procedure Call
T.C.A.O.	TeleConference Assistée par Ordinateur
T.C.P.	Transmission Control Protocol
U.I.M.S.	User Interface Management System
W.Y.S.I.W.I.S.	What You See Is What I See

EXEMPLE D'UN PARTAGE D'OBJETS

Nous allons présenter et détailler sur un exemple de connexion et d'initialisation d'un objet partagé, les méthodes caractéristiques du système de partage d'objets.

Les phases que nous allons détailler seront les suivantes :

1 - mise en attente des connexions clientes du serveur.

2 - connexion au serveur d'une station cliente.

3 - le serveur partage sa fenêtre Transcript. L'objet pointé par la variable globale Transcript sera partagé. Il est à noter que cette étape peut, sans problème, être inversée avec la précédente.

4 - La station cliente initialise un 'proxy' et lui envoie un message.

---

1 - Mise en attente des connections clientes du serveur.

```
SerNomServeur <-- NameService new
```

SerNomServeur est une variable globale.

```
Classe : NameService

new
^ super new start
```

Classe : NameService

```
start
  "initialisation des variables d'instances"
  |aSharedObject skt d ns |
  services := Array new: 20. services at:1 put: 'Name Serveur'.
  connections := Array new: 20.
  aSharedObject := SharedObject with: self. aSharedObject gid:1.
  LinkTable := Array new: 20.
  ConnectionProcess inNil ifFalse: [ ConnectionProcess terminate].
  " open the master socket"
  skt _ UnixSocketAccessor newTCPserverAtPort: 6000.
  skt listenFor: 1.
  d _ Delay forSeconds: 5.

  " Création d'un processus chargé de détecter les connexions clientes "
  ConnectionProcess _ [[true] whileTrue: [ns _ nil.
  [ ns isNil ] whileTrue:[ d wait.ns _ skt acceptNonBlock].

  "Initialisation du premier élément de la LinkTable avec l'objet partagé
serveur de nom"
  LinkTable at:1 put: ( ConnectionLink with: aSharedObject on: (Connection
start: ns))]] newProcess.
  ConnectionProcess priority: Processor userInterrupt Priority; resume.
  ^ aSharedObject! !
```

Dans cette méthode, plusieurs points sont à signaler.

. Un processus est démarré sur le serveur pour détecter les connections clientes. Ce processus est associé à une instance de la classe Delay qui permet de le "réveiller" toutes les x secondes pour éviter de monopoliser le processeur.

. La LinkTable est initialisée avec le premier élément qui est le Serveur de Nom. Ce premier élément est une instance de la classe ConnectionLink qui associe le serveur de nom à une connection.

. Nous avons initialisé la taille de la LinkTable avec 20 éléments, mais ce nombre peut être modifié sans problème.

---

2 - La seconde phase est la connection de la station cliente au serveur.

```
SerNomClient <-- ClientNameService new
```

Classe : ClientNameService

```
new
^ super new start
```

Classe : ClientNameService

```
start
" lance une connexion cliente"
  | skt |
  LinkTable := Array new: 20.
  skt _ UnixSocketAccessor newTCPclientToHost: 'ServerHostName' port:
6000.
  LinkTable at:1 put: ( ConnectionLink with: self on: ( Connection start: skt)).
```

La LinkTable de la station cliente est initialisée et une socket est ouverte sur le serveur. D'autre part, une connexion est démarrée sur la station cliente.

Classe : Connection

```
start: socket
  ^ super new start: socket! !
```

Classe : Connection

```
start: skt
  iStream _ ExternalConnection new input: skt ; output: skt; readStream.
oStream _ External Connection new input: skt; output: skt; writeStream.
  iStream binary. oStream binary.
  self input.
  ^ self! !
```

La méthode input démarre un processus qui va scruter le socket pour récupérer les paquets envoyés par les stations (pour une station cliente seule les communications avec le serveur sont possibles, nous sommes en réseau étoilé).

Classe : Connection

```
input

  "Démarre un processus chargé de récupérer les paquets envoyés par les
autres stations"

  | c p o aPolicy |

  c := true.
  [[c] whileTrue: [p _ iStream next. "attente en lecture des paquets"
          aPolicy_(LinkTable at: p) policy. p _ iStream next. "le premier
champ correspond au type du message"
          p = = 3 ifTrue: [self close: self.c _ false ] "un paquet fermé"
          ifFalse: [o _ self unLoad. "remise en forme du message"
          p = = 0 ifTrue: [ [aPolicy execute: o on: self] forkAt: Processor
timingPriority]. "paquet d'exécution"
          p = = 1 ifTrue: [ aPolicy reply: o ]. "paquet réponse"
          p = = 2 ifTrue: [ aPolicy open: o on: self] "paquet ouvert"
          ]]] forkAt: Processor timingPriority!
```

Comme nous le voyons, les champs ajoutés aux messages sérialisés ou aux objets retournés servent à différents tests. Le premier champ est l'identification de l'objet à partir de son identité globale ensuite le choix est effectué sur le type du message.

Chaque station cliente a un processus connection et le serveur a autant de processus connection qu'il y a de stations connectées (nous avons une socket par client).

---

3 - La troisième étape est le partage de l'objet 'Transcript' par le serveur.

Cela se fait en utilisant le serveur de nom et plus précisement par l'objet partagé d'identité globale égale à 1 et qui a sa variable d'instance "localObject" pointant sur le serveur de nom.

```
SerNomServeur addName: 'Transcript' with: Transcript
```

Cette méthode permet d'utiliser l'une des fonctionnalités du serveur de nom, c'est-à-dire ajouter un nom d'objet, lui attribuer une identité globale et renvoyer ce nouvel objet.

Classe : NameService

```
addName: anObject with: aSObject
  | so gid |
  (self findGid: anObject) isNil ifTrue: [gid := self add: anObject].
  so := SharedObject with: aSObject.
  so gid: gid.
  LinkTable at: gid put: (ConnectionLink with: so on: (LinkTable at:1 )
connection).
  ^ so!
```

'Transcript' est une chaîne de caractères qui correspond à l'identificateur de l'objet à partager Transcript.

Le serveur de nom va vérifier si l'identificateur n'a pas encore était déclaré. Il va ensuite créer un "sharedObject" avec une identité globale égale à 2 (attribution de façon incrémentale, seul le 1 était attribué au serveur de nom) et dont la variable locale 'localObject' pointera sur l'objet Transcript. La LinkTable sera ensuite mise à jour. Elle est maintenant composée de deux éléments (le serveur de nom et l'objet Tanscript).

A partir de maintenant, cet objet est accessible par l'ensemble des stations connectées.

---

4 - La quatrième étape est l'accès de cet objet par la station cliente.

Elle se décompose en deux phases : l'initialisation et l'envoie du message.

```
A <-- SerNomClient name: 'Transcript'
```

L'initialisation du proxy sur la station cliente se réalise à partir de l'identificateur de l'objet. D'une façon identique au partage d'un objet, la création d'un proxy se fait par l'intermédiaire du serveur de nom. Comme nous nous trouvons sur une station cliente, nous allons utiliser le proxy correspondant au serveur de nom (premier élément de la LinkTable).

Classe : ClientNameService

```
name: anObject
  | gid aProxy |
  gid := self findGid: anObject.
  gid isNil ifTrue:[^nil ].
  aProxy _ Proxy new gid : gid.
  LinkTable at: gid put: ( ConnectionLink with: a Proxy on: ((LinkTable at: 1)
policy localObject connections at: gid )).
  ^ aProxy! !
```

La première chose à faire consiste à trouver l'identité globale de l'objet partagé dont l'identificateur est 'Transcript'.

La méthode findGid: 'Transcript' n'est pas définie dans cette classe ce qui provoque l'exécution de la méthode "doesNotUnderstand" que nous avons redéfini dans la classe Policy.

Classe : Policy

```
doesNotUnderstand: aMessage
  "execute aMessage and wait for a reply from network"
  self process: aMessage.
  ^ self waitForReply!
```

Classe : ClientNameService

```
process: aMessage
  "send aMessage on the network"
  (LinkTable at:gid) connection send: aMessage for: gid!
```

La connection est récupérée, le message est sérialisé et mis en forme pour être envoyé sur le réseau.

Classe : Connection

```
send: aMessage for: gid
  " sérialisation du message aMessage avec le BOS
    - envoie du résultat sur la socket"
  oStream nextPut: gid ; nextPut: 0. self load: aMessage. oStream commit !
```

Classe : Connection

```
load: anObject
  "sérialisation de l'objet, anObject"
  | bs |
  bs _ BinaryStorage put: anObject
  oStream nextWordput: bs size. oStream nextPutAll: bs . oStream commit !
```

Le processus correspondant à la connection sur le serveur récupère le message. Le premier champ de l'objet sérialisé est égale à 1 car il s'agit du serveur de nom. Le message étant de type exécution, il sera remis en forme pour être exécuté sur le localObject.

Classe : SharedObject

```
execute: aMessage on: aConnection
  "le processus est sous contrôle d'un sémaphore"
  |v|
  v := mutex critical: [localObject perform: aMessage selector
                withArguments: aMessage arguments].
  aConnection reply: v for: gid!
```

La réponse est envoyée sur le réseau, à partir de la connection attachée au policy (ici, il s'agit de l'objet partagé pointant sur le serveur de nom).

L'exécution de la méthode "findGid: 'Transcript" a renvoyé 2 (il correspond au second élément de la LinkTable).

Classe : Connection

```
reply: anObject for: gid
  " - serialization of the object anObject
    - send it on aSocket
  return itself"
  oStream nextPut: gid; nextPut: 1 .self load: anObject. oStream commit!
```

Le processus sur la station cliente associé à la connection reçoit la réponse. D'une façon identique à celle que nous avons précédamment exposée, le message va être remis en forme et comme il s'agit d'un paquet de type résultat, le processus associé à la méthode "name: 'Transcript'"va être débloqué permettant ainsi la poursuite de son exécution.

L'exécution de cette méthode a eu pour conséquence la création d'un proxy avec une identité globale égale à deux et la mise à jour de la LinkTable.

A partir de maintenant, l'utilisation du proxy est transparente pour l'utilisateur. Le système gère les mécanismes d'indirection pour envoyer les messages sur l'objet partagé.

```
A show: 'essai'
```

La station cliente peut utiliser l'ensemble des messages compris par l'objet proxy faisant référence au Transcript.

Comme il s'agit d'un proxy et que le message n'est pas défini, cela va générer le message "doesNotUnderStand: unMessage". Le mécanisme va alors être identique à celui que nous venons de décrire avec pour différence une identité globale différente.

EXEMPLE DE DEVELOPPEMENT D'UNE APPLICATION PARTAGEE

Dans cette annexe nous montrons comment écrire une application coopérative, en nous servant de couples vue/contrôleur définis dans notre boîte à outils. Cette application interactive est composée de deux fenêtres : une fenêtre liste et une fenêtre texte. Dans la fenêtre liste est affiché l'ensemble des textes disponibles. En en sélectionnant un, son affichage est fait dans la fenêtre texte.

Les nouveaux couples vue/contrôleur que nous avons conçus permettent de définir deux types d'interactions possibles en fonction du statut (contrôleur ou non de l'espace public).

Les interactions que nous choisissons sont résumées dans le tableau ci-dessous :

	Contrôle l'espace public			**Ne contrôle pas l'espace public**		
	Souris b. gauche	Souris b. milieu	Clavier	Souris b. gauche	Souris b. milieu	Clavier
Fenêtre liste	Sélection item possible	*Menu* OP1 OP2	Non	Sélection item impossible	*Menu* OP1	Non
Fenêtre texte	Sélection de texte possible	*Menu* COPY CUT PASTE ACCEPT	Oui	Sélection de texte possible	*Menu* COPY	Non

Les figures suivantes montrent ce que l'on peut voir sur les stations des utilisateurs

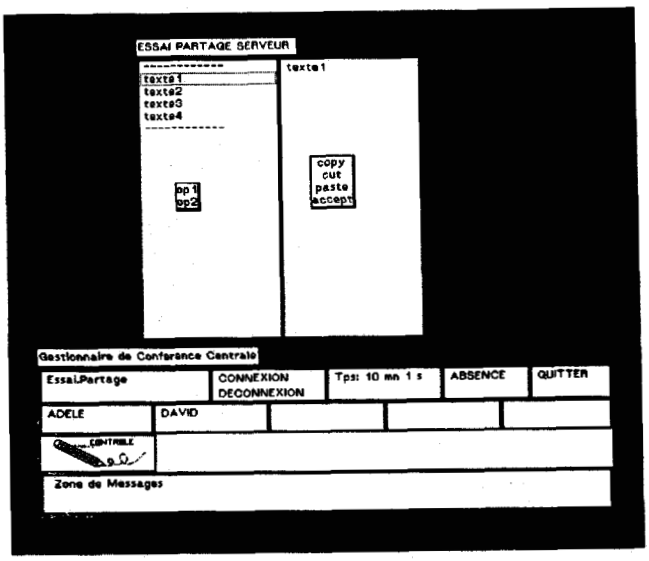


Figure 1. *Le contrôle est attribué à Adèle.*

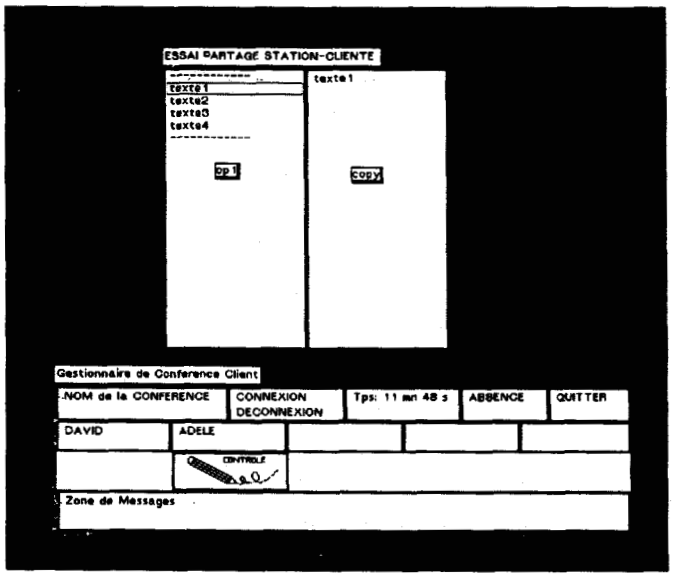


Figure 2. *Le contrôle est attribué à Adèle. Station cliente.*

Les ouvertures des fenêtres sur le serveur et les stations clientes se font respectivement par les messages suivants :

EssaiPartageVue nouveauServeur

EssaiPartageVue nouveauClient

Nous remarquerons que pour l'application implémentée sur le serveur, son modèle, instance de la classe EssaiPartageModele, est initialisé comme un objet partagé. Le modèle de celle implémentée sur la station cliente est un proxy. Comme

nous l'avons montré, il faut un échange d'informations entre le gestionnaire de conférence et l'application que nous réalisons en initialisant la variable d'instance "topViewAppli" des gestionnaires de conférence client et serveur.

Les méthodes définies dans la classe du modèle se développent d'une façon tout à fait classique. Le programmeur n'a pas à gérer les mises à jour des vues distantes. Il n'y a pas non plus de test à faire pour renvoyer le menu correspondant au statut de l'utilisateur. Aux méthodes op1 et op2 peuvent être associées n'importe quelles actions. Avec le couple vue/contrôleur utilisé pour la fenêtre texte, les mises à jour sont faites pour chaque caractère frappé.

La figure suivante montre l'écran d'un utilisateur situé sur une station cliente. N'ayant pas le contrôle de l'espace public, il peut néanmoins copier du texte de l'espace public dans son espace privé pour le travailler.

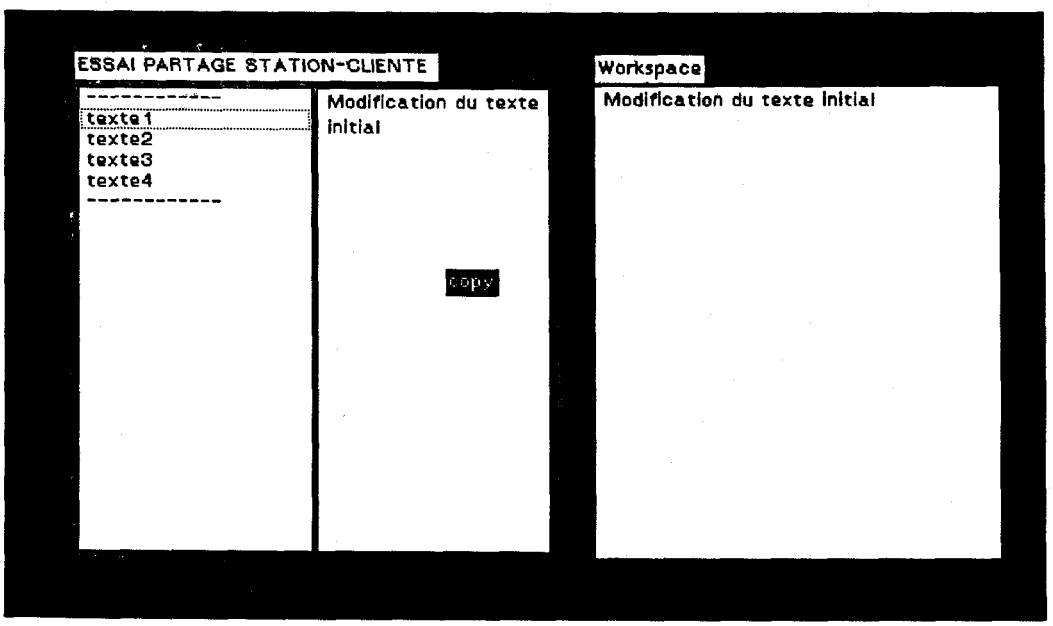


Figure 3. *Exemple d'action qu'un utilisateur n'ayant pas le contrôle peut faire.*

```
Model subclass: #EssaiPartageModele
    instanceVariableNames: 'item ind listeItem conItem '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Conference-Application'
```

Catégorie instance : 'menu'

menu1

"menu renvoyé dans la fenêtre liste lorsque le participant en a le contrôle "

```
^ ActionMenu
        labels: 'op1\op2' withCRs
        lines: #()
        selectors: #( op1 op2 ).
```

menu1Part

"menu renvoyé dans la fenêtre liste lorsque le participant n'en a pas le contrôle "

```
^ ActionMenu
        labels: 'op1' withCRs
        lines: #()
        selectors: #( op1 ).
```

menu2

"menu renvoyé dans la fenêtre texte lorsque le participant en a le contrôle "

```
^ ActionMenu
        labels: 'copy\cut\paste\accept' withCRs
        lines: #()
        selectors: #( copySelection cut paste accept ).
```

menu2Part

"menu renvoyé dans la fenêtre texte lorsque le participant n'en a pas le contrôle "

```
^ ActionMenu
        labels: 'copy' withCRs
        lines: #()
        selectors: #( copySelection ).
```

Catégorie instance : 'texte'

acceptText: aText from: whoCare

```
^ true
```

text

"renvoie le texte correspondant à l'item courant sélectionné de la fenêtre liste "

```
| selectionCourante |
selectionCourante _ (dependents at: 2 ) selection .
selectionCourante = 0        ifTrue: [^ '' asText]
                             ifFalse: [ ^ conItem at: selectionCourante ]
```

Catégorie instance : 'private'

item

```
^ item
```

item: anOb

"cette méthode est invoquée lorsque l'on sélectionne un item dans la fenêtre liste"

"L'item courant devient celui sélectionné"

```
item _ anOb.
```

"on avertit le modèle qu'il y a une modification"

```
self changed: #text
```

Catégorie instance : 'initialisation'

initialise

"initialisation de deux variables d'instance"

```
listeItem _ OrderedCollection new.
listeItem       add: #texte1 ;  add: #texte2 ;
                add: #texte3 ; add: #texte4.

conItem _ OrderedCollection new.
conItem       add: 'texte1' asText ;  add: 'texte2' asText ;
              add: 'texte3' asText ; add: 'texte4' asText .
```

Catégorie classe : 'creation'

new

```
^ super new initialise
```

```
Object subclass: #EssaiPartageVue
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Conference-Application'
```

Catégorie instance : 'initialisation'

initialiseClient

```
"Cette méthode sert à créer la fenêtre applicative sur le serveur"

    | listeVue texteVue mod topVue |

"modèle de l'application est le proxy sur le modèle serveur
on récupère ce proxy par l'intermédiaire du gestionnaire"
    mod _ TopGesCli model modelAppli .

"Création de la vue principale"
    topVue _ StandardSystemPartageClientView new.
    topVue      label: 'ESSAI PARTAGE STATION-CLIENTE' ;
        minimumSize: 300@300 ; maximumSize: 300 @ 300 ;
        borderWidth: 3 ; model: mod.

"Création de la fenêtre liste avec les deux sélecteurs de messages pour les menus"
    listeVue _ SelectionInListPartageClientView on: mod
                                          printItems: true
                                          oneItem: false
                                          aspect: #item
                                          change: #item:
                                          list: #listeItem
                                          menu: #menu1
                                          menuPart: #menu1Part
                                          initialSelection: nil .
    listeVue numIndice: 1 .
```

```
"Création de la fenêtre liste avec les deux sélecteurs de messages pour les menus"
        texteVue _ CodePartageClientView        on: mod
                                                aspect: #text
                                                change: #acceptText:from:
                                                menu: #menu2
                                                menuPart: #menu2Part
                                                initialSelection: nil .
        texteVue newText: ' ' asText.
        texteVue numIndice: 2 .

        topVue addSubView: listeVue in: (0@0 extent: 0.5@1) borderWidth: 2.
        topVue addSubView: texteVue in: (0.5@0 extent: 0.5@1) borderWidth: 2.

"Initialisation de la variable d'instance topViewAppli du gestionnaire conférence"
        TopGesCli model topViewAppli: topVue.

"Comme l'ouverture de la fenêtre est faite de façon distante, une position d'origine doit être
spécifiée"
        topVue controller open1: 100@100
```

initialiseServeur

```
"Cette méthode sert à créer la fenêtre applicative sur le serveur"
        | listeVue texteVue mod  topVue |

        mod _ EssaiPartageModele new .          "modele de l'application"

"Création de la vue principale"
        topVue _ StandardSystemPartageView new.
        topVue label: 'ESSAI PARTAGE SERVEUR' ;
                minimumSize: 300@300 ; maximumSize: 300 @ 300 ;
                borderWidth: 3 ; model: mod.

"Création de la fenêtre liste avec les deux sélecteurs de messages pour les menus"
        listeVue _ SelectionInListPartageView   on: mod
                                                printItems: true
                                                oneItem: false
                                                aspect: #item
                                                change: #item:
                                                list: #listeItem
                                                menu: #menu1
                                                menuPart: #menu1Part
                                                initialSelection: nil .
        listeVue numIndice: 1 .
```

```
"Création de la fenêtre liste avec les deux sélecteurs de messages pour les menus"
    texteVue _ CodePartageView on: mod
                               aspect: #text
                               change: #acceptText:from:
                               menu:#menu2
                               menuPart: #menu2Part
                               initialSelection: nil .
    texteVue newText: ' ' asText.
    texteVue numIndice: 2 .

    topVue addSubView: listeVue in: (0@0 extent: 0.5@1) borderWidth: 2.
    topVue addSubView: texteVue in: (0.5@0 extent: 0.5@1) borderWidth: 2.

"Initialisation de la variable d'instance topViewAppli du gestionnaire conference"
    TopGesSer model topViewAppli: topVue.

    topVue controller open
```

Catégorie classe : 'creation'

nouveauClient

```
^ super new initialiseClient
```

nouveauServeur

```
^ super new initialiseServeur
```

REGLES ET PRESENTATION DU JEU DU KANBAN

Cette annexe est complémentaire à la présentation du jeu faite dans la première partie du troisième chapitre.

Ce jeu, appelé "Jeu du Kanban", est un jeu à vocation pédagogique. Il s'agit d'une simulation d'atelier qui permet d'apprécier les avantages et les inconvénients de deux méthodes différentes de planification :

. la méthode MRP (Manufacturing Resource Planning),

. la méthode Kanban.

La simulation est prévue sur 8 semaines, divisées en 4 périodes de semaines qui correspondent chacune à une certaine forme d'organisation de l'atelier.

1 - Présentation générale

1.1 - Données de base

L'atelier est composé de 5 postes de travail, et chaque poste est tenu par une équipe de joueurs (1 ou 2). Une sixième équipe joue le rôle de responsable de Direction. L'animateur est chargé de veiller au bon déroulement du jeu et tient généralement le rôle du responsable de direction. Les produits sont représentés par des jetons, de couleurs et de formes différentes, qui sont placés dans des containers qui circulent entre les postes.

1.1.1 - Durée du Jeu

La durée normale est d'un jour pour la simulation. Celle-ci peut être suivie d'une ou plusieurs journées de réflexion en groupe pour tirer le meilleur parti des observations faites au cours du jeu. Suivant l'entreprise, il pourra s'agir d'engager une réflexion sur l'amélioration de la qualité, l'amélioration de la fiabilité des machines,

etc..., ou sur la possibilité pratique de mettre en oeuvre la méthode Kanban dans un atelier.

1.1.2 - Disposition des joueurs

Les joueurs prennent place autour d'une table, les postes pouvant être disposés de différentes façons. Un exemple de disposition a été donné dans le chapitre trois.

1.1.3 - La société

L'origine de la Société se situe vers 1936, avec la création par un ouvrier professionnel d'un petit atelier de sous-traitance mécanique. Mais la véritable expansion date de 1952, quand le fils du fondateur déposa un brevet d'invention relatif à un "réducteur à engrenages planétaires". La Société a pris à cette époque le nom de REDIX, et se consacre depuis, à la fabrication et à la vente de mécanismes de réduction de vitesse utilisés dans l'industrie de la manutention, du bâtiment et des travaux publics. Aujourd'hui, REDIX emploie 120 personnes dans son usine de VILLENEUVE (AVEYRON). Son chiffre d'affaires est de 80.000.000 francs, en expansion régulière de 10 % par an. La production qui fait l'objet du JEU du KANBAN ne concerne qu'un des ateliers de l'usine, celui qui fabrique les réducteurs de faible puissance.

Remarque : pour des raisons de simplicité, seuls six produits finis ont été considérés, avec un processus de production également simplifié.

Ventes : Le marché sur lequel évolue REDIX pour les réducteurs de faible puissance est un marché de série. Les ventes sont faites à partir d'un stock de produits finis, ce qui permet d'expédier au client "à lettre vue". La production est faite en fonction des prévisions de vente.

1.1.4 - Les produits

Les produits finis ont tous la même structure, représentée par la nomenclature ci-dessous.

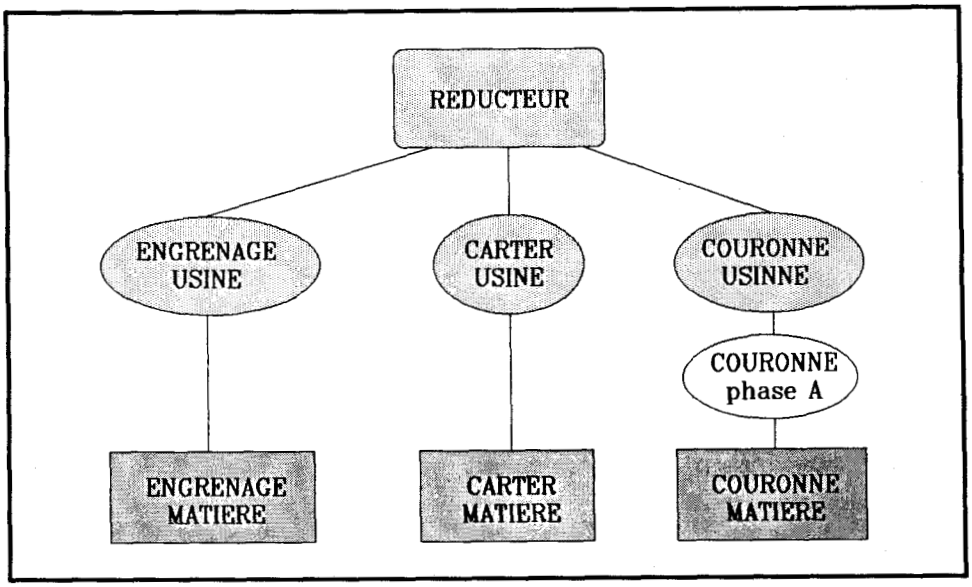


Figure 1. *Nomenclature des produits*

1.2 - L'échelle de temps

L'unité élémentaire de temps est l'heure. Il y a 8 heures de travail par jour, 5 jours par semaine. Le pas de simulation est l'heure, c'est-à-dire que le jeu avance d'heure en heure. A chaque pas, il y a soit production d'un container si la machine est en activité, soit arrêt de la machine (panne, réglage, etc...).

Les deux postes Usinage Couronnes (phase A et phase B) ont une cadence double des autres postes (200 pièces à l'heure au lieu de 100). Comme dans le jeu, ces containers contiennent deux fois plus de pièces (200 au lieu de 100), la cadence reste toujours de 1 container par pas de simulation.

La synchronisation des joueurs est assurée par l'animateur, qui affiche l'heure de la semaine à l'aide du calendrier prévu à cet effet. L'utilisation des feuilles de poste aide également à la synchronisation des joueurs.

1.3 - Gestion des matières et des stocks

Les matières sont représentées par des jetons de forme et de couleur différentes. Un jeton représente toujours 100 pièces usinées. Pour faire un produit fini, il faut donc regrouper dans un même container 3 jetons de formes différentes.

. Les containers d'engrenages et de carters contiennent 100 pièces, soit 1 jeton.

. Les containers de couronnes contiennent 200 pièces, soit 2 jetons identiques.

. Les containers de produits finis contiennent 100 produits finis, soit 3 jetons différents (1 de chaque pièce usinée). Les approvisionnements en provenance des fournisseurs sont normalement disponibles, sans limitation de quantité.

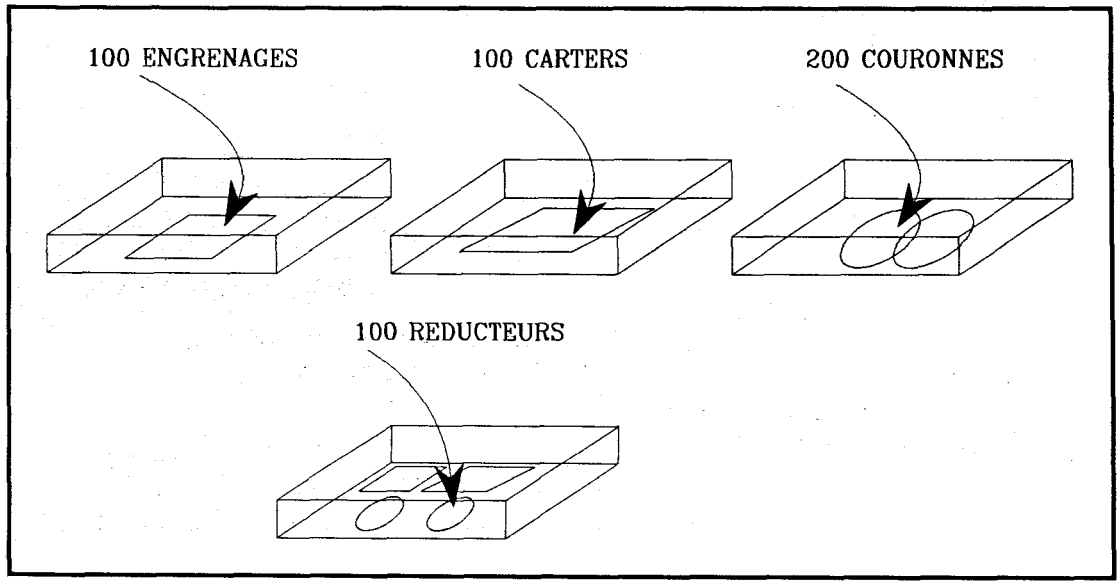


Figure 2. *Représentation des différents produits*

1.4 - Le processus de fabrication

L'atelier est composé de 5 postes de travail, dont 4 postes d'usinage et 1 poste de montage. Une fois les pièces usinées, elles sont rangées dans un magasin de pièces qui est situé près de la section de montage. Les couronnes usinées en phase A sont également mises en stock, en attendant d'être prises en phase B. Le stock de produits finis est rangé dans un magasin, à proximité de la ligne de montage.

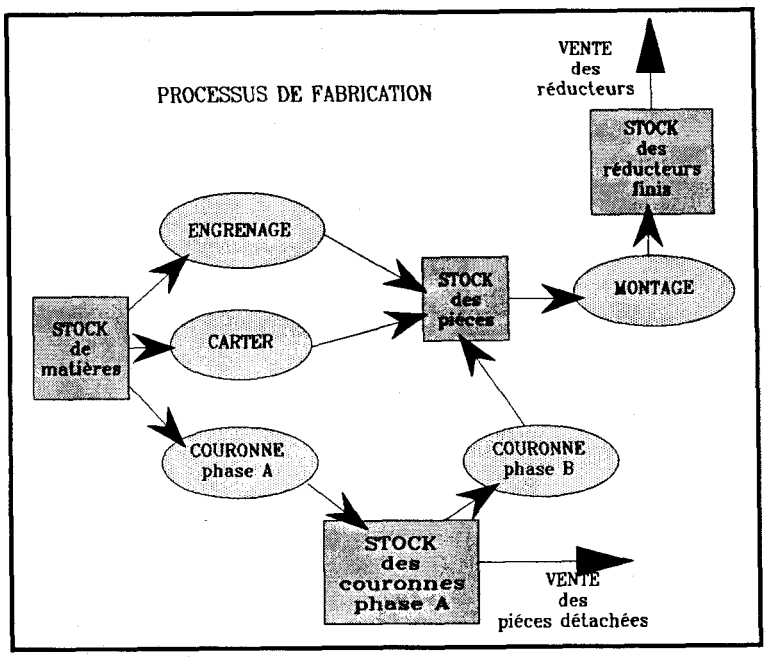


Figure 3. *Processus de fabrication.*

Chaque composant est d'abord usiné, à partir de matière première prise dans le stock, puis assemblé pour réaliser le produit fini. Il y a, d'autre part, des ventes de pièces semi-finies qui sont destinées à la rechange. Cette vente concerne exclusivement le composant "couronne phase A".

Les produits finis existent en 6 références, qui résultent de la combinaison au montage de composants différents. Chaque référence correspond à un rapport de réduction différent.

Engrenage	Deux références (jetons : jaune, blanc)
Carter	Deux références (jetons : rouge, bleu)
Couronne	Trois références (jetons : blanc, vert, rouge)

La façon dont les composants se combinent pour constituer les produits vendus est représentée sur le tableau ci-dessous :

Réducteurs finis

	R1	R2	R3	R4	R5	R6
Engrenage	jaune	jaune	jaune	blanc	blanc	blanc
Carter	rouge	rouge	rouge	rouge	bleu	bleu
Couronne	blanc	vert	rouge	blanc	vert	rouge

Pièces de rechanges

Uniquement la pièce : COURONNE blanc en phase A.

1.5 - Les machines

L'organisation des machines est conçue de telle façon qu'il y ait un équilibrage satisfaisant entre les capacités installées et les ventes.

Les cadences normales de travail sont les suivantes :

Engrenages	100 pièces à l'heure
Carters	100 pièces à l'heure
Couronnes (phase A)	200 pièces à l'heure
Couronnes (phase B)	200 pièces à l'heure
Montage	100 produits finis à l'heure

La cadence moyenne sur une période peut différer de la cadence instantanée. Il faut en effet tenir compte des changements de série, des pannes de machines et des autres incidents susceptibles de ralentir la production.

Pour passer d'une référence à une autre, il est nécessaire de réaliser un changement de série. Les temps de changement de série sont les suivants (au début de la simulation) :

Engrenages	3 heures
Carters	3 heures
Couronnes (phase A)	2 heures
Couronnes (phase B)	4 heures
Montage	1 heure

Une machine peut être dans trois états :

- soit elle produit au rythme indiqué plus haut,

- soit elle est arrêtée pour un incident (panne, etc...),

- soit elle est arrêtée sur décision du joueur parce qu'il n'y a plus de travail planifié.

Au cours de la simulation, l'état de la machine est noté sur une feuille de poste.

Les trois types de composants sont matérialisés par des jetons de forme différente, dont les couleurs constituent les variantes.

Coûts des produits :

Engrenage usiné	25 F
Carter usiné	55 F
Couronne usinée phase A	9 F
Couronne usinée phase B	10 F
Produit fini	100 F

Prix de vente des produits :

Chaque produit est vendu 150 F, la pièce de rechange 30 F.

1.6 - La main-d'oeuvre de production

L'atelier emploie au total, dans ses 5 postes, 20 personnes, 8 heures par jour et 5 jours par semaine.

Il est important de souligner qu'au cours de la simulation le volume des effectifs n'est pas explicitement géré par les participants. C'est le planning des machines qui est pris en considération pour déterminer la cadence de production.

Sauf incident signalé, on admet qu'il y a équilibre satisfaisant entre les effectifs présents et la production prévue de sorte que l'utilisation de la capacité main d'oeuvre n'entre pas en ligne de compte dans les décisions des participants.

Quand un joueur décide d'arrêter une machine pour des raisons de planning, il peut considérer que la main d'oeuvre est ré-employée ailleurs.

1.7 - Changement de série

Lorsqu'un poste décide de changer de série, il doit s'interrompre pendant la durée prévue et noter le changement sur la feuille de poste.

Si un incident, tel qu'une panne machine, survient pendant le changement de série, il est considéré comme survenant au démarrage de la nouvelle série.

La décision de changer de série est toujours prise par le poste de travail, mais les modalités de cette décision diffèrent suivant la méthode de programmation de la production.

Au cours de la phase I du jeu (méthode MRP), ces décisions devront être programmées en début de semaine.

Au cours des phases suivantes (méthode KANBAN), ces décisions seront prises au fur et à mesure de la fabrication.

1.8 - Incidents et informations

Le JEU DU KANBAN est destiné à simuler le fonctionnement d'un atelier dans des conditions représentatives de la réalité vécue en entreprise. Il est donc normal que des événements variés viennent interférer avec le fonctionnement prévu.

Il y a deux types d'événements possibles :

- l'incident de fabrication ou d'approvisionnement, qui surgit de façon aléatoire ;

- l'information qui évoque des questions d'ordre général à des moments prévus dans le déroulement du jeu.

Les événements sont matérialisés par des cartes. Il y a deux paquets de cartes, un pour chaque type d'événements.

1.8.1 - Incidents

Comme dans toute usine, il y a chez REDIX des incidents pendant la fabrication.

Ces incidents, qui recouvrent tous les événements incontrôlés perturbant la production, jouent un rôle important dans la pédagogie. En effet :

- Les méthodes MRP et KANBAN présentent des différences significatives dans leur façon de réagir face à des perturbations.

- C'est en grande partie la réduction de ces perturbations qui permet de "tendre le flux" dans la méthode KANBAN.

- L'analyse des incidents et la recherche de solutions permettront au groupe de travailler sur des données concrètes.

A chaque incident correspond une carte INCIDENT.

Ces cartes forment un paquet placé par l'animateur au centre de la table. Les cartes ne sont pas triées, l'ordre des incidents n'étant pas déterminé à l'avance. A chaque pas du jeu, on lance un dé. Si le résultat est 1 ou 6, il y a un incident et on tire une carte. Sur la carte apparaissent trois conséquences possibles de l'incident, de gravité croissante. Le poste qui a reçu l'incident (n° marqué sur la carte), tire alors une nouvelle fois le dé. Suivant le résultat, la conséquence correspondante est appliquée.

CARTE INCIDENT Poste engrenage

Nature : Ralentissement de production

Causes : Pour faire face au surcroit de commande, il a été décidé de travailler un jour férié.

Conséquences

1 Compte tenu du personnel qui a accepté de venir travailler, une répartition satisfaisante du travail a pu être préparée par le bureau de production
Pas d'arrêt machine

2 Le chef de fabrication perd son temps à répartir l'effectif disponible.
1 h d'arrêt à l'usinage des engrenages.

3 Le personnel a exceptionnellement accepté de venir dans sa totalité à la demande de la Direction Générale. On a oublié de préparer les approvisionnements.
Arrêt de l'ensemble de l'atelier d'usinage : 2h

Concepts : Cohérence de l'organisation.

Figure 4. *Exemple d'une carte incident.*

Conséquences d'un incident :

Comme on s'en rend compte en parcourant le paquet de CARTES INCIDENTS,

les conséquences d'un incident peuvent être multiples. Elles diffèrent par leur nature et leur gravité. Les différentes conséquences possibles sont :

1. Arrêt macnine : immédiat, à exécuter au moment choisi par le joueur mais dans un délai donné ou à exécuter au démarrage d'une nouvelle série.

2. Pièces rebutées, pour cause de non-conformité. Les pièces sont purement et simplement retirées de leur container.

3. Retouches et reprises :

- en ligne : les pièces doivent repasser une nouvelle fois sur la machine indiquée (reprise),

- hors ligne : les pièces sont retirées du stock d'en-cours pour une durée donnée (retouche).

4. Refus clients : pièces à refabriquer, annulation de commande.

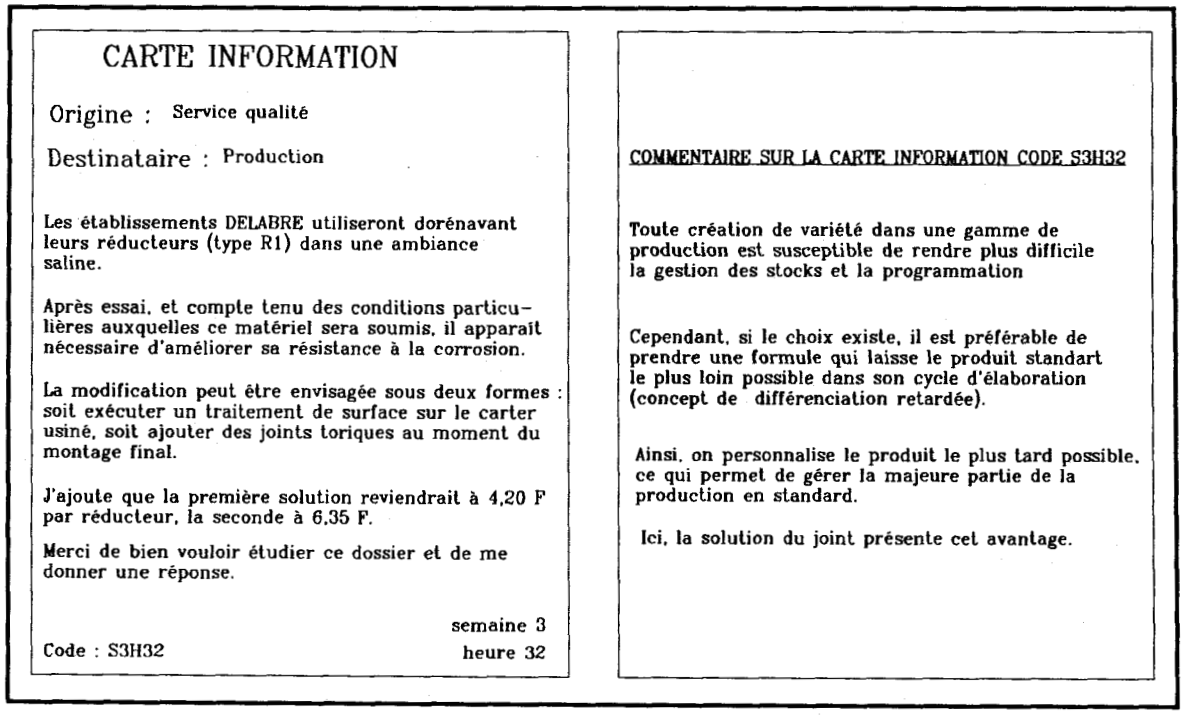
5. Pièces indisponibles.

Ce dernier cas correspond à des opérations de tri, à des pièces en cours de contrôle ou en cours de retouche. On matérialisera cette situation en mettant de côté les containers pendant la durée indiquée sur la carte.

1.8.2 - Information

Les cartes d'information sont tirées à des instants précis par le responsable des ventes. Il dispose en effet d'un dossier de direction dans lequel se trouve le planning des ventes. Ce planning des ventes, dont il est le seul à prendre connaissance, contient les ventes à effectuer et les cartes d'information à tirer aux instants précis.

Ces cartes permettent de faire réfléchir les participants sur différents sujets, et permettent de remplir les conditions nécessaires au bon déroulement des méthodes de planification qui évoluent tout au long du jeu.



CARTE INFORMATION

Origine : Service qualité

Destinataire : Production

Les établissements DELABRE utiliseront dorénavant leurs réducteurs (type R1) dans une ambiance saline.

Après essai, et compte tenu des conditions particulières auxquelles ce matériel sera soumis, il apparaît nécessaire d'améliorer sa résistance à la corrosion.

La modification peut être envisagée sous deux formes : soit exécuter un traitement de surface sur le carter usiné, soit ajouter des joints toriques au moment du montage final.

J'ajoute que la première solution reviendrait à 4,20 F par réducteur, la seconde à 6,35 F.

Merci de bien vouloir étudier ce dossier et de me donner une réponse.

semaine 3

Code : S3H32 heure 32

COMMENTAIRE SUR LA CARTE INFORMATION CODE S3H32

Toute création de variété dans une gamme de production est susceptible de rendre plus difficile la gestion des stocks et la programmation

Cependant, si le choix existe, il est préférable de prendre une formule qui laisse le produit standart le plus loin possible dans son cycle d'élaboration (concept de différenciation retardée).

Ainsi, on personnalise le produit le plus tard possible, ce qui permet de gérer la majeure partie de la production en standard.

Ici, la solution du joint présente cet avantage.

Figure 5. *Exemple d'une carte information.*

2 - Déroulement du jeu

2.1 - Rôle des équipes

Il y a cinq postes de travail et un poste de responsable de direction.

2.1.1 - Rôle de chaque équipe de travail

- Elles programment la fabrication sur chacun des postes (quantité à faire pour la semaine, détail de chaque lancement), en tenant compte des informations disponibles sur demande auprès de la Direction.

- Elles sont confrontées à des incidents de fabrication, (qualité, pannes, fournisseurs en retard, etc...), dont elles subissent les conséquences, et qui peuvent leur imposer de modifier leur programmation. Les incidents sont représentés par des cartes tirées au hasard avec des dés.

- Elles débattent entre elles des thèmes évoqués chaque semaine dans les Cartes Information.

- A la fin de chaque phase du jeu, elles dégagent les points-clés relatifs à la méthode de planification qui vient d'être appliquée.

- A la fin du jeu, elles mesurent les résultats atteints : satisfaction des clients, efficacité industrielle, qualité.

- Le rôle du responsable de Direction est d'assurer les expéditions, et de mettre en jeu les Cartes Information.

Il dispose en outre de certaines informations qui peuvent être communiquées aux équipes de postes, à la condition que celles-ci en fassent la demande précise.

2.1.2 - Rôle du responsable de direction

1) Il assure les expéditions et enregistre les manquants

A partir de la liste des demandes, il effectue les livraisons aux clients. En pratique, cela consiste à enlever les containers des références demandées du stock (produit fini ou pièces détachées).

En cas de pièces manquantes, il note sur la feuille prévue à cet effet (feuille de livraison - figure 6) les produits manquants. Il y a une feuille de livraison pour 4 semaines, 2 feuilles sont donc nécessaires pour la durée du jeu.

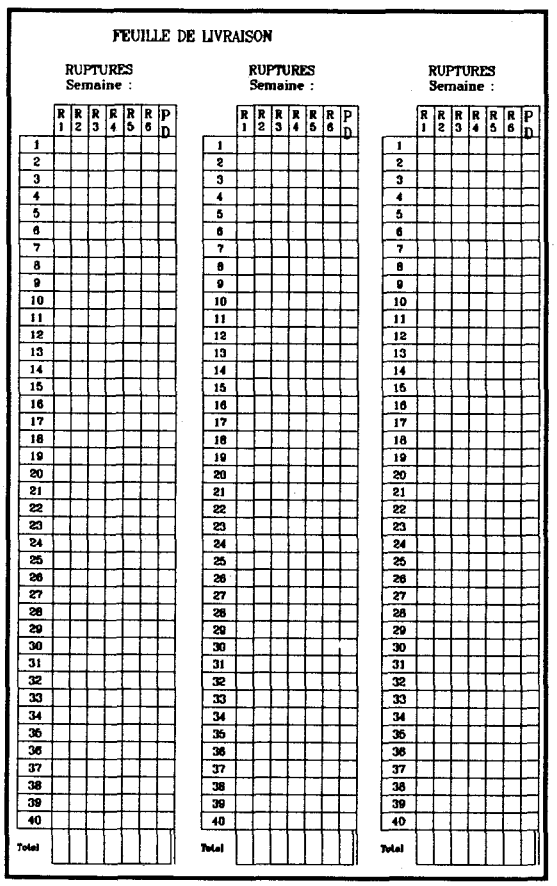
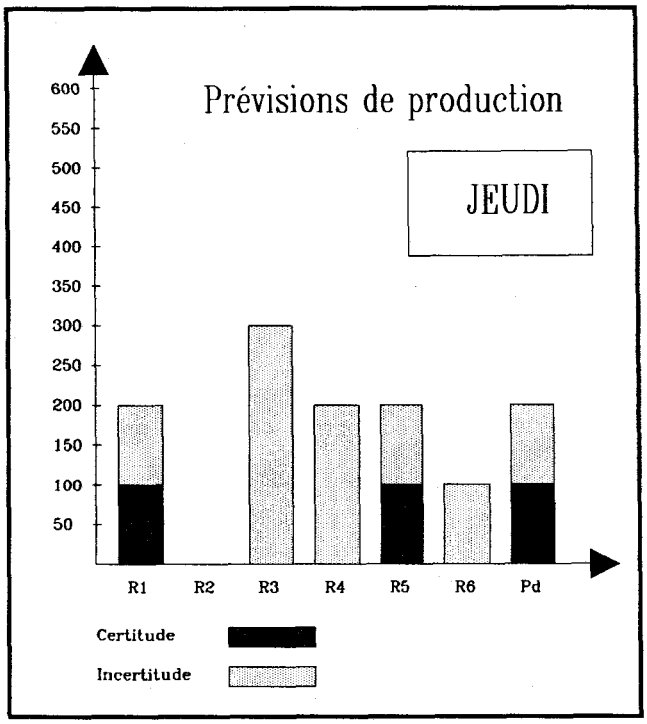


Figure 6. *Feuille de livraison*



STATISTIQUES DES VENTES PAR SEMAINE

	Prévisions	Variations
R1	700	100
R2	100	100
R3	550	150
R4	1050	150
R5	500	200
R6	400	100
Couronnes A	1200	400

Figures 7a et 7b. *Prévisions des ventes et des écarts probables.*

Le responsable commercial peut demander à la production, en cas de manquants, de prendre toutes les dispositions pour livrer le plus rapidement possible. Ceci peut impliquer de changer de référence en en-cours au montage avant la fin de la série programmée.

2) Il communique aux postes certaines informations commerciales à leur demande

En tant que responsable commercial, il dispose de différentes informations sur les ventes :

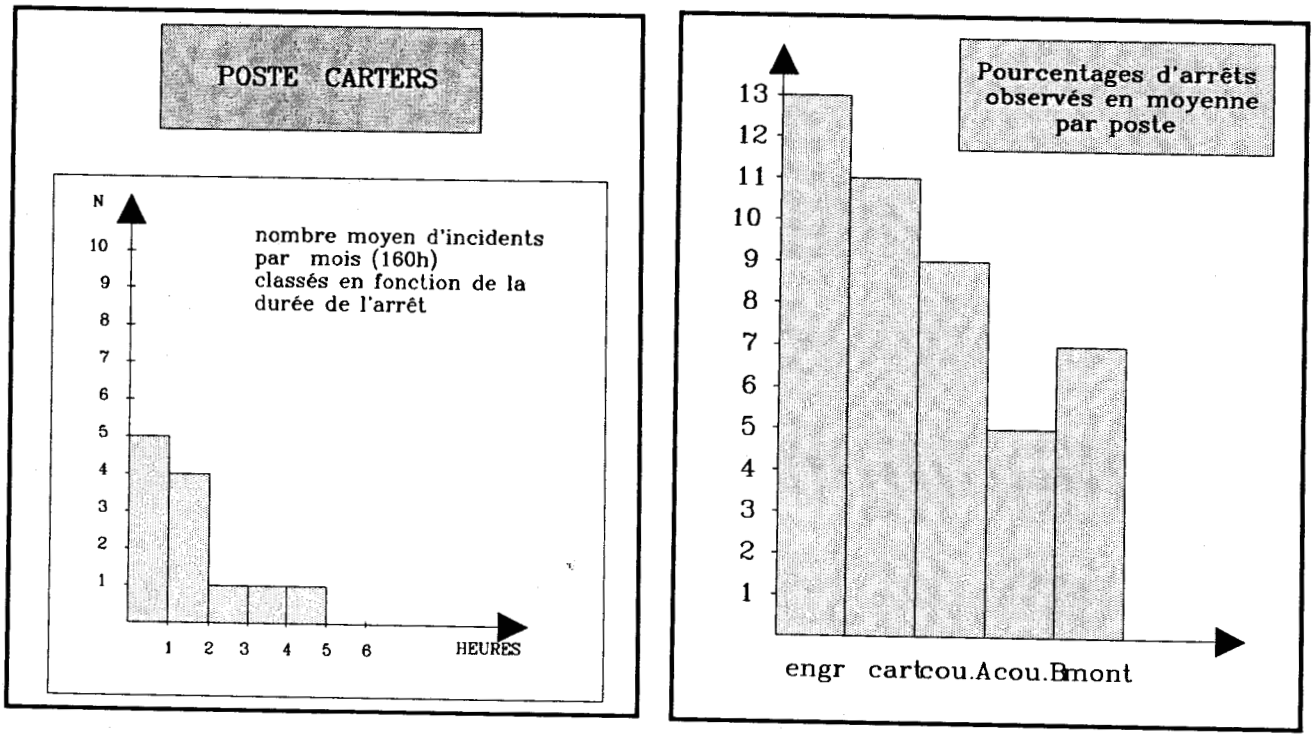
- Statistiques des ventes par référence (Prévision de vente : figure 7a)
- Ecarts probables prévisions/ventes (figure 7b).

Les demandes d'expédition ne sont pas exactement connues à l'avance. Au cours du jeu, il faudra par conséquent faire face à des variations.

Pour programmer leur production, les postes peuvent faire appel aux informations statistiques évoquées plus haut. Le responsable commercial communique l'information en transmettant la feuille correspondante au poste qui en fait la demande. On notera que, si un poste n'exprime pas une demande d'information, celle-ci n'a pas à lui être communiquée.

3) Il communique aux postes en faisant la demande certaines informations techniques

En tant que responsable technique, il dispose de différentes informations sur le fonctionnement de l'atelier :



Figures 8a et 8b. *Histogrammes des arrêts.*

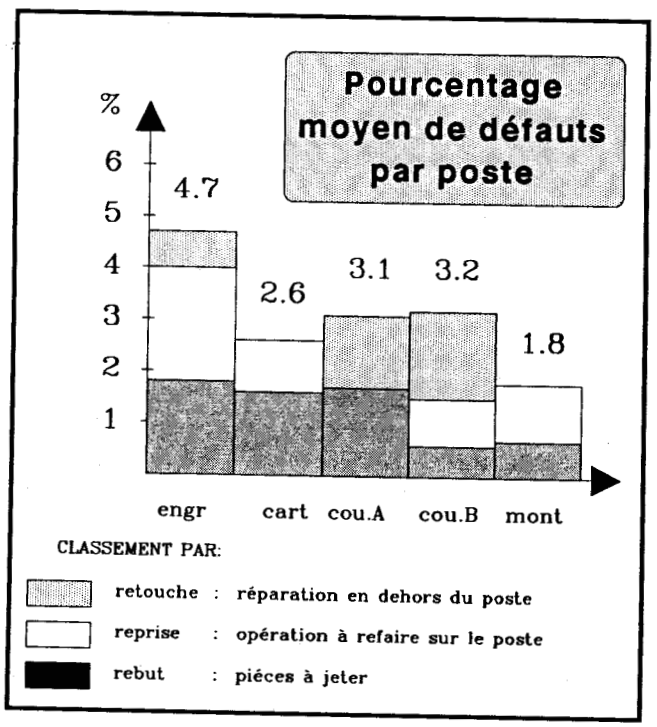


Figure 9. *Pourcentages moyens de défauts.*

- Histogrammes des arrêts, pour chaque poste (figure 8a)
- Pourcentage moyen d'arrêt par poste (figure 8b)
- Taux de défaut moyen des fabrications (figure 9).

Il s'agit d'informations statistiques, mesurées au cours des mois écoulés. Comme pour les statistiques de ventes, ces informations ne doivent être communiquées qu'aux postes qui en font la demande explicite.

4) Il déclenche le tirage des cartes informations

Au début du jeu, l'animateur pose sur la table, des CARTES INFORMATION. Celles-ci sont classées dans l'ordre chronologique. Leur tirage est déclenché par un code qui figure sur le dossier des expéditions, dans la colonne intitulée INFORMATIONS.

Le responsable de Direction lit, au moment voulu, la carte information correspondante.

Par exemple, la carte S1H19 doit être lue en semaine 1 à la 19ème heure.

Les cartes de type S3H0 doivent être lues juste avant de début la 3ème semaine.

5) Il effectue les inventaires

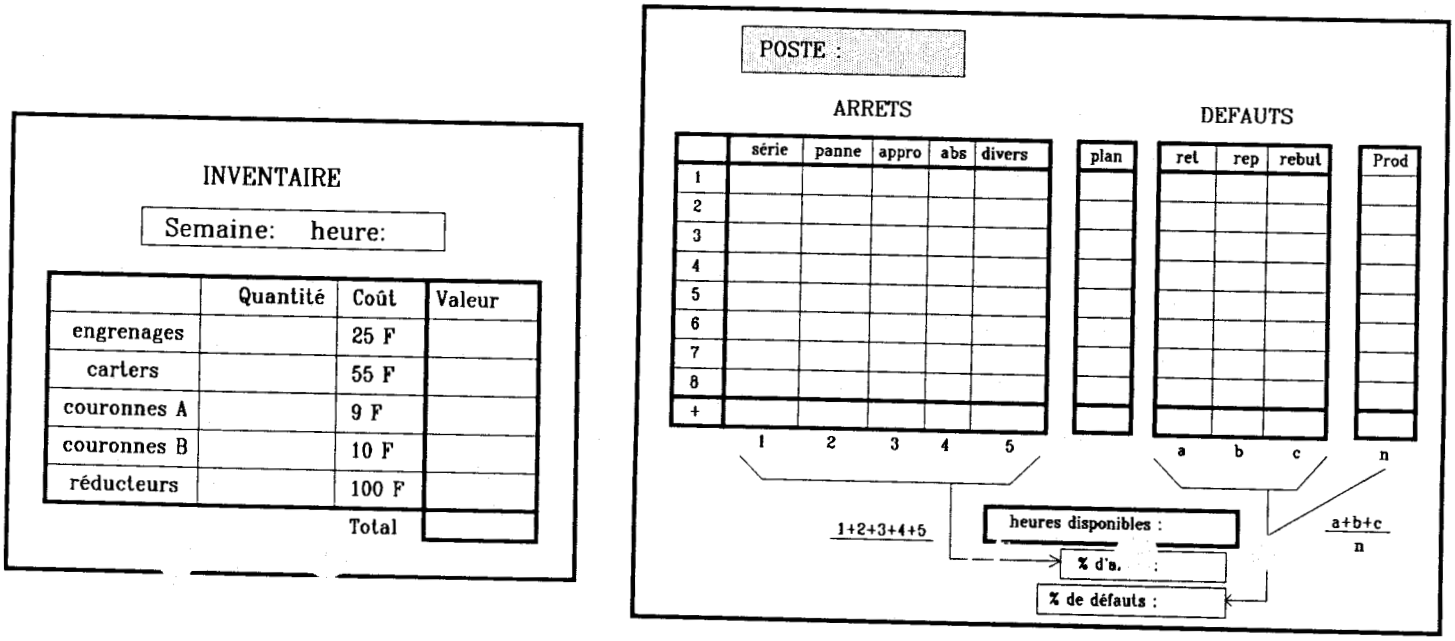
Au moment jugé opportun, le groupe décide de comptabiliser les stocks.

Le responsable de Direction dispose d'une feuille d'inventaire, qu'il remplit en comptant les jetons dans les containers (figure 10a).

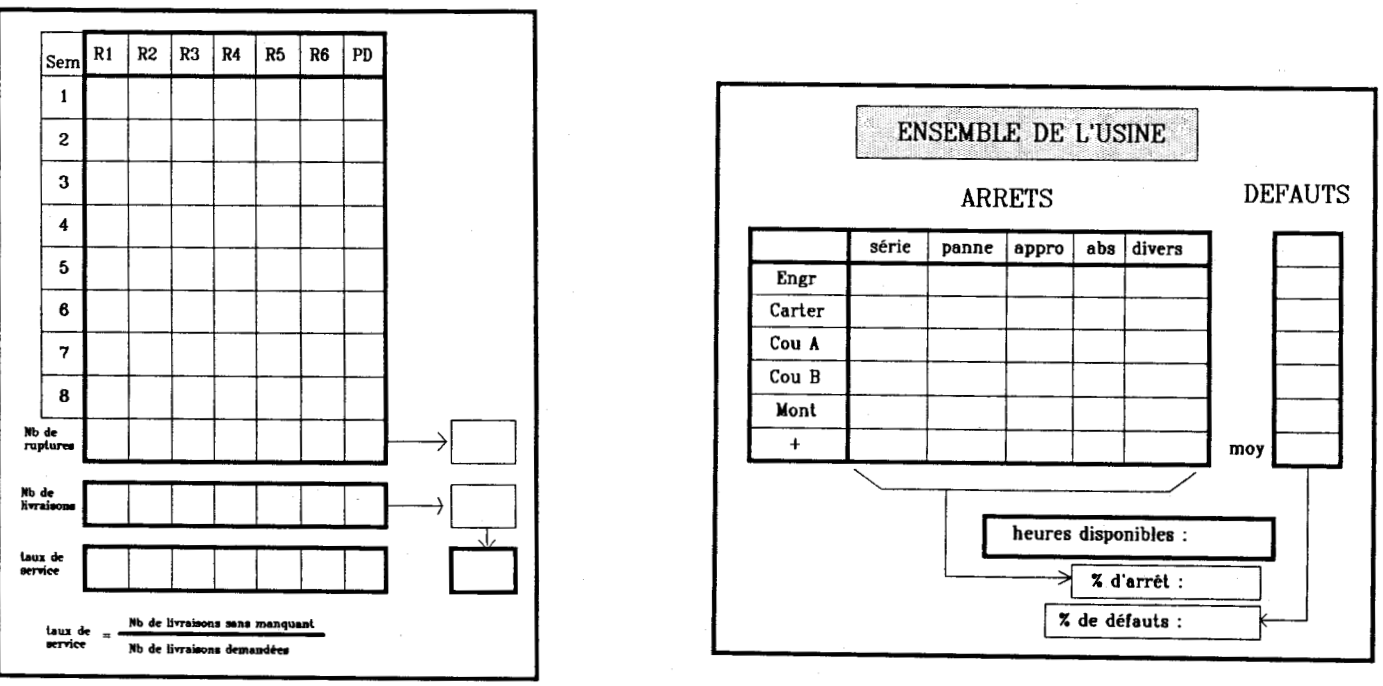
6) Il analyse les résultats

. Résultats commerciaux : une feuille de calcul du taux de service est à sa disposition (figure 10b) pour mesurer le taux de service, soit en cours de simulation, soit à la fin. Cette feuille est remise en début de jeu au responsable commercial.

. Résultats techniques : il distribue, en fin de partie, les feuilles d'analyse (figure 11a) de façon à ce que chaque poste récapitule les incidents. Sur la carte de synthèse (figure 11b), il regroupe ces résultats pour obtenir la moyenne de l'atelier.



Figures 10a et 10b. *Feuilles d'inventaire et d'analyse.*



Figures 11a et 11b. *Feuilles d'analyse et de synthèse.*

2.2 - Démarrage du jeu

Chaque participant reçoit un dossier contenant les informations préalablement exposées qu'il lit avant la séance ou en début de séance. L'animateur s'assure que les principaux points sont compris.

Il rappelle les buts de la simulation, et insiste sur le fait qu'elle doit déboucher sur une réflexion du groupe visant à améliorer le fonctionnement de l'atelier. Le jeu du KANBAN n'est pas un jeu de compétition, mais un exercice de groupe. Les participants peuvent d'ailleurs changer de poste au cours des différentes phases, sans que cela nuise au jeu.

La mise en place des joueurs se fait de la façon suivante :

1. Attribution à un joueur du rôle de responsable des ventes, et remise du dossier correspondant.

2. Création des équipes de poste (1 à 2 joueurs par poste).

3. Mise en place des postes autour de la table, en commençant par placer le poste de montage, puis les autres à partir de celui-ci.

4. Attribution des feuilles de postes, et commentaire sur leur utilisation (figure 12a).

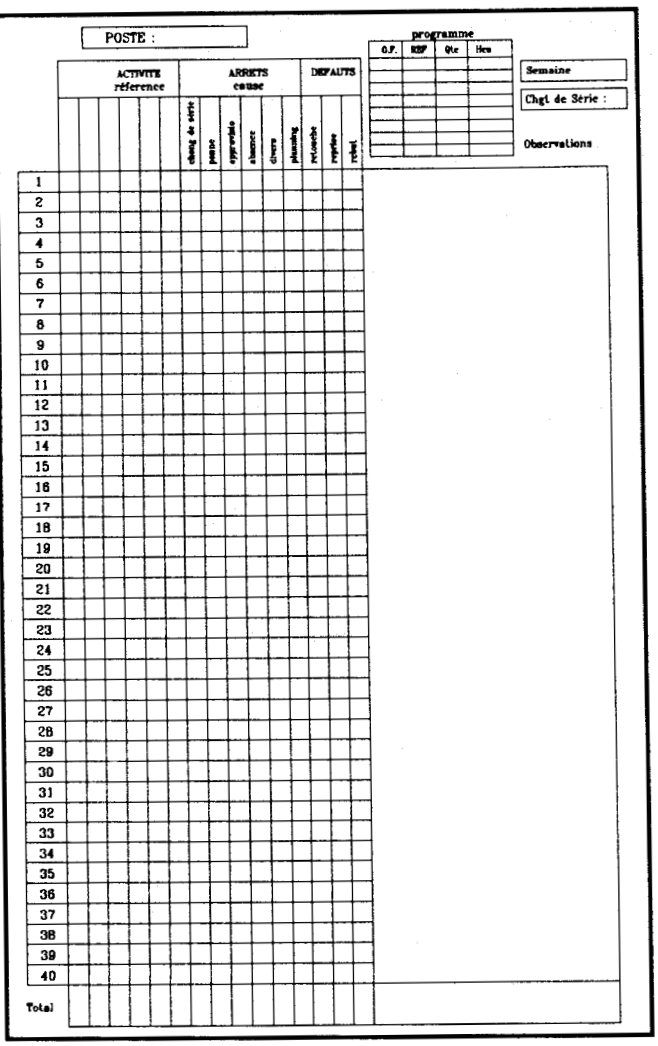
5. Distribution des jetons de matière première aux postes qui en consomment (premier usinage).

6. Mise en place des containers au milieu de la table.

L'animateur explique le rôle des cartes : CARTE INCIDENT DE FABRICATION, CARTE INFORMATION.

2.3 - Organisation du jeu

Le premier objectif demandé aux participants est l'établissement d'un plan général de production pour la semaine. A partir de celui-ci, un plan d'activité individuel par poste en sera tiré. Ce plan devra être respecté par les joueurs et cela quels que soient les aléas rencontrés dans le processus de fabrication (aléas introduits par les cartes incidents) lors de la première semaine de simulation.



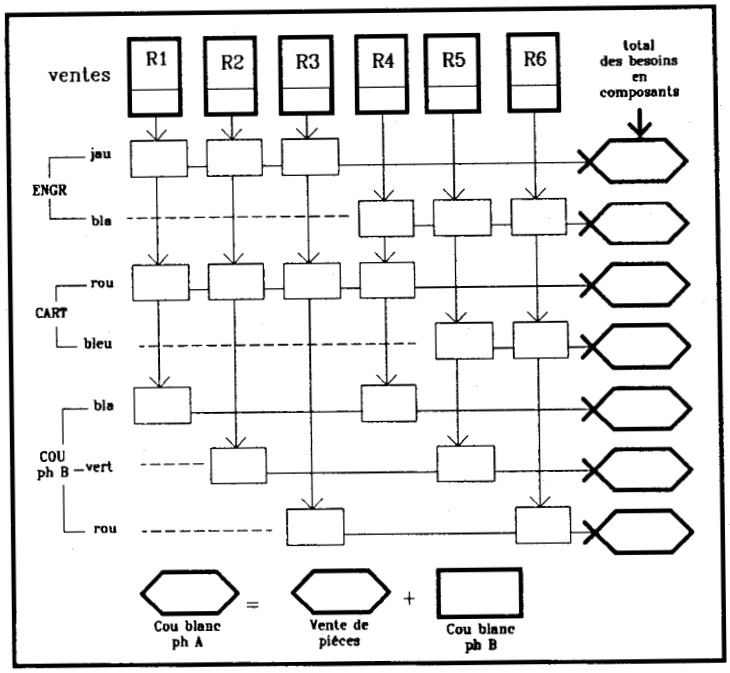


Figure 12a et 12b. *Feuilles de poste et de calcul.*

2.3.1 - Etablissement du plan de production

Pendant cette phase, l'animateur observe l'attitude des joueurs et son rôle consiste uniquement à communiquer, lorsque les participants en font la demande explicite, les informations qu'il a à sa disposition dans son dossier de direction.

Nous rappelons que ces informations sont les suivantes :

. statistique des ventes (figure 7a)

. écarts probables prévisions/ventes (figure 7b)

. histogramme des arrêts par poste (figure 8a)

. pourcentage moyen d'arrêts par poste (figure 8b)

. taux de défaut moyen des fabrications (figure 9).

Après avoir effectué la phase de démarrage présentée précédemment, l'animateur distribue un plan de production vierge et laisse les joueurs se débrouiller.

Cette phase dure en moyenne deux heures pendant lesquelles la négociation et la discussion sont intenses entre les joueurs. A partir des informations mises à leur disposition, ils vont essayer d'organiser la production en fonction des besoins de chacun. En effet, chaque décision de production d'un produit affecte l'ensemble des postes car les réducteurs sont un assemblage des pièces produites sur chacun de ces postes. Des feuilles identiques à la figure 12b leur ont été remises pour faciliter la comptabilisation des pièces à produire sur chaque poste. Après chaque décision, il est important que les joueurs s'assurent que la production programmée est potentiellement réalisable et cela en tenant compte du temps de changement de série fixé pour chaque poste.

Lorsque le groupe s'est mis d'accord sur le plan de production de la première semaine, les joueurs vont en extraire le plan d'activité pour leur poste. Celui-ci sera suivi lors de la première phase de simulation. A partir de cet instant, les joueurs sont prêts à passer à la phase suivante et la simulation proprement dite peut débuter.

2.3.2 - Phase de simulation (première semaine)

Pendant cette première phase de simulation les décisions de changements de série ne sont pas prises au fur et à mesure de la fabrication mais sont programmées en début de semaine sur le plan d'activité que les participants ont déduit de la phase précédente. Avant de commencer la simulation, la possibilité de constituer des stocks initiaux est offerte aux joueurs.

Lorsque les décisions sont prises, la simulation peut commencer et le rôle de l'animateur va consister, à chaque changement d'heure, à :

. jeter le dé une ou deux fois pour déterminer la conséquence de la carte d'incidents pour le poste concerné ;

. lire la carte d'information programmée à l'heure courante ;

. effectuer les ventes en prélevant les jetons dans les stocks correspondant aux réducteurs finis et aux pièces détachées à partir de son planning des ventes.

Le changement d'heure est laissé à l'appréciation de l'animateur qui va cependant s'assurer que les joueurs ont effectué la tâche correspondante à leur plan d'activité. Tout

au long du jeu, des explications, des discussions à partir des concepts proposés par les cartes incidents et d'informations pourront avoir lieu.

Il est à noter que le comportement des joueurs dans cette première phase de jeu est relativement passif car ils suivent leur plan et subissent les conséquences des cartes incidents. Aucune prise de décision n'est possible.

A la fin de la semaine, un inventaire des stocks et une analyse des résultats sont effectués. Le temps de déroulement de cette phase est très variable car il dépend des discussions et des questions qu'il peut y avoir dans le groupe sur les différents sujets. Sans discussion, une dizaine de minutes sont suffisantes pour conclure cette phase.

2.3.3 - Simulation des semaines suivantes

A partir de la troisième semaine, la méthode de planification mise en jeu est la méthode KANBAN (une description de cette méthode est donnée en annexe). Pour l'utilisation de cette méthode, un planning Kanban est attribué à chaque poste. Durant cette phase, les décisions de changement de séries sont maintenant prises à chaque heure (en respectant cependant les temps de changement de séries).

En début de semaine, les joueurs déterminent le nombre de Kanbans qu'ils jugent nécessaires au fonctionnement de leur poste. On peut brièvement rappeler qu'un Kanban est une étiquette prise du planning et mise sur le container au moment de la fabrication et que ce Kanban sera remis sur le planning du poste correspondant lorsque les pièces fabriquées seront utilisées par le poste en aval de la chaîne de production.

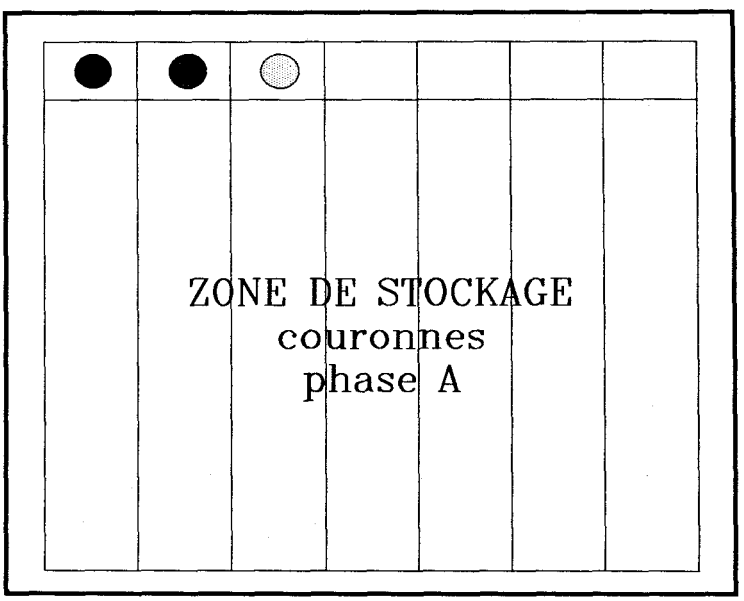


Figure n° 13. *Feuille de stocks.*

A l'emplacement des stocks est placé une feuille identique à la figure 13 et la figure 14 illustre le planning des postes sur lequel sont placés les Kanbans.

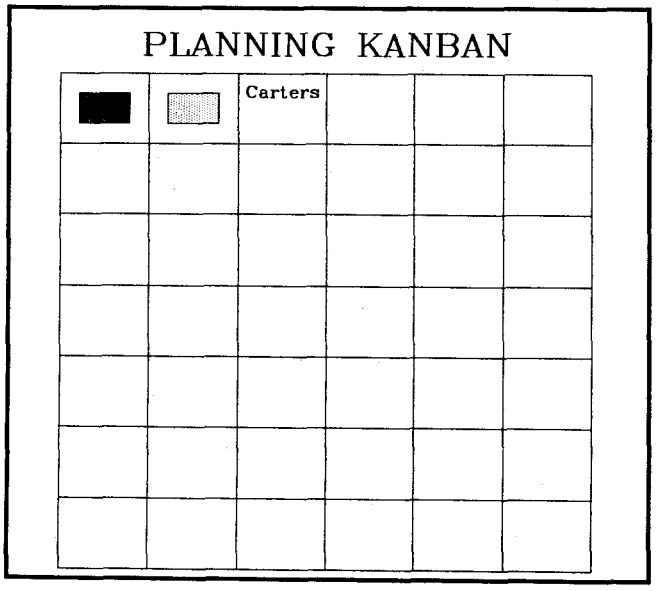


Figure n° 14. *Planning Kanban.*

Dans le jeu tel qu'il se déroule, les joueurs voient l'ensemble des stocks et le planning de chaque poste.

D'une manière identique à la phase précédente, le moment du changement d'heure est déterminé par l'animateur à partir d'observations générales sur l'ensemble du jeu. Cette phase est plus riche en discussions de la part du groupe car les joueurs ont la possibilité de réagir aux événements extérieurs simulés par les cartes "incidents".

METHODE KANBAN

Le mot KANBAN est tiré du Japonais et il peut être traduit par le mot "Carte". Dans tous les types de Kanban, nous retrouvons la carte qui est l'élément moteur essentiel du système.

1) Introduction

La méthode Kanban, généralement associée à la méthode du Juste à Temps permet de fonctionner à "zéro stock". La philosophie du Juste à Temps qui peut paraître très simple, à première vue, a pour objectif d'obtenir le produit voulu, au moment voulu et dans la quantité voulue. L'idée fondamentale est de voir le processus de fabrication comme un flux et de considérer les stocks comme des défauts de flux.

La métaphore, que l'on adopte souvent, est celle d'une corde que l'on tire entre des galets.

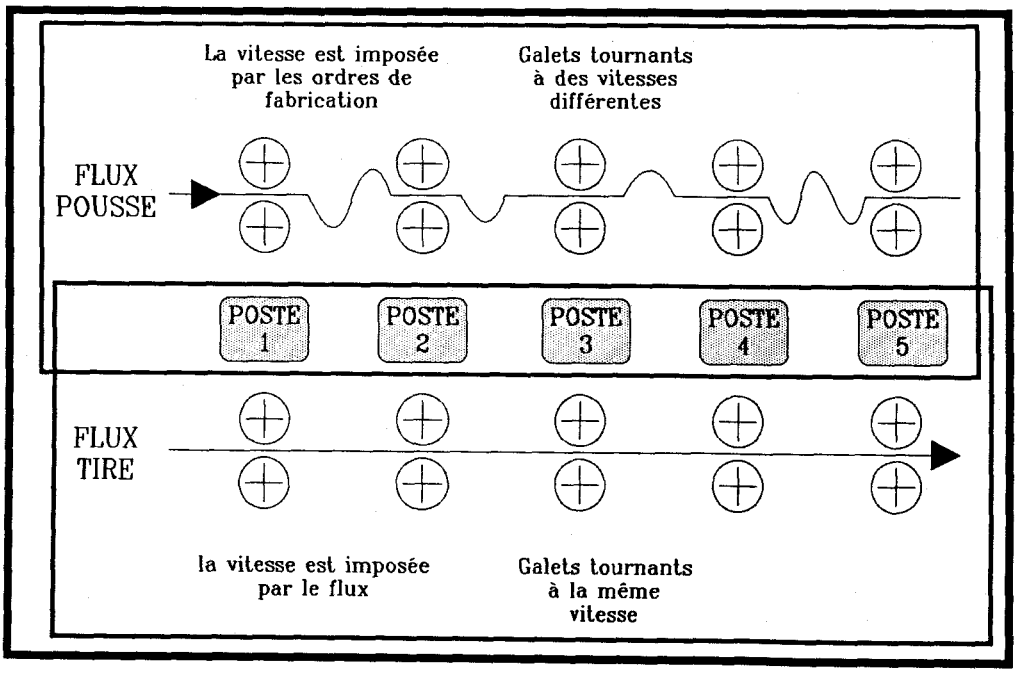


Figure 1. *Flux poussé et flux tiré.*

Il faut, en flux tiré, que le poste situé en amont d'un poste, soit en mesure de satisfaire les besoins de celui situé en aval de la chaîne.

Pour que la méthode soit efficace, il faut donc que l'information circule entre les postes et plus spécialement dans un sens aval-amont. Un moyen de faire remonter cette information est d'utiliser des kanbans. On trouve divers Kanbans :

. Le Kanban d'approvisionnement : d'un magasin vers un atelier.

. Le Kanban de fabrication : d'un atelier vers un autre atelier situé dans des bâtiments différents.

. Le Kanban de stockage : d'un atelier vers un magasin.

. Le Kanban d'expédition : d'un atelier vers le point d'expédition gare routière.

. Le Kanban interposte : D'une machine vers une autre située dans le même bâtiment.

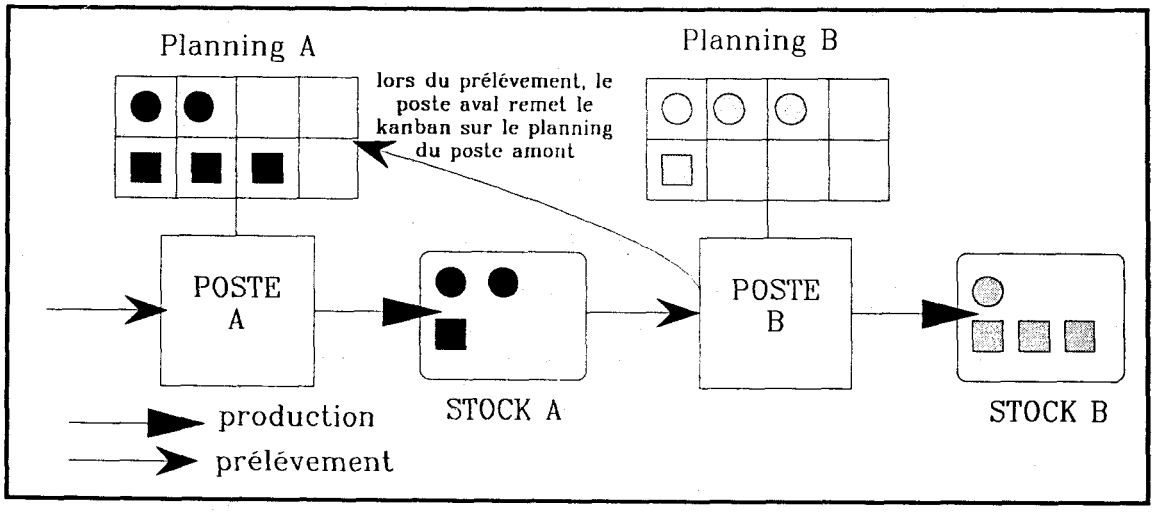


Figure 2. *Schématisation d'un système Kanban.*

Le nombre de Kanbans est fixe, ce qui permet de connaître aux divers points de la chaîne, le nombre de produits en stocks. Lorsqu'un ouvrier sur un poste i préleve le contenu d'un container, il place le Kanban dans une boite à Kanbans qui sera relevée à intervalles réguliers. L'ordre des kanbans dans la file indique l'ordre d'exécution des travaux au poste i. Les kanbans seront alors remis sur les plannings des postes avals correspondants.

2) Les stocks

Avec la méthode du juste à temps, l'objectif est de diminuer les stocks, voire de les réduire. En effet, les stocks existent bien souvent pour le confort et la sécurité.

Cependant, ils ont pour inconvénients de masquer les vrais problèmes et d'empêcher leurs résolutions. On a aussi tendance à faire des stocks pour les raisons suivantes :

. Stocks pour le travail par lots.

. Stocks pour l'utilisation des capacités.

. Stocks d'anticipation commerciale.

La principale raison, pour laquel on cherche à reduire les stocks, est liée aux différents coûts qu'ils engendrent :

. Valeurs immobilisées.

. Frais financiers liés aux emprunts.

. Surfaces industrielles immobilisées.

. Augmentation des manutentionnaires.

. Frais de gestion en moyen et en personnel.

. Vieillissement des pièces.

. Fin de série.

3) Les avantages du Juste à Temps

les avantages de cette méthode sont nombreux et sont récapitulés ci-dessous.

. Baisse des coûts financiers et autres coûts liés aux stocks.

. Baisse de la surface occupée.

. Baisse des besoins en magasin de stockage

. Détection plus rapide des pièces défectueuses

. Moins de pièces à contrôler ou à reprendre en cas de besoin.

. Circulation plus rapide entre les postes de travail de l'information relative au produit

. Amélioration de l'esprit de groupe et de solidarité

. Le stock ne masque plus les incidents, il devient difficile de les ignorer.

. Amélioration du service client

. Simplicité

. La responsabilité de la gestion est décentralisée sur le terrain

4) Les inconvénients du Juste à Temps

Bien que cette méthode possède de nombreux avantages, elle a également des inconvénients :

. Une réorganisation compléte des ateliers de fabrication (disposition en ligne).

. Une capacité machine surdimmensionnée.

. Une flexibilité humaine.

. Pas de gestion globale et prévisionnelle.

. Une nécessité d'utilisation d'une gestion Kanban chez les fournisseurs et les clients.

5) Conditions de réussite du Juste à Temps

Pour la mise en oeuvre de cette méthode certaines conditions sont nécessaires :

. Le produit est standard (sa définition précise est stable sur une période de temps suffisamment longue).

. L'atelier est constitué de façon à ce que la production puisse se faire suivant un flux (machine en ligne).

. La demande des pièces est suffisamment régulière.

. La qualité du produit est satisfaisante.

. Le nombre de panne des machines est réduit pour éviter une rupture de flux.

. Des moyens rapides de changement de séries sont mis en place (Méthode SMED).

Satisfaire cet ensemble de conditions revient à résoudre les problèmes suivants :

. Réduction des délais. (délais de transport, de sous-traitance, de sortie de pièces de magasin, de contrôle des pièces etc) car la sous tension du flux

est d'autant plus difficile qu'il y a des délais dans la circulation du flux.

. Evolution des produits. La mise en tension du flux sera d'autant plus efficace que les composants ou les sous-ensembles auront été standardisés. Il faut éviter des modifications techniques trop nombreuses au cours du temps ou les réaliser de façon groupée.

. Etablir de nouvelles relations avec les fournisseurs. Le flux ne se limite pas uniquement à l'entreprise et il faut donc que les fournisseurs : accroissent leur flexibilité, réduisent la taille des lots de livraison et respectent les délais.

. Avoir une demande commerciale lissée. Voir avec les clients s'il est possible d'échelonner les commandes dans le temps

. Résoudre des problèmes humains. Le principe du flux tendu est en contradiction avec les principes Tayloriens d'organisation du travail. Il n'est plus possible d'appliquer des primes de rendement individuel mais il faut adapter des primes collectives. La priorité est donnée à la qualité et à la régularité du travail.

RESUME

Depuis quelques années, une nouvelle discipline de recherche se développe : il s'agit du Travail Coopératif Supporté par Ordinateur (traduit en anglais par Computer Supported Cooperative Work - C.S.C.W.). Les systèmes C.S.C.W. englobent un large éventail d'outils qui diffèrent suivant la nature des interactions et la localisation des participants. Leurs champs d'application sont nombreux mais celui qui intéresse particulièrement notre laboratoire est relatif à l'enseignement. Le travail présenté concerne l'étude d'un système de Conférence Temps Réel permettant le travail coopératif d'un groupe composé de 5 à 6 personnes.

La première partie de ce mémoire sera consacrée à une présentation des différents types de systèmes existant pour en dégager les concepts fondamentaux (le W.Y.S.I.W.I.S., la gestion et le rôle des participants, les télépointeurs...). Nous détaillerons aussi les techniques adoptées pour concevoir les applications coopératives et les différents types d'architecture.

Dans la seconde partie, nous présenterons l'architecture logicielle et matérielle de notre prototype. Nous commencerons tout d'abord par exposer les extensions et modifications apportées à l'environnement objets retenu pour le prototypage (Smalltalk-80) : coopération entre objets, mécanismes de Proxy, répartition du modèle d'U.I.M.S. M.V.C. de Smalltalk, mise au point d'outils de débuging distants. Ensuite, les choix relatifs à l'implémentation des mécanismes et outils propres à la téléconférence seront expliqués et justifiés en prenant tout particulièrement en compte les problèmes de répartition.

La troisième partie sera consacrée à la réalisation d'une application du système dans le domaine de la simulation répartie d'une gestion de production destinée à l'enseignement de la productique.

La conclusion mettra en valeur les apports de notre architecture et posera les problèmes relatifs à son évolution : prise en compte du parallelisme d'activités, relâchement du W.Y.S.I.W.I.S. et l'introduction du multimédia.

Mots-clés : Téléconférence - Travail Coopératif - Interface Multi-utilisateurs - Activités coordonnées - Ecran Partagé - Télépointeur - Langage à objets - Système distribué.