Construction de collecticiels : étude d'architectures logicielles et de fonctions de contrôle

Rushed Kanawati

THÈSE

Présentée par

Rushed Kanawati

Pour obtenir
Le Titre de Docteur de l'INPG
(Arrêté ministriel du 30 mars 1992)

Spécialité : INFORMATIQUE

Construction de collecticiels : étude d'architectures logicielles et de fonctions de contrôle

Thèse soutenue le 24 Novembre 1997 devant le jury composé de :

MM. :	MAZARÉ	Guy	Président
	DERYCKE	Alain	Rapporteurs
	TRÉHEL	Michel	
	MOSSIÈRE	Jacques	Examinateurs
	RIVEILL	Michel	

Thèse préparée au sein du projet SIRAC (IMAG-INRIA)

À l'étoile ...
Au soleil et à la lune
aux planètes de Joseph
encore et pour toujours

Remerciments :

Le temps est venu pour remercier tous ceux qu'ont participé directement ou indirectement à l'aboutissement de cette thèse. Je remercie en particulier :

Messieurs Michel Tréhel et Alain Derycke pour avoir accepté d'être rapporteurs de cette thèse.

Monsieur Guy Mazaré qui m'a honnoré en acceptant de présider le jury d'évaluation de ce travail.

Messieurs Jacques Mossière et Michel Riveill qui ont dirigié ce travail

Monsieur Roland Balter qui m'a accueilli dans son équipe de recherche.

Les membres de l'équipe Sirac et les amis de l'Inria Rhône-Alpes.

Les membres du groupe du travail Scoop.

Mes amis grenoblois et syriens expatriés.

Mes parents et mes frères et sœures.

Un grand merci à part à Maria, ma femme qui a su m'encourager aux moments difficiles.

Résumé :

Nous nous intéressons dans ce travail à la problématique du développement des applications pour le travail coopératif, dites aussi collecticiels. Un collecticiel est à la fois une application multi-utilisateurs, répartie et interactive. La somme des trois propriétés précédentes rend le développement de ce type d'applications particulièrement difficile. Une approche souvent empruntée pour la construction des collecticiels consiste à développer une *plate-forme* qui fournit les services requis pour la coopération. Dans ce rapport nous identifions les principaux services demandés à une telle plate-forme et nous décrivons et nous comparons les différentes approches possibles pour réaliser ces services.

Nous proposons ensuite un nouvel environnement de coopération appelé *Colt* (pour Collaboration Terrain). Une première qualité de *Colt* est l'intégration des deux modes de travail : individuel et en groupe. Les utilisateurs partagent un espace d'information où chacun a le droit de *voir* et de *se mouvoir* selon des *rôles* qui lui sont attribués. Une deuxième qualité importante est l'adaptabilité fonctionnelle et structurelle. Selon l'axe structurel *Colt* permet de définir et de réajuster dynamiquement les rôles des utilisateurs, de définir autant d'activités coopératives que l'on souhaite et d'utiliser dans ces différentes activités les outils dont on a besoin. Sur l'axe fonctionnel, les utilisateurs peuvent définir et changer dynamiquement la configuration des activités.

Une attention particulière est faite en vue de doter l'environnement de stratégies variées pour contrôler les accès concurrents des utilisateurs aux données partagées. L'environnement *Colt* propose une famille de protocoles de tour de rôles et intègre un protocole original, appelé *LICRA*. L'algorithme *LICRA* est un algorithme *optimiste* fondé sur la détection de dépendances et la résolution *automatique* des conflits en utilisant un mécanisme de transformation d'opérations.

Un premier prototype de l'environnement *Colt* est aujourd'hui disponible pour une plate-forme *UNIX*. Le prototype est implanté en utilisant l'environnement de développement *TCL-TK*.

Abstract :

The focus of this dissertation is the study of groupware or CSCW-applications development. One major approach for groupware development consist of constructing a platform that provide the different services required for supporting collaboration. In this work we start by out-lighting the main requirements for a CSCW platform. Then we describe and compare the different possible approaches for implementing these requirements.

The results of the above mentioned study are used to feed the conception and the implementation of a new CSCW platform that we call : *Colt* (for Collaboration Terrain). Main features of the proposed platform are : 1) Easy transition between individual and collaborative work and 2) High functional and structural flexibility.

Colt users share a common information space. Each has his own role that defines his view on the shared space. The flexibility of the environment allows the users to define a wide variety of collaborative activities and to dynamically adjust and reconfigure existing ones.

A special attention is paid to provide the environment with various strategies for concurrency control. In this goal, an original protocol, called *LICRA* (for Lock-free Interactive Concurrency Resolution Algorithm) is integrated in the environment. *LICRA* implements an optimistic strategy based on the use of operation transformation mechanism.

A first prototype of *Colt*, developed in TCL-TK, is now available for a Unix-platform.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Le contexte général du travail

Nous nous intéressons dans ce travail à la construction d'un nouvel type d'applications informatiques dites *applications coopératives* ou *collecticiels*. Une application coopérative est un logiciel qui permet à un groupe d'utilisateurs de travailler ensemble et d'interagir entre eux dans le but de réaliser une tâche commune. La motivation pour la construction de telles applications est évidente: Dans la vie réelle des entreprises et des organisations humaines les individus passent une grande part de leurs temps en travaillant en groupes (réunions de travail, discussions, rédaction commune de plans et de rapports d'activités, sessions de diagnostic et de prise de décisions). Par suite l'idée de construire des logiciels qui supportent aussi bien le travail en groupe que les applications actuelles supportent le travail individuel est forcément séduisante. Cependant la construction de collecticiels s'avère une tâche intrinsèquement difficile. Les constructeurs doivent traiter de nouveaux problèmes qui s'ajoutent à la problématique classique de la construction de logiciels. Certains de ces problèmes sont *techniques*; d'autres appartient au domaine *sociologique*. Parmi les problèmes techniques nous citons les suivants :

- Le collecticiel est une application *multi-utilisateurs*. Il doit traiter plusieurs flux d'entrée à la fois. Des stratégies de synchronisation entre les contributions des différents utilisateurs et de gestion de la concurrence d'accès doivent être prévues [Greenberg et al.94].

- Le collecticiel est une application *répartie*. Les utilisateurs sont dans la plupart de cas géographiquement dispersés. Par conséquent le collecticiel doit prendre en considération l'aspect de communication à travers des réseaux informatiques; l'ordonnancement des messages échangés et l'absence d'un état global du système [Balter et al.91].

- Le collecticiel est une application *interactive*. Il est soumis à des contraintes temporelles sévères. De plus il doit vérifier de nombreuses propriétés ergonomiques [Salber95].

Sur le plan sociologique plusieurs problèmes sont à résoudre :

- Comment concilier la protection de la *vie privée* et la sécurité des utilisateurs d'une part et le besoin de rendre visible l'activité de chacun (pour favoriser la coopération) d'une autre part.
- Quel modèle de rendez-vous faut-il adopter ? et comment les utilisateurs peuvent se mettre à coopérer [Edwards95].
- Comment équilibrer l'effort fourni pour utiliser le collecticiel et le profit apporté par son utilisation [Markus et al.90].
- Comment concilier les deux modes de travail : individuel et coopératif ?

La coopération entre des informaticiens, des sociologues et des ethnologues semble être une nécessité pour la construction de collecticiels opérationnels [Grundin94]. La pluridisciplinarité est une caractéristique du Travail Coopératif Assisté par Ordinateur (*TCAO*). Elle est en même temps la principale source de difficulté de construction des collecticiels. Le poids considérable qu'ont les aspects sociaux renforce la nécessité des outils de prototypage rapide. Avec de tels outils les constructeurs peuvent rapidement évaluer l'utilisabilité des solutions techniques qu'ils proposent dans des conditions réelles de travail [Cosquer et al.94]. En conséquence la tendance actuelle des constructeurs des applications coopératives est orientée vers le développement des *plates-formes* qui serviront pour le développement et l'exécution des collecticiels variés. Cette approche, comparée à une approche monolithique, présente outre le prototypage rapide l'avantage de réduire le coût de développement de nouvelles applications et d'augmenter la fiabilité des applications due à la réutilisation de code. De nombreux projets de recherche se fixent l'objectif de construire une plate-forme pour supporter le travail coopératif. Nous citons pour l'exemple *GroupKit* [Greenberg et al.94], *XTV* [AW96], *Rendez-vous* [Hill92] et *COLA* [Trevor et al.93]. La présente thèse vise à contribuer à la définition d'une plate- forme générique pour le travail coopératif. Par générique nous voulons dire supporter de scénarios et de types variés de coopération. Nous résumons nos objectifs dans la section suivante.

1.2 Objectifs

Ce travail est a un objectif double :

1. Identifier les services systèmes nécessaires pour supporter la construction et l'exécution des collecticiels.

2. Étudier et comparer les différentes approches possibles pour la mise en œuvre des services identifiés. Le but est de fixer un ensemble de directives pour la construction d'une plate-forme générique pour le *TCAO*.

Concernant le premier objectif, notre position est la suivante. La coopération repose sur le partage de ressources par un groupe d'utilisateurs. La technologie actuelle de l'information présente des solutions intéressantes pour faciliter le partage d'informations entre des utilisateurs géographiquement dispersés (systèmes répartis, bases de données, le *WEB*). La construction d'applications coopératives repose sans doute sur l'exploitation de ces technologies. Cependant elle nécessitera de modifier les politiques d'exploitation de services actuellement disponibles (par exemple abandonner le principe de la transparence de la distribution [Greenberg et al.94]) et d'ajouter de nouveaux services (par exemple un service de gestion de rendez-vous [Edwards95]). Nous souhaitons fournir une *couche logicielle* qui regroupe les services requis pour supporter la coopération assistée par ordinateur. La construction de cette couche repose sur une technologie de gestion de la distribution et du partage d'informations non forcément *orientée coopération*.

Concernant le deuxième objectif, il est clair que nous ne pouvons pas aborder dans l'espace d'une seule thèse tous les problèmes posés par la mise en œuvre des services requis pour la construction des collecticiels. Nous avons choisi de concentrer notre action sur les deux problèmes suivants :

1. Comment supporter la création de nouvelles activités coopératives ? et quel modèle de rendez-vous faut-il fournir ?

2. Comment coordonner les contributions des utilisateurs impliqués dans une tâche commune surtout en cas de coopération temps réel où chaque utilisateur est sensé recevoir immédiatement toute contribution des autres utilisateurs.

1.3 Cadre du travail

Ce travail est mené au sein du projet *Sirac* (IMAG-INRIA) dont l'objectif principal est de concevoir et de réaliser un environnement pour le développement et l'exécution d'applications réparties [Balter et al.95]. Les recherches sont menées dans les deux thèmes suivants :

1. Construire des applications réparties en combinant des techniques de programmation à base d'objets et des techniques d'intégration de composants.

2. Développer un service pour le support d'objets partagés persistants répartis. L'objectif est de fournir un support adaptable et efficace utilisable pour la construction de plates-formes à objets répartis et de serveurs d'objets en utilisant une mémoire virtuelle partagée répartie.

Les applications coopératives sont choisies comme des applications pilotes pour explorer les nouveaux besoins systèmes et pour tester et valider les services fournis par le système *Sirac*.

1.4 Démarche suivie

Dans un premier temps, nous nous somme intéressés, à travers le développement d'une plate-forme appelée *CoopScan*[1] à la construction des applications coopératives temps réels [Balter et al.96b]. Dans *CoopScan* nous nous sommes fixés les objectifs suivants :

- Expérimenter l'approche de construction de collecticiels synchrones par réutilisation d'applications mono-utilisateurs existantes.
- Expérimenter l'approche de séparation des politiques de contrôle des applications employées.

Les résultats de l'expérimentation de *CoopScan* nous ont conforté dans notre choix de séparer les politiques de contrôle des applications utilisées. Ils montrent aussi que la transformation d'applications existantes en applications coopératives est une approche viable *à condition* que l'application à transformer vérifie certaines conditions de modularité et d'ouverture à son environnement d'exécution.

Dans une deuxième phase nous avons travaillé à étendre les fonctions offertes par *CoopScan* afin de supporter des formes plus génériques du travail coopératif. Nous avons abordé en particulier les problèmes suivants :

- Étudier le problème du passage facile entre le mode de travail individuel et le mode de travail en groupe.
- Fournir un modèle de spécification et de gestion des activités coopératives.
- Étudier le problème d'expression de droit d'accès et de sécurité des données employées dans des activités de groupe.

Les solutions proposées pour résoudre les problèmes cités ci-dessus sont regroupées dans une plate-forme appelée *Colt* que nous décrivons dans le (chapitre 4).

1.5 Contributions

Cette thèse contribue au domaine du *TCAO* selon les axes suivants :

1. Identification des besoins requis d'une plate-forme générique pour le TCAO (chapitre 2).

[1] Le développement de *CoopScan* est fait en étroite collaboration avec *Slim Ben Atallah* du projet *Sirac* et l'équipe de recherche en systèmes répartis au CNET-Paris (Issy les Moulineaux).

2. Étude et comparaison des différentes approches pour le développement de plates-formes pour le TCAO (chapitre 2).

3. Proposition d'un nouvel modèle de gestion d'activités coopératives (chapitre 3).

4. Proposition d'un nouvel protocole pour la gestion de la concurrence pour les collecticiels temps réel (chapitre 3).

1.6 Organisation de la suite du rapport

La suite du rapport est organisée en quatre chapitres qui sont les suivants :

Chapitre 2 : La TCAO, problématique et solutions Ce chapitre est divisé en trois parties. La première définit la classe d'applications que nous souhaitons supporter la construction. La deuxième analyse les besoins requis pour développer les applications choisies. Finalement, la troisième partie présente une comparaison fonctionnelle des travaux existants qui visent à fournir une plate-forme pour le *TCAO*. Le but est de dégager les choix de construction les plus adaptés pour réaliser les services identifiés.

Chapitre 3 : Colt, un environnement de coopération décrit notre proposition d'un environnement intégré pour supporter le travail coopératif. Dans un premier temps nous motivons notre choix pour la construction d'environnement intégré pour le travail coopératif et nous précisons les objectifs fixés pour un tel environnement. Ensuite nous décrivons l'architecture logique de *Colt* et la mise en œuvre de cet environnement. Finalement nous comparons notre proposition avec d'autres travaux similaires.

Chapitre 4 : Contrôle de la concurrence dans les collecticiels synchrones présente une étude des différentes solutions du problème de la concurrence d'accès et de gestion de droit de parole dans les collecticiels synchrones à architecture dupliquée. Un nouvel algorithme optimiste fondé sur le principe de la transformation d'opérations est proposé dans ce chapitre.

Chapitre 5 : Conclusion rappelle les principaux résultats et évoque des perspectives de ce travail.

Chapitre 2

Le TCAO, problématique et solutions

2.1 Introduction

Le Travail Coopératif Assisté par Ordinateur (*TCAO*[1]) est un récent domaine d'activités qui suscite l'intérêt de la recherche et de l'industrie. L'objet du *TCAO* est d'adapter la technologie de l'information aux besoins des utilisateurs impliqués dans des activités de groupe. Une définition du *TCAO* que nous devons à *Bannon* et *Schmidt* est la suivante [Bannon et al.91] :

> *Le TCAO est le domaine de recherche répondant aux questions suivantes : quelles sont les caractéristiques spécifiques du travail coopératif comme opposé au travail effectué par des individus isolés? Comment l'informatique peut-elle être appliquée pour soutenir les problèmes logistiques du travail coopératif? Comment les conceptions abordent-elles les délicats et complexes problèmes des systèmes qui façonnent les relations sociales?*

Le terme *collecticiel*, par analogie à logiciel et didacticiel, est choisi pour traduire le terme anglais *groupware*[2]. Une approche souvent empruntée pour la construction des collecticiels consiste à développer une plate-forme qui fournit les services requis pour la coopération. Cette approche, comparé à l'approche de construction monolithique, présente l'avantage de réduire le coût de développement et de faciliter le prototypage rapide de nouveaux collecticiels. De plus elle permet de fournir aux utilisateurs un environnement homogène (en terme d'interface-utilisateur) pour utiliser des collecticiels variées et par conséquence faciliter l'emploi des collecticiels.

[1] TCAO est la traduction en français du CSCW : Computer Supported Collaborative Work.

[2] La traduction officielle est synergiciel mais ce terme est peu employé dans les communications scientifiques.

Le présente chapitre a trois objectifs :

1. Préciser et définir la classe des applications que nous voulons construire (section 2.2).

2. Étudier et comparer les différentes approches possibles pour la construction des applications identifiées.

Pour délimiter notre étude de comparaison nous avons choisi d'évaluer les travaux existants en termes des fonctions et des services qu'ils proposent pour résoudre deux problèmes fondamentaux :

1. Comment un utilisateur peut faire partie d'une activité coopérative?

2. Comment coordonner les contributions des participants à une activité coopérative?

Notre démarche est la suivante. D'abord nous étudions dans la section (2.3) qu'est ce qu'une plate-forme *exemplaire* devrait fournir comme services pour résoudre les deux problèmes cités ci-dessus. Cette étude nous permet de fixer un ensemble de fonctions qui servent de base d'*évaluation fonctionnelle* des plates-formes existantes. Le nombre des plates-formes pour le *TCAO* étant très grand, il devient souhaitable, voire nécessaire, de comparer de familles ou de *classes*, de plates-formes. Dans cette optique nous proposons dans la section (2.4) une classification des approches de construction des plates-formes pour le *TCAO*. Ensuite nous présentons dans la section (2.5) des exemples représentatifs des approches identifiées et nous les comparons en fonction des critères retenus dans la section (2.3). Les principaux résultats tirés de cette étude sont résumés dans la section (2.6).

2.2 Définitions et classifications

À part le consensus qu'un *collecticiel est une application qui assiste un groupe d'individus impliqués dans une tâche commune*, peu d'autres points regroupent les différentes définitions du collecticiel. La divergence entre les définitions proposées dans la littérature est due d'une part à la grande diversité des applications qui assistent les groupes et d'autre part aux divers origines de la recherche en *TCAO* (systèmes répartis, interface homme-machine, automatisation du bureau).

Nous exposons dans la suite trois approches classiques pour la définition et la classification des collecticiels : par analyse de fonctions, par analyse de composants et par analyse de communication.

2.2.1 Définitions par analyse de fonctions

Cette approche repose sur l'identification des fonctions (ou des services) attendues d'un collecticiel. De la façon la plus élémentaire une définition par analyse de fonctions se fait en énumérant des exemples de collecticiels. Une telle démarche est empruntée dans [Karsenty94a] où l'auteur classe les collecticiels dans cinq catégories : 1) les messageries électroniques, 2) les éditeurs partagés, 3) les conférences assistées par ordinateur, 4) les systèmes d'aide à la décision et 5) les coordinateurs. Le bas niveau d'abstraction d'une telle définition limite son utilité ; étant fondée sur l'analyse des systèmes existants cette classification ne peut pas prendre en compte les nouveaux types de systèmes pouvant apparaître.

Ellis et *Wainer* proposent dans [Ellis et al.94b] une classification fonctionnelle selon laquelle ils distinguent quatre classes de collecticiels :

- Le dépositaire est une application qui permet à un groupe d'utilisateurs de manipuler un ensemble d'objets communs (ex. éditeur coopératif).
- Le "synchroniseur" est une application dont l'objectif est de coordonner les activités d'un groupe (ex. *flux de travail*[3]).
- Le communicateur est une application qui supporte les communications interpersonnelles (ex. le courrier électronique, applications de téléconférence).
- L'agent est une application qui fait appel à des méthodes de l'intelligence artificielle pour assister un groupe d'utilisateurs engagés dans un travail commun.

L'introduction de la classe *agent* met en cause l'orthogonalité de la classification proposée. La notion d'agent peut être employée dans des applications variées, y compris les trois premières classes de collecticiels identifiées par la classification introduite ci-dessus.

2.2.2 Définitions par analyse de composants

Une définition par analyse de composants consiste à identifier les principaux composants qui forment un collecticiel. Un premier exemple est le modèle proposé dans [Ellis et al.94b]. D'après ce modèle un collecticiel est défini par la combinaison de trois composants, dits aussi *modèles*, qui sont les suivants :

- Le modèle "ontologique"[4] décrit les classes des objets manipulés par le collecticiel ainsi que l'ensemble des opérations possibles sur ces objets.

[3] Traduction du terme anglais Workflow.

[4] Bien que le sens exacte du terme ontologique est relatif à l'être en tant que tel, son usage est étendu à tort au domaine des logiciels.

- Le modèle de coordination décrit les activités associées à chaque participant ainsi que les règles de coordination entre les différentes activités (par exemple précedences temporelles) nécessaires pour accomplir une tâche.

- Le modèle de l'interface utilisateur décrit les modèles de visualisation et des interactions avec les objets disponibles, les autres participants et le contexte du travail (représentation de la composition du groupe, les participants actuelles, les positions des autres dans l'espace partagé, etc).

Les deux modèles "ontologique" et d'interface utilisateur sont communs à tous les logiciels. Ils ne sont pas propres au domaine des collecticiels. Par suite, le trait caractéristique des collecticiels, d'après ce modèle, est le modèle de coordination. Or la fonction de coordination telle qu'elle est définie ci- dessus est une fonction de base dans toute application ou système multi-processus. C'est le cas par exemple des environnements intégrés pour le développement de logiciels. En fait, nous pouvons considérer une telle application comme un collecticiel *atypique* où un seul utilisateur humain (le programmeur) coopère avec des agents logiciels (éditeur, compilateur, debouggeur) pour accomplir une tâche donnée (le développement d'un logiciel). C'est sans doute le raisonnement qui a conduit *Ellis* et *Wainer* à ajouter la classe *agent* dans leur classification fonctionnelle [Ellis et al.94a] (voir 2.2.1). En raison de la spécificité évidente du travail en groupe, nous suggérons d'*exclure* du domaine du *TCAO* toute application faisant intervenir des groupes *atypiques* composés d'un seul individu. Ceci n'exclut pas la possibilité que l'ensemble d'acteurs dans un collecticiel soit composé d'un groupe hybride d'acteurs : acteurs humains et autres logiciels.

Un deuxième exemple de définition par analyse de composants est le modèle du *trèfle* proposé par le groupe français de travail *Scoop* du pôle de recherche centrée sur la communication homme machine [5]. Le modèle du trèfle décompose un collecticiel en trois *espaces* que nous citons les définitions telles qu'elles sont données dans [Salber95] :

- L'espace de production désigne les objets qui résultent d'une activité de groupe. Il décrit les concepts qui motivent l'action de groupe et qui dénotent l'œuvre commun mais aussi l'espace privé de chaque utilisateur comme dans un système mono-utilisateur.

- L'espace de coordination définit les acteurs notamment les individus, les groupes, les rôles, voire des agents logiciels intelligents, identifie les activités et les tâches (notamment leurs relations temporelles), désigne enfin les acteurs responsables des tâches et des activités. Tandis que l'espace de production offre une vue statique du système, l'espace de coordination en définit la dynamique.

[5] http://iihm.imag.fr/scoop/

- L'espace de communication fournit aux acteurs du système la possibilité d'échanger de l'information. Le contenu sémantique de cette information concerne les acteurs communicants. Il est étranger au système qui se contente de servir de messager.

Cette décomposition a le mérite de rendre explicite l'aspect de communication directe entre les utilisateurs ; l'essence de toute activité de groupe. Les trois espaces cités ci-dessus sont présents dans tout collecticiel même s'ils sont accentués différemment par les différents collecticiels (par exemple, une application de messagerie électronique met l'accent sur l'espace de communication tandis qu'une application d'édition coopérative met l'accent sur l'espace de la production). Par suite, le *trèfle* donne une définition générale d'une application de *TCAO* sans définir de classes de collecticiels.

2.2.3 Définition par analyse de la communication

La communication est à la base de toute activité de groupe. Une classification classique des collecticiels est fondée sur les caractéristiques du lien de communication employé. Deux critères fondamentaux caractérisent un lien de communication :

1. L'*espace* qui sépare les personnes communicantes. Ces dernières peuvent être au *même endroit* physique ou *géographiquement éloignées*.

2. Le *mode de communication* qui peut être *synchrone* ou *asynchrone*. Dans le premier cas toute action *pertinente* faite par un utilisateur est transmise immédiatement aux autres tandis que dans le deuxième cas la transmission des actions d'un utilisateur est contraint à respecter certaines règles (par exemple, avoir la confirmation explicite de l'utilisateur lui même).

Par conséquent, quatre classes de collecticiels peuvent être identifiées (voir figure 2.1). *Grundin* affine la classification précédente en introduisant la notion de l'*imprévisibilité* sur les deux axes. Un travail coopératif peut avoir lieu en même temps à des instants prévisibles ou non. De même, il peut se faire à des endroits distincts prévisibles ou non (cas d'emploi des postes de travail mobiles).

En pratique le même collecticiel doit satisfaire les quatre types de coopération illustrés à la figure (2.1). Pour justifier ceci prenons l'exemple de la rédaction commune d'un document. Une telle activité passe par au moins trois phases : 1) Discussion et élaboration du plan du document et attribution des rôles aux coauteurs, 2) Rédaction des différentes parties du document et 3) Correction du document. Les deux phases 1 et 3 nécessitent souvent des communications synchrones, tandis qu'il est opportun d'effectuer la phase 2 d'une manière asynchrone [Nastos92]. De même les rédacteurs doivent pouvoir continuer à coopérer qu'ils soient dans le même local physique ou géographiquement distribués.

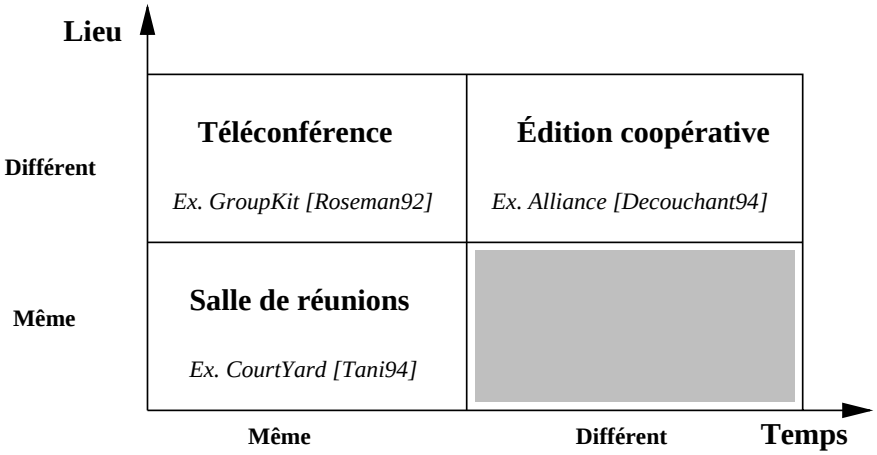


FIG. 2.1 - : Classification espace-temps des collecticiels.

2.2.4 Synthèse

Les différentes définitions présentées ci-dessus montrent que les collecticiels forment une sous-classe de la classe des applications multi-utilisateurs. À la différence des applications multi-utilisateurs *classiques* telles que les bases de données, les collecticiels ont la particularité de rendre chaque utilisateur *conscient* de l'existence des autres dans le système (propagation synchrone ou asynchrone des actions). Nous disons que les collecticiels sont des applications multi-utilisateurs *non-transparentes*.

Comme le montre le modèle du trèfle (section 2.2.2), trois principales fonctions caractérisent le fonctionnement d'un collecticiel :

- La fonction de communication dont la tâche est de permettre le transfert et l'échange *directe* de connaissances entre les individus coopérants.
- La fonction de partage est fondée sur le partage d'un espace d'information commun.
- La fonction de coordination définit les règles d'interaction entre les utilisateurs et entre les utilisateurs et l'espace d'information partagé.

Nous proposons dans la suite une nouvelle classification de collecticiels fondée sur la nature de l'entité contrôlé par le système. Trois types de collecticiels sont identifiés :

- **Les collecticiels à immersion** où le système contrôle les utilisateurs (ou la communication directe entre les utilisateurs). Ces collecticiels implantent des *extensions virtuelles* de l'espace physique. Un premier exemple est constitué par les applications de *médiaspaces permanents* [Dourish95b]. D'autres exemples sont les applications des *mondes virtuels* [Palanque et al.95] et les

applications de réalité augmentée pour les groupes. La mise en œuvre des collecticiels à immersion pose de nombreux problèmes au niveau de la protection de la vie privée et de l'expression de droits d'accès [Salber et al.95, Dourish95b].

- **Les collecticiels procéduraux** où le système contrôle la communication et l'acheminement des connaissances entre les utilisateurs. Dans ce type de collecticiels les interactions possibles entre les utilisateurs sont connues et fixées à l'avance. Les collecticiels procéduraux couvrent des domaines de travail dont les règles sont bien établies. La plupart des exemples de ce type d'applications sont développés dans le domaine de l'*automatisation du bureau* comme le système *OM-1* [Ishii et al.91] et le système *OTM* [Lochovsky et al.88].

- **Les collecticiels contractuels** dans lesquels les interactions entre les utilisateurs sont imprévisibles et où la participation d'un utilisateur à une activité coopérative est précédée par un *contrat* (description du but et des politiques de contrôle et d'interaction à employer). Le *contrat* peut être explicite ou implicite. Le lancement d'une activité coopérative par invitation est un exemple de contrat explicite tandis que le lancement d'une activité suite à un accès simultané à un document partagé est un exemple de contrat implicite [Edwards94]. Les applications d'édition coopérative, d'apprentissage assisté par ordinateur ou de téléconférence sont des exemples des collecticiels contractuels.

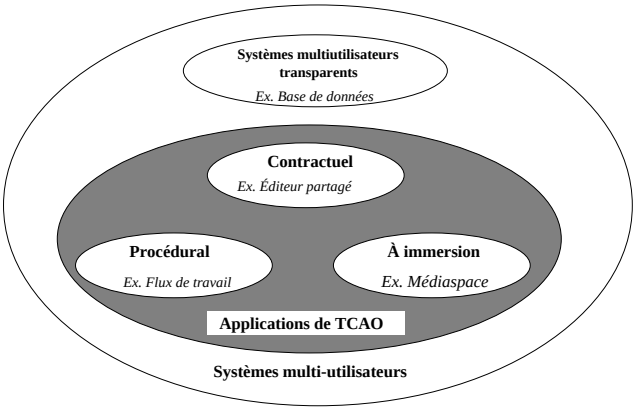


FIG. 2.2 - : Classes d'applications multi-utilisateurs.

Il est à noter que les différents types de collecticiels font appel aux mêmes mécanismes de base. Leurs différences résident dans la politique d'exploitation de ces mécanismes. Par exemple, une application de médiaspace et une application d'édition coopérative font usage d'un canal multimédias temps réel pour assurer la communication entre les utilisateurs. Tandis que le démarrage d'une communication multimédias est liée à l'édition d'un document dans l'éditeur coopératif,

celui-ci est fait automatiquement lorsque deux ou plusieurs utilisateurs se croisent dans le médiaspace. Les besoins en termes de contrôle ne sont évidement pas les mêmes dans les deux cas.

Nous nous intéressons dans la suite de ce travail aux collecticiels contractuels qui représentent pour nous le cas le plus général des applications coopératives. En fait, un collecticiel procédural est, par nature, dédié à une organisation ou à un type d'organisations précis (il implante des règles d'interaction figées à l'avance selon le modèle de l'organisation). De son côté, un collecticiel à immersion est avant tout une application qui favorisent les communications interpersonnelles non planifiées. De ce point de vue ce type de d'applications peut être considéré comme un cas particulier de la classe des collecticiels contractuels.

2.3 Critères d'études : fonctions de contrôle

Nous présentons dans ce chapitre une étude d'identification des servives qu'une plate-forme *exemplaire* pour le *TCAO* doit fournir pour résoudre deux problèmes fondamentaux qui sont :

1. La participation à une activité coopérative.
2. La coordination des contributions des utilisateurs au sein d'une activité.

Dans l'annexe B. nous présentons une étude plus générale pour l'identification des services requis pour le *TCAO*.

2.3.1 La participation à une activité coopérative

Nous décomposons la procédure de la participation d'un utilisateur à une activité de groupe en trois phases :

1. La phase de l'information pendant laquelle l'utilisateur prend conscience de l'*existence* de l'activité.
2. La phase de l'enregistrement dont le début est marqué par la manifestation de l'intérêt de l'utilisateur à rejoindre l'activité en question. Elle se termine par l'autorisation ou le refus de la participation de cet utilisateur.
3. La phase de l'intégration pendant laquelle l'utilisateur prend possession des ressources qui lui faut (données, droits, applications) pour pleinement participer au déroulement de l'activité.

Un exemple qui illustre cette décomposition est la procédure classique de la participation à une conférence scientifique. La phase de l'information correspond à la diffusion d'appel aux communications, la phase de l'enregistrement correspond à la soumission d'un article à la comité de programme de la conférence et la phase

de l'intégration correspond au déplacement vers le lieu de la conférence et la présentation du papier soumis. La dernière phase est subordonnée à l'acceptation par la comité de programme de l'article soumis.

La mise en œuvre des trois phases définies ci-dessus repose sur la réalisation des trois services que nous appelons respectivement, *service de rendez-vous*, *service d'enregistrement* et *service d'intégration*. Nous présentons dans la suite chacun des services cités ci-dessus.

2.3.1.1 Service de rendez-vous

Il s'agit de fournir un mécanisme qui permet aux utilisateurs de *se rencontrer* et de démarrer de nouvelles activités coopératives. Nous distinguons deux principales approches de gestion de rendez-vous ; l'approche explicite et l'approche implicite. Selon *l'approche explicite* un utilisateur se charge de désigner les participants potentiels à l'activité. La désignation des utilisateurs peut être directe ou indirecte. De même les utilisateurs désignés peuvent être notifiés par la création d'une activité d'une manière directe ou indirecte. La notification directe correspond à la réception automatique de l'événement (par exemple réception d'un courrier). Dans le deuxième cas il faut que l'utilisateur lui-même consulte un serveur spécifique pour trouver les activités qui lui concernent.

Selon l'*approche implicite* le système se charge de *détecter* les situations possibles de coopération. Par exemple le système peut proposer à deux utilisateurs qui emploient le même document (en même temps) de coopérer ensemble sur l'édition de ce document [Benford et al.95]. C'est le cas du système *Intermizzo* qui propose un service sophistiqué pour la gestion implicite de rendez-vous ; le système maintient une base de description des activités (individuelles ou coopératives) actuelles de chacun des utilisateurs du système. Les descripteurs des activités sont examinés systématiquement dans le but de détecter des éventuelles recouvrements entre les objectifs de deux activités isolées l'une de l'autre. Lorsqu'un tel recouvrement est détecté le système propose aux utilisateurs concernés de fusionner leurs activités en une seule activité coopérative. Pour aider le système à prendre les bonnes initiatives chaque utilisateur peut indiquer au système, sous forme de *règles d'interaction*, sa disponibilité à la coopération. Par exemple un doctorant qui rédige son mémoire de thèse peut spécifier la règle suivante : *Que personne ne me dérange sauf mon tuteur !!*. Le système intègre une base organisationnelle qui contient une description des relations formelles entre les utilisateurs. Cette base permet au système de reconnaître le tuteur de l'étudiant dans l'exemple précédant.

L'approche explicite est adaptée à la gestion des activités *formelles* dont les utilisateurs ont une certaine connaissance *a priori* (ex. réunion d'équipe, séminaire). En revanche, l'approche implicite favorise la coopération non planifiée ; elle *simule* les discussions qui peuvent être générées suit à des rencontres fortuites. Par exemple une *coopération*, ou disons une conversation utile, peut être engagée

entre deux personnes qui se rencontrent dans le même rayon d'une bibliothèque et qui s'aperçoivent que tous les deux s'intéressent au même sujet (ils cherchent le même ouvrage par exemple). Plusieurs études montrent que la coopération non planifiée est aussi importante pour la progression du travail dans une organisation que la coopération planifiée d'où l'importance de supporter les deux modèles de gestion de rendez-vous.

Les deux approches ont des avantages et des inconvénient inversés. En transférant la responsabilité de prise de décisions pour la création d'une activité de l'utilisateur (approche explicite) au système (approche implicite) nous dégageons l'utilisateur de la charge de gestion de rendez-vous (adresser les invitations, consulter un agenda, etc) et nous facilitons la transition entre le travail individuel et le travail en groupe. Mais en même temps nous rendons la mise en œuvre du service de rendez-vous plus complexe car il faut assurer au système un minimum d'*intelligence* pour que ses initiatives ne soient pas trop déplacées de point de vue des utilisateurs (interrompre systématiquement l'activité d'un utilisateur pour lui proposer de coopérer avec d'autres peut conduire l'utilisateur à ne plus utiliser le système). Le tableau (2.1) résume les caractéristiques des deux approches de gestion de rendez-vous.

	Approche explicite	**Approche implicite**
Mise en œuvre	Simple	Complexe
Nature de l'activité	Formelle	Informelle
Surcoût utilisateur	Oui	Non
Intégration du travail individuel	Non	Oui

TAB. 2.1 - : Les deux approches d'implantation d'un service de rendez-vous.

2.3.1.2 Service de l'enregistrement

Pour qu'un individu s'intègre dans un groupe il lui faut généralement remplir certaines conditions ou avoir l'autorisation d'un ou plusieurs membres du groupe cible. L'enregistrement dans ce cas est qualifié d'*enregistrement supervisé*. À l'opposé nous trouvons le mode d'*enregistrement libre* où aucune condition n'est posée sur la participation de nouveaux utilisateurs (cas d'une réunion publique).

Dans le cas de l'application d'une politique d'enregistrement supervisé l'approbation de la participation d'un nouvel utilisateur peut être faite par un autre utilisateur ou par le système lui même. Dans le premier cas nous parlons d'*enregistrement supervisé explicite* tandis que dans le deuxième cas nous parlons d'*enregistrement supervisé implicite*. Un exemple d'une politique explicite est l'emploi d'un système de vote. L'implantation d'une politique implicite repose sur la définition de

contraintes sur la participation à l'activité. Un exemple est de limiter le nombre de participants comme c'est le cas dans la plupart des jeux.

Le choix de la stratégie d'enregistrement dépend de la nature de la tâche à effectuer. Une plate-forme générique doit permettre l'emploi de deux stratégies définies ci-dessus.

2.3.1.3 Service de l'intégration

La fonction principale du service de l'intégration est de transmettre aux nouveaux arrivants le contexte actuel de l'activité (la description de la configuration de l'activité : les participants, les politiques de contrôle employées et les données manipulées dans l'activité). À la fin de l'intégration d'un utilisateur il faut que tous les participants aient la même vision du contexte de l'activité. Une propriété centrale de la fonction d'intégration est de supporter l'intégration d'un utilisateur pendant le déroulement de l'activité (problème de la participation dynamique) [Chung et al.94, Villemur95]. L'importance de cette propriété découle de la nature imprévisible du comportement des utilisateurs qui fait qu'une demande d'intégration peut survenir à n'importe quel moment.

2.3.2 La coordination

Le but d'un service de coordination est de fournir aux utilisateurs impliqués dans une activité coopérative des mécanismes qui leur permettent de *coordonner* leurs *efforts* afin d'atteindre les objectifs fixés. Nous identifions trois aspects principaux traités par un service de coordination :

1. La répartition des tâches consiste à définir les responsabilités de différents utilisateurs.

2. La présentation de la co-présence dont le but est de fournir à chaque participant des connaissances décrivant les interactions des autres utilisateurs avec l'espace partagé.

3. La résolution des conflits dont le but est d'éviter que l'application des opérations *contradictoires* sur un même objet par deux utilisateurs différents en même temps.

2.3.2.1 La répartition des tâches

Une approche classique pour faciliter la répartition et l'assignation des responsabilités dans le contexte d'une activité coopérative repose sur l'emploi du concept du *rôle fonctionnel*. Un rôle fonctionnel est une représentation d'un acteur (utilisateur) abstrait auquel on assigne un ensemble de *droits*. Les droits associés à un rôle fonctionnel définissent les actions que peuvent entreprendre

ce *rôle* dans l'activité. Des exemples classiques sont les rôles *rédacteur* et *lecteur* dans une activité d'édition.

Une classe particulière des rôles fonctionnels est constituée par les rôles de contrôle. Ces derniers portent sur la gestion et le contrôle du déroulement de l'activité. L'exemple type est le rôle du *président* ou du gestionnaire d'une activité. Le *président* est assigné la responsabilité de fixer les paramètres de la configuration de l'activité (le choix des politiques de contrôle à appliquer). Plusieurs travaux dans le domaine du *TCAO* proposent des modèles différents pour la définition et la gestion des rôles fonctionnels [Shen et al.92, Coulouris et al.94, Smith et al.94, Kanawati95]. Ces différentes études montrent que les propriétés suivantes sont requises d'un système de gestion de rôles :

- Expression de droits en termes d'opérations possibles sur les objets manipulés au lieu de simples opérations de lecture et l'écriture. Plus généralement l'expression de droits d'accès doit être faite à grain fin (ceci concerne les trois dimensions d'une règle d'accès à savoir l'acteur, l'objet et le droit).
- Permettre l'attribution et le changement dynamiques des rôles attribués aux utilisateurs.
- Permettre l'évolution dynamique des règles d'accès associés à un rôle.

2.3.2.2 La présentation de la co-présence

Dans le contexte d'une activité coopérative, avoir des connaissances sur les contributions, les attributions, les positions et les interactions des autres avec l'espace partagé est une condition nécessaire pour qu'un utilisateur puisse planifier ses propres actions et ses propres contributions. Fournir les renseignements cités ci-dessus est la fonction du service de la présentation de la co-présence. Curieusement, l'analyse des besoins et des approches de présentation de la co-présence est souvent occulté ou partiellement traité dans la littérature du *TCAO*. Peu de travaux ont traité ce sujet pourtant d'importance capitale. Pour cela nous étudions dans la suite ce service en plus de détail que nous le faisons pour les autres services.

Un service de présentation de la co-présence permet à chaque participant de répondre aux questions suivantes [Karsenty94b, Gutwin et al.96] :

- Qui sommes-nous? Autrement dit, un utilisateur doit être informé des noms et des attributions des participants actuels et potentiels (enregistrés et/ou invités) à l'activité.
- Où sommes-nous? Cette question comprend deux volets que nous représentons par les deux questions suivantes qu'un utilisateur peut poser :
  1. *Où suis-je par rapport aux autres*? Il s'agit de montrer les positions des autres utilisateurs dans l'espace partagé. Par exemple dans une

activité d'édition il faut montrer les sections et les paragraphes édités par chacun des participants. Un outil largement employé pour donner ce genre d'information est le bar de défilement multi-ascenseurs (un ascenseur par utilisateur).

2. *Où sont les autres*? Il s'agit de montrer comment les autres perçoivent l'espace partagé. Pouvoir voir ce que les autres voient permet de comprendre leurs commentaires. Une expérimentation d'un collecticiel synchrone d'aide à la décoration des pièces (en trois dimensions) a montré que les participants emploient fréquemment dans leurs communications orales des mots déictiques (par exemple effaces-moi *cet* objet, bouger *vers le droit*, approches *ici*..,etc) [Shu et al.94]. Avoir une fonction qui permet de voir ce qu'un autre voit permet de mieux comprendre ce qu'il veux dire. Une extension de la fonction précédente consiste à fournir une fonction *miroir* qui permet à un utilisateur de voir comment un autre utilisateur le perçoit.

- Que faisons-nous? Et que pouvons-nous faire? Le premier volet de cette question concerne la notification de chaque participant des actions faites par les autres dans l'espace partagé. La notification peut prendre deux formes :

  1. Propager dans la session les actions faites par un participant. Deux politiques de propagation d'actions peuvent être employées : la propagation synchrone et la propagation asynchrone. Les deux modes de propagation peuvent être employés simultanément ou alternativement dans une même session [Nastos92].
  2. Décrire la *nature* des actions faites par chaque participant sans rejouer les actions elles-mêmes. Cette approche convient pour les situations de coopération asynchrone. Une solution souvent empruntée pour mettre en œuvre cette approche consiste à employer des icônes spéciales comme c'est le cas de l'éditeur coopératif *Alliance* [Decouchant et al.96].

  La deuxième question concerne la visualisation des limites d'action de chaque utilisateur en fonction de l'état actuel de l'exploitation de l'espace partagé et en fonction des privilèges attribués à cet utilisateur. Un exemple est de montrer les zones de l'espace verrouillées par les autres utilisateurs.

- Comment sommes-nous arrivé ici? Il s'agit de présenter l'historique de l'évolution de l'œuvre commun. Avoir accès à l'historique de l'activité sert d'une part à fixer les responsabilités (qui a fait quoi?) et en conséquence mieux équilibrer le délicat rapport *effort fourni / gain individuel apporté* par l'emploi du collecticiel. D'autre part, l'historique permet à un utilisateur qui rejoint une session (qu'elle soit synchrone ou asynchrone) de mieux comprendre son évolution afin de mieux réfléchir ses propres contributions.

La mise en œuvre des mécanismes répondant aux diverses questions précédentes est sujet à plusieurs contraintes que nous regroupons sous la forme de trois *principes* :

1. Le principe du collecte passif selon lequel il faut éviter d'impliquer les utilisateurs dans le processus du collecte et de la distribution des informations décrivant la co-présence [Dourish et al.92].

2. Le principe de la non-distraction complèmente le premier principe dans le sens où la présentation de la co-présence ne doit pas *distraire* les utilisateurs de leurs tâches. Dans cette même optique *Dourish* et *Bellotti* suggèrent dans [Dourish et al.92] que la présentation de la co-présence doit prendre lieu dans le même espace visuel occupé par l'œuvre commun. Au contraire, d'autres travaux concluent que peu importe la zone d'affichage pourvue que l'interprétation des informations de la co-présence soit facile voire immédiate [Tatar et al.91, Gutwin et al.96]. Certains constructeurs des collecticiels ont opté vers l'emploi des effets sonores spéciaux pour décrire la co-présence. Le principe est d'associer des sons *significatifs* aux différentes opérations fournies par le collecticiel. Par exemple jouer le son d'ouverture d'une porte pour annoncer l'arrivée d'un nouvel participant et jouer le son de glisse pour décrire une opération de déplacement d'un objet. Nous pensons que l'emploi des sons doit être limité à la représentation des actes plutôt rares comme l'arrivée et le départ des utilisateurs ou l'ouverture d'un nouveau document. La transcription sonore de toute action peut générer un *vacarme* peu informatif.

3. Le principe de la non-intérference fonctionnelle selon lequel les outils de la présentation de la co-présence ne doivent pas affecter les fonctions propres du collecticiel. Par exemple plusieurs travaux emploient les couleurs pour identifier les participants et leur actions comme c'est le cas dans *GroupDesign* [Karsenty94a] , ceci implique que l'application elle-même ne peut plus manipuler des objets couloures sinon les utilisateurs seront confus.

Une difficulté majeur que rencontrent les constructeurs des collecticiels est l'absence de métrique pour évaluer la conformité de leurs réalisations aux principes cités ci-dessus. Peu de travaux ont abordé le problème complexe de l'évaluation de l'utilité des mécanismes de la co-présence. Ceci constitue à notre avis un axe important de recherche dans le domaine de construction de collecticiels.

2.3.2.3 La résolution de conflits

L'existence d'un service de présentation de la co-présence, aussi complet qu'il soit t-il, n'élimine pas le risque d'accès concurrents à un même objet de l'espace partagé. Des protocoles de gestion de la concurrence doivent être prévus. Dans

[Greenberg et al.94], les auteurs montrent qu'il n'existe pas une politique qui soit compatible avec tous les besoins des collecticiels. Une plate-forme doit fournir une variété de protocoles de gestion de la concurrence. Nous étudions ce problème en détails dans le chapitre (4).

2.3.3 Synthèse

Nous résumons dans cette section les services identifiés à l'issue de notre étude des deux problèmes de la participation et de la coordination dans les activités coopératives.

Services requis pour la participation

- Implanter les deux stratégies de gestion de rendez-vous : explicite et implicite.
- Permettre le passage souple entre les deux modes de travail individuel et en groupe.
- Implanter les deux stratégies d'enregistrement : libre et supervisé (implicite ou explicite)
- Permettre la participation dynamique de nouvel arrivants ainsi que les départs anticipés.

Services requis pour la coordination

- Fournir un service de définition et de gestion de rôles fonctionnels.
- Supporter les deux types d'interaction : synchrone et asynchrone.
- Implanter un service de présentation de la co-présence. Ce service doit couvrir les quatre aspects de la description de la co-présence (Qui? Où? Quoi? et Comment?)
- Implanter des stratégies variées de gestion de la concurrence.

À l'ensemble des fonctions citées ci-dessus s'ajoutent les deux fonctions générales suivantes (voir annexe B) :

- Permettre le changement dynamique de la configuration des activités.
- Permettre l'évolution future des fonctions offertes.

Nous nous servons des fonctions citées ci-dessus pour évaluer fonctionnellement les différentes plates-formes pour le *TCAO* proposées dans la littérature. Avant, nous présentons dans la section suivante une nouvelle classification des approches de construction de ces plates-formes.

2.4 Classification de plates-formes pour le TCAO

Nous proposons dans cette section une classification de plates-formes pour le TCAO fondée sur les trois critères suivants (voir figure 2.3):

1. *Le niveau du support de la coopération* qui peut être au niveau de l'interface homme-machine, au niveau des données manipulées ou au niveau du support de la communication.

2. *Le schéma de mise en œuvre* qui peut être centralisé, distribué ou hybride.

3. *La nature des applications supportées.* Une plate-forme peut supporter le développement de nouvelles applications ou la réutilisation d'applications existantes.

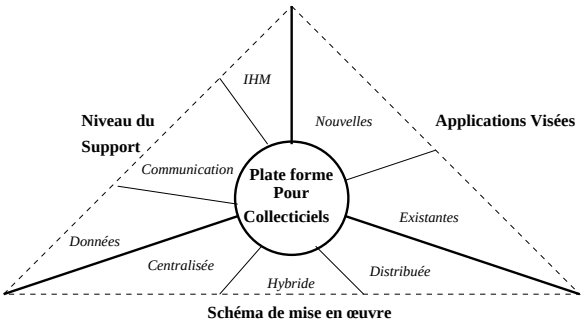


FIG. 2.3 - : Classification des plates-formes pour la construction de collecticiels.

2.4.1 Niveau du support

Niveau de l'interface homme-machine Il s'agit de concevoir un système de gestion d'interfaces-utilisateurs qui prend en compte les aspects du travail en groupe (présentation de la co-présence, coordination, communication et partage). Cette approche repose sur l'implantation d'une couche logicielle qui intercepte les interactions d'un utilisateur avec l'interface homme-machine d'une application et de reprdouir ces interactions sur les interfaces-utilisateurs des autres. Ainsi cette approche conçoit un collecticiel comme une *application partagée* entre plusieurs utilisateurs. Le partage d'une application sous entends que les utilisateurs sont tous présents ensemble en même temps dans l'espace de l'application. L'interaction entre les utilisateurs peut être synchrone ou asynchrone selon la stratégie de la diffusion des événements interceptés.

Plusieurs approches sont possibles pour construire une application partagée. Pour les comprendre nous commençons par rappeler la structuration d'une ap-

plication interactive mono-utilisateur. Une décomposition classique d'une application interactive sépare celle-ci en deux modules [Coutaz90] :

1. *Le module de présentation* gère l'interaction de l'utilisateur avec l'application.

2. *Le noyau fonctionnel* implante les fonctions propres de l'application. Il gère les données manipulés par l'application.

Le module de la présentation est décomposé à son tour en deux sous modules : *le pilote* et *le gestionnaire*. Le pilote se charge d'interagir avec les unités d'entrées/sorties de l'ordinateur (clavier, écran, souris, ..., etc). Le flux d'entrée est traité par le gestionnaire qui possède une description de l'interface-utilisateur de l'application qui lui permet d'interpréter les séquences d'événements bas niveau captés et de les synthétiser en commandes applicatives. Par exemple un clic souris dans une zone de l'écran où se trouve un bouton est interprété d'être une commande d'activation de ce bouton ; la connaissance employée par le gestionnaire pour interpréter cette commande est simplement la position du bouton et la surface qu'il occupe sur l'écran.

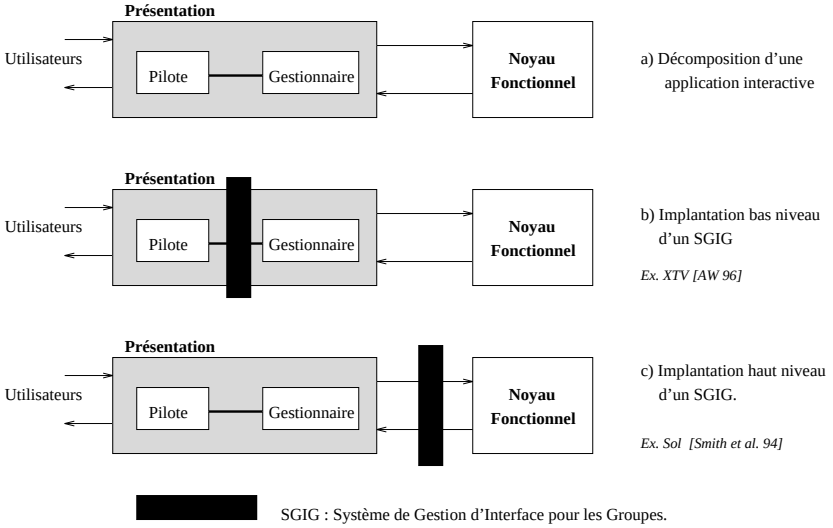


FIG. 2.4 - : Construction de collecticiels par partage d'applications.

La figure (2.4.a) illustre la structuration d'une application interactive. Les deux figures (2.4.b) et (2.4.c) illustrent les deux principales approches de construction d'applications partagées. La première approche dite bas niveau consiste à intercepter le flux d'entrée produit par chaque utilisateur et de le diffuser aux autres utilisateurs. La deuxième approche dite haut niveau consiste à diffuser les commandes produites par le gestionnaire.

Niveau de données Selon cette approche les fonctions de la coopération sont rattachées directement aux primitives de manipulation de données. Le principe est de construire des bases de stockage de données (ou de documents) partagées entre les utilisateurs du système. À chaque utilisateur est associée une vue sur la base partagée. Une vue définit les informations disponibles pour l'utilisateur et les fonctions que ce dernier peut appliquer sur ces informations. Lorsqu'un utilisateur modifie un objet qui appartient à la vue d'un autre utilisateur ce dernier sera averti. Les utilisateurs ne sont pas forcément co-présents en même temps dans l'espace partagé. Une première approche de mise en œuvre de la coopération par partage de données consiste à implanter un serveur partagé où les utilisateurs peuvent déposer et retirer de l'information. Un tel serveur est doté généralement des mécanismes de présentation de la co-présence des utilisateurs. Un exemple est l'application *BCSW* [Bentley et al.97]. Une deuxième approche consiste à faire propager les modifications apportées à un objet à tous les utilisateurs qui posédent cet objet comme c'est le cas dans *GroupDesign* [Karsenty94b] et *CoopScan* [Balter et al.96b].

Niveau de la communication Cette approche repose sur l'élaboration de mécanismes de contrôle du routage de l'information au sein d'un groupe d'utilisateurs. Il s'agit de fournir un service de *messagerie* évolué qui permet d'adresser un message à des groupes d'utilisateurs, spécifier la manière d'acheminement d'un message (ex. diffusion, envoi d'une manière séquentielle, ...), attacher des réactions à la réception d'un type de messages, etc. Cette approche implante un modèle d'interaction asynchrone. Des exemples qui illustrent cette approche sont : *Messie* [Sasse et al.96] et *ActiveMail* [Goldberg et al.92] qui sont tous les deux fondés sur l'extension du courrier électronique (e-mail).

2.4.2 Schéma de mise en œuvre

Tout collecticiel peut être décomposé en trois modules : la présentation, le noyau fonctionnel et *le module de contrôle* [Balter et al.96a]. Le rôle de ce dernier est de gérer la coopération entre les utilisateurs. Parmi les fonctions du module de contrôle nous retrouvons les fonctions de gestion de groupe, la gestion des accès concurrents aux données partagées et la gestion des rôles des utilisateurs.

Le schéma de mise en œuvre d'un collecticiel décrit le schéma de la répartition de différents modules identifiés ci-dessus entre les sites participants. Dans le cas général, un exemplaire de l'unité de présentation graphique est dupliqué sur chaque site. Par conséquent, le type de l'architecture d'un collecticiel dépend du schéma de la répartition des deux modules restant : le noyau fonctionnel et le module de contrôle. Chacun de ces deux modules peut être implanté d'une manière centralisée ou dupliquée. Ceci nous laisse le choix entre trois types d'architectures possibles : centralisée, dupliquée et hybride.

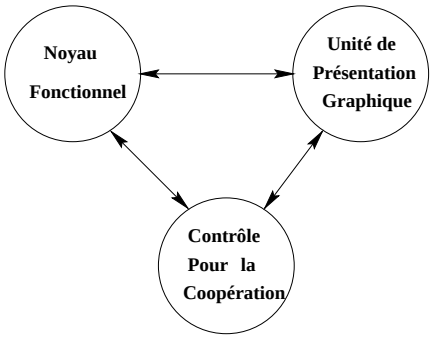


FIG. 2.5 - : Décomposition modulaire d'un collecticiel.

L'architecture centralisée selon laquelle une seule instance du noyau fonctionnel et une autre du module de contrôle sont exécutées sur un site central. Des exemples de plates-formes à architecture centralisée sont les plates-formes *XTV* [AW96] et *Rendez-vous* [Patterson et al.90].

Un inconvénient majeur de cette architecture est sa vulnérabilité (la défaillance ou l'isolation du site central entraîne l'arrêt du système). Un deuxième inconvénient est le trafic réseau important (en nombre et en volume de messages) qu'elle induit puisque toutes les commandes utilisateurs et leurs résultats (y compris l'affichage des résultats) sont véhiculés à travers le réseau. Des études de comparaison de cette architecture avec l'architecture dupliquée sont présentées dans [Balter et al.96a, Crowley et al.90].

L'architecture totalement dupliquée selon laquelle chaque site exécute une instance du noyau fonctionnel et une autre du module de contrôle. Des exemples sont *CoopScan* [Balter et al.96b], *MMconf* [Crowley et al.90] et *GroupKit* [Roseman et al.92]. Cette architecture résout les problèmes de la centralisation. Cependant elle nécessite d'élaborer des politiques de contrôle de la concurrence et de gestion de la cohérence de données plus complexes que dans le cas de l'architecture centralisée (voir chapitre 4).

L'architecture hybride repose sur la combinaison des deux architectures définies ci-dessus. Par exemple une architecture hybride peut consister à avoir une seule instance du noyau fonctionnel et des modules de contrôle dupliqués. C'est le cas par exemple de l'architecture *MEAD* [Bentely et al.92b]. Une autre réalisation d'une architecture hybride consiste à répartir les ressources d'un collecticiel sur les sites participants. Un exemple est le modèle d'architecture à *serveurs dupliqués* proposé dans [Cortes et al.95].

2.4.3 Nature des applications supportées

Une plate-forme peut supporter la construction de nouvelles applications comme c'est le cas de *GroupKit* [Roseman et al.92] et *Sol* [Smith et al.94] ou la transformation d'applications existantes en applications coopératives comme c'est le cas de *XTV* [AW96] et *CoopScan* [Balter et al.96a]. La transformation d'applications existantes peut être faite avec ou sans modification de code source. Dans le premier cas (transformation avec modification de code) nous parlons de construction par *adaptation* [Knister et al.93] tandis que dans le deuxième cas nous parlons d'*intégration* d'applications existantes. En ce qui concerne l'intégration, deux grandes approches sont possibles 1) *l'intégration au niveau du système de fenêtres* et 2) *l'intégration au niveau de l'application* [Benatallah et al.94].

Intégration au niveau de système de fenêtres. Pour illustrer cette approche prenons l'exemple concret des applications qui s'exécutent sous le système de fenêtres *X-Window*. *X-Window* implante un modèle client-serveur ; une application *X*, dite aussi *client X*, délègue la gestion de son interface utilisateur à un serveur d'affichage : le *serveur X*. Le client et le serveur communiquent entre eux par le biais d'un canal d'affichage.

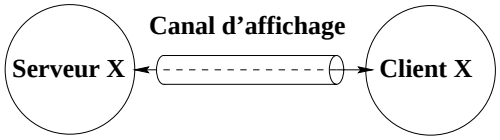


FIG. 2.6 - : Le modèle client-serveur du système de fenêtres X-Window.

La communication à travers le canal d'affichage est régie par le *protocole de communication X*. Celui-ci décrit le format logique (les structures de données) et la signification des messages échangés dans les deux sens sur le canal d'affichage. Deux types de messages sont définis par le protocole *X* : les *requêtes* et les *événements*, émis respectivement par le client et par le serveur.

Le principe de transformation d'une application *X* en collecticiel consiste à intercaler un *pseudo-serveur* entre le client (l'application à transformer) et le serveur*X*. Le *pseudo-serveur* se comporte comme un *client X* vis à vis des *serveurs X* et en tant que *serveur* vis à vis des *clients*. Ainsi il joue le rôle de *démultiplexeur* pour transmettre les requêtes d'affichage aux différents serveurs et pour transmettre les événements d'entrée au(x) client(s).

Ce schéma d'intégration est valide pour toute application X. Une fois le *pseudo-serveur* est mis en place les utilisateurs peuvent partager toutes les applications X qui leurs sont disponibles. Toute la difficulté réside donc dans la construction et la mise en œuvre du *pseudo-serveur*. Une étude détaillée de l'implantation d'un *pseudo-serveur* est présentée dans [AW96]. La mise en œuvre

d'une plate-forme d'extension d'un système de fenêtres peut être faite d'une manière centralisée ou dupliquée. Des exemples d'implantation centralisée sont la plate-forme *XTV* [AW96] et *Shared X* [Gust88]. *MMconf* est un exemple d'une implantation dupliquée [Crowley et al.90].

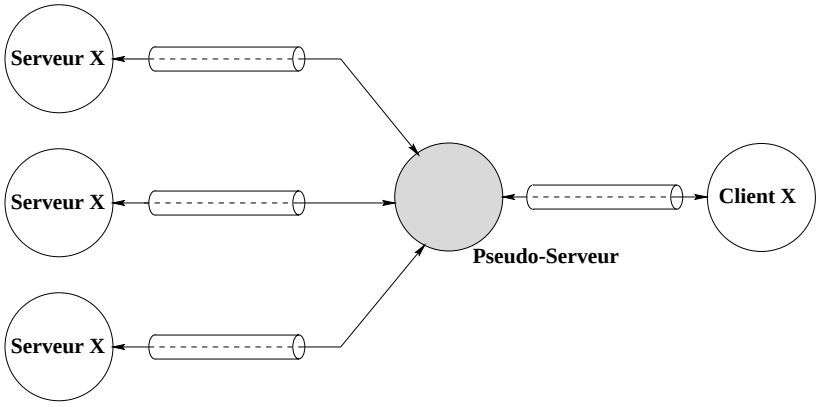


FIG. 2.7 - : L'architecture de la plate-forme XTV

Intégration au niveau de l'application. Cette approche est restreinte aux applications dites *ouvertes*. Une application est ouverte si elle vérifie les deux conditions suivantes :

- Fournir à son environnement d'exécution une interface de programmation (*API*) qui permet à des modules logiciels extérieurs à l'application d'accéder aux ressources de celle-ci et de reproduire les mêmes actions qu'on peut effectuer à travers son interface-utilisateur.

- Fournir à son environnement d'exécution un *système de notification* (Callback). Ce système donne la possibilité à des modules logiciels extérieurs d'être avertis des tentatives d'actions faites à travers l'interface-utilisateur.

Un exemple d'une application ouverte est l'éditeur *Thot* [Quint et al.94] et l'application *GraphWidget* [BL91]. Pour illustrer l'approche de l'intégration au niveau de l'application nous décrivons dans la suite notre expérience de la transformation de *Thot* en éditeur coopératif synchrone [Balter et al.96b]. Le système de notification fourni par *Thot*, nommé *ECF* pour External Call Facility, génère deux types d'événements :

1. Les événements de type PRE qui précèdent l'exécution effective d'une action commandée via l'interface utilisateur. Un événement PRE spécifie la nature de la commande demandée (ex. modification, suppression, insertion) mais pas les paramètres de la commande (qui ne sont pas encore connus).

2. Les événements de type POST produits après l'exécution d'une commande. Un événement POST décrit tous les paramètres de l'action accomplie : nature, élément cible et modification apportée.

Un langage spécifique, appelé le langage I, permet d'associer des réactions aux événements produits. Les réactions sont des fonctions C. Dans le cas des événements PRE les réactions sont obligatoirement de type booléen dont la valeur est *True* si la réaction remplace le traitement standard de l'éditeur. La transformation du *Thot* en éditeur coopératif synchrone est faite de la façon suivante. Chaque site participant à une session d'édition exécute une copie de l'éditeur et une copie de module de contrôle ; l'architecture est donc complètement dupliquée. Lorsqu'un utilisateur applique une commande via l'interface utilisateur, un événement PRE décrivant la commande est intercepté par le module de contrôle. Ce dernier permet l'exécution locale de l'opération si celle-ci est une opération à effet exclusivement local (comme par exemple le sauvegarde du document) ou bien si le site local détient le droit de parole. Dans le cas contraire le module de contrôle empêche l'exécution de l'opération, en retournant à l'éditeur la valeur booléen *True*. L'algorithme suivant montre la réaction du module de contrôle aux événements PRE.

Algorithme 2.1 *Réaction aux événements* PRE*;*

Soit Op l'opération décrite par l'événement intercepté;
SI *(Op est locale)* OU *(je suis le détenteur de droit de parole)* ALORS
    Exécuter localement Op; // Un événement POST *va être généré*
SINON
    Empêcher l'exécution de Op;

L'exécution d'une opération est suivie par la génération d'un événement POST. Le module de contrôle réagit à ce type d'événements selon l'algorithme suivant :

Algorithme 2.2 *Réaction aux événements* POST*;*

Soit Op l'opération décrite par l'événement capté;
SI *Op est locale* ALORS
    Exit;
SINON
    Diffuser l'opération à tous les autres sites;

Adaptation d'applications. Une approche générale pour *adapter* des applications existantes en vue de les transformer en collecticiels consiste à procéder en deux étapes :

1. Fournir une bibliothèque de fonctions de gestion de la coopération dont le code est indépendant des applications cibles.

2. Développer pour chaque application à adapter un module spécifique qui fait la liaison entre les fonctions de l'application et la bibliothèque de gestion de la coopération.

Cette approche suppose que le code source de l'application est disponible. La difficulté principale réside dans l'analyse du code des applications existantes afin de localiser les points où il faut insérer les appels aux primitives de la bibliothèque. Des exemples de cette approche sont la plate-forme *CoEx* [Patel et al.93] et la plate-forme *ConferenceToolKit* [Bonfiglio et al.91].

Des applications différentes requièrent des *efforts* différents de transformation. L'expérience de *CoEx* montre que les applications construites d'une manière modulaire avec une séparation nette entre le noyau fonctionnel et le module de l'interface-utilisateur sont les applications les plus faciles à transformer. Le tableau (2.2) montre que l'adaptation des applications est plutôt une opération non-coûteuse. Les trois chiffres représentent respectivement le nombre de lignes de code initial, modifiés et ajoutés.

Application	Code initial	Code modifié	Code ajouté
`Éditeur de texte (sted)`	500	100	300
`Éditeur de dessin (idraw)`	19,000	700	800
`Tableau de calcul`	119,000	1,000	2,000

TAB. 2.2 - : Modification et ajout de code nécessaires pour l'adaptation d'applications dans CoEx.

La construction de collecticiels par *adaptation* présente l'avantage d'être plus souple pour la mise en œuvre de protocoles de contrôle. L'inconvénient majeur est qu'elle nécessite d'avoir le code source de l'application. L'approche d'intégration d'applications, au niveau de l'application ou au niveau de système de fenêtres, n'a pas cette limitation. La différence principale entre les deux approches d'intégration d'applications réside dans le niveau sémantique des événements manipulés. Dans le cas de l'intégration au niveau de système de fenêtres les événements sont des événements de bas niveau sémantique, tandis que dans l'intégration au niveau de l'application les événements sont d'un niveau sémantique élevé : insertion d'une texte, application d'une commande.

L'approche d'intégration au niveau du système de fenêtres présente l'avantage d'être indépendant des applications à intégrer. La construction du module de contrôle, dans ce cas, dépend exclusivement du système de fenêtres en question. Les objets manipulés selon cette approche sont de bas niveau : écran, fenêtres et pixels. Ceci implique le partage de tout autre objet d'un niveau sémantique plus haut comme par exemple un curseur de défilement. Dans [Balter et al.96a] une comparaison entre les deux approches en termes de nombre de messages véhiculés

et en termes de volume de messages échangés montre que l'intégration au niveau du système de fenêtres consomme plus de ressources (nombre de messages et bande passante du réseau) que son homologue au niveau de l'application. Le tableau (2.3) résume les caractéristiques de chacune des approches de réutilisation d'applications.

Critère	Int.-SF.	Int.-App.	Adaptation
Mise en œuvre	Indépendant de l'app.	Dépend de l'app.	Dépend de l'app.
Inter-opérabilité	Non	Oui	Oui
Trafic réseau	Élevé	Moins important	Moins important
Code Source	Non nécessaire	Nécessaire	Nécessaire
Application Ouvert	Non nécessaire	Nécessaire	Non nécessaire

TAB. 2.3 - : Principales caractéristiques des approches de transformation d'applications existantes en collecticiels.

2.5 Exemples

Nous présentons dans la suite quelques plates-formes représentatives des approches de construction discutées ci-dessus. Noue mettons l'accent dans cette présentation sur la satisfaction des plates-formes retenues des services identifiés dans la section (2.3.3). Les plates-formes choisies sont énumérées dans le tableau suivant.

Plate-forme	Support	Architecture	Applications
XTV	IHM	Centralisée	Intégration- SF
Sol	IHM	Hybride	Nouvelles
GroupKit	IHM-Données	Dupliquéé	Nouvelles
CoopScan	Données	Dupliquée	Intégration-App.
Mead	Données	Hybride	Nouvelles
GroupDesign	Données	Dupliquée	Intégration-App.
ActiveMail	Communication	Centralisée	Nouvelles

TAB. 2.4 - : Inventaire des plates-formes choisies pour l'étude de comparaison des approches de construction de plates-formes pour le TCAO.

Nous analysons et justifions les limitations fonctionnelles présentées par les différentes plates-formes étudiées ici dans la section (2.6).

2.5.1 XTV

La plate-forme *XTV* supporte exclusivement la construction de collecticiels par intégration d'applications existantes au niveau du système de fenêtres *X-Window*[AW96].Les collecticiels construits implantent tous un modèle de coopération synchrone. L'architecture de mise en œuvre est totalement centralisée (voir Figure 2.7).

Le lancement d'une nouvelle session correspond à l'exécution d'un serveur *XTV* sur un site central. Une session est identifiée par une adresse constituée par le nom de la machine hôte et le numéro de la session (donné par l'utilisateur). Pour participer à une session un utilisateur lance un client *XTV* qui se connecte au serveur de la session. Pour établir la connexion, le client doit savoir l'identificateur de la session à rejoindre. La diffusion de l'adresse d'une session est à la charge de l'utilisateur qui la crée. Celui-ci peut notifier les participants potentiels, directement par l'envoi d'un courrier ou par téléphone, ou indirectement en enregistrant l'adresse dans un répertoire, ou un annuaire de sessions. Par suite *XTV* offre seulement un système explicite de gestion de rendez-vous.

L'enregistrement est libre. Cependant un utilisateur privilégié, dit le *président*, a le droit de forcer la déconnexion de n'importe quel participant. Le rôle président est automatiquement attribué à l'utilisateur qui démarre la session. Celui-ci garde ce rôle le long de la session.

La participation tardive des utilisateurs est tolérée grâce à la mise en œuvre d'un protocole de participation dynamique [Chung et al.93]. Une politique de tour de rôle est implantée pour résoudre le problème des accès concurrents. À chaque application intégrée est associé un jeton. Le passage du jeton entre les participants est fait selon la règle du premier demandeur premier servi. La figure (2.8) illustre l'interface de contrôle de *XTV*. Cette interface affiche d'une manière permanente les informations suivantes : le nom du président de la session, la liste des participants et la liste des applications employées.

Pour savoir qui a le droit de parole sur une application donnée, l'utilisateur doit d'abord sélectionner l'application dans la liste des outils ensuite appuyer sur le bouton `Floor-Info`. Le nom du détenteur du jeton est alors affiché dans la zone de messages système (en bas de la fenêtre de contrôle). Pour avoir plus de renseignements sur un participant il suffit de sélectionner son nom dans la liste des participants et appuyer sur le bouton `Participant-Info`. Ceci a pour effet de montrer le nom et l'adresse de l'utilisateur ainsi que l'ensemble des applications qu'il utilise. Une idée intéressante proposée dans *XTV* consiste à fournir des outils d'aide à la discussion. Dans cette optique, deux applications sont constamment à la disposition des participants à une session *XTV*: une application de bavardage (*chatting*) et une autre implantant un tableau blanc partagé.

Une autre idée qui vise à améliorer la perception de l'espace partagé (constitué par l'ensemble des applications partagées) consiste à permettre de visualiser toutes les applications partagées dans la même fenêtre virtuelle.

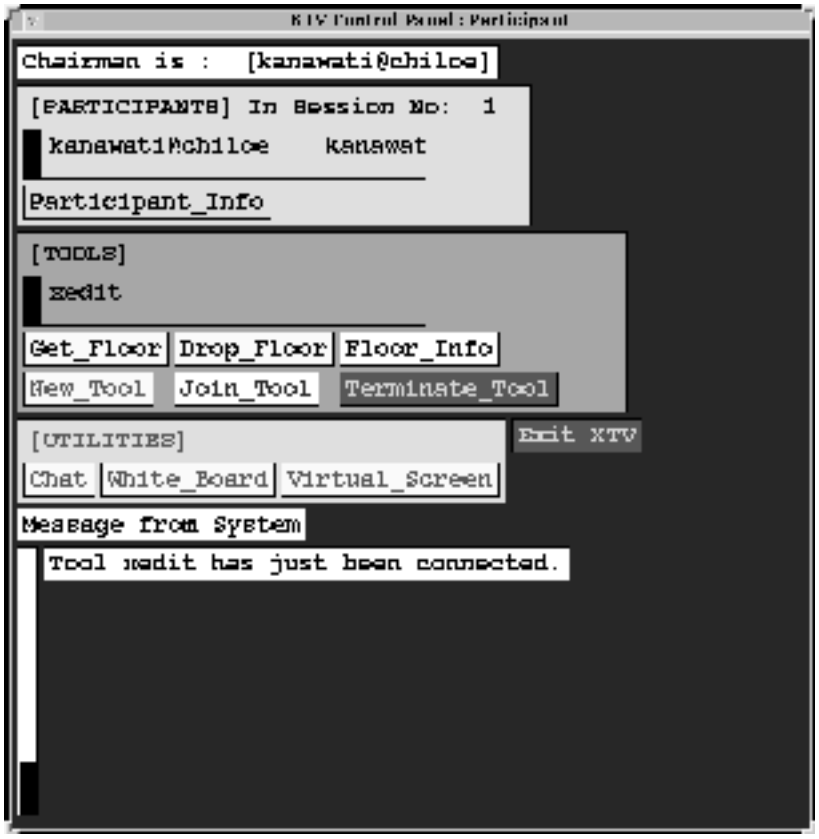


FIG. 2.8 - : Interface-utilisateur de XTV.

2.5.2 La plate-forme SOL

SOL est un environnement de construction d'interfaces de groupe. L'unité de construction est un objet d'interface dit *SOLO* (pour *SOL* object) [Smith et al.94]. Des exemples de *SOLO* sont un bouton, un champ texte ou n'importe quel autre objet utilisé dans la construction des interfaces utilisateurs. À chaque type de *SOLO* est associé un nombre fini d'opérations et une matrice de protection par rôle existant. Un rôle est simplement un groupe d'utilisateurs qui partagent les mêmes privilèges. La matrice de protection comprend deux listes extensibles de droits d'accès. La première liste spécifie le droit d'utilisation des opérations associées au *SOLO* en question. La deuxième liste spécifie le partage des résultats des opérations appliquées entre les différents utilisateurs. Le tableau (2.5) présente un exemple d'une matrice de protection associé à un rôle R sur un bouton qui fournit les opérations suivantes : *Appuyer, Bouger* et *Redimensionner*. Un T ou t (respectivement un F ou f) dans la liste d'utilisation signifient que l'opération correspondante est autorisée (respectivement non autorisée). Un T ou t (respectivement un F ou f) dans la liste de partage signifient que les autres utilisateurs partagent (respectivement ne partagent pas) l'effet de l'exécution de l'opération

Opération	Utilisation	Partage
Appuyer	T	T
Bouger	t	f
Redimensionner	F	F

TAB. 2.5 - : Exemple d'une matrice de protection d'un bouton Solo associé à un rôle R.

correspondante par un détenteur du rôle R. Les valeurs T et F ne sont pas modifiables par l'utilisateur contrairement aux valeurs t et f. Deux types d'interaction avec une application sont définies :

- *Interaction superficielle* : c'est le cas d'une action utilisateur qui ne nécessite pas un traitement par le noyau fonctionnel de l'application. Par exemple, bouger un objet d'interface.
- *Interaction profonde* : c'est le cas d'une action qui nécessite un traitement par l'application partagée. Comme par exemple l'activation d'un bouton.

La figure (2.9) illustre l'architecture générale d'une application construite à l'aide du système *SOL*. L'architecture est d'hybride puisque certaines opérations sont

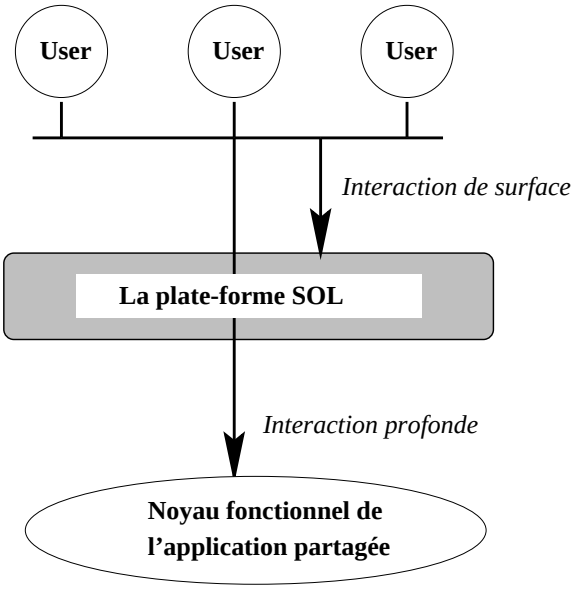


FIG. 2.9 - : L'architecture de la plate-forme SOL

prises en compte en local (cas d'interactions superficielles) d'autres sont exécutées par le noyau fonctionnel central de l'application.

2.5.3 GroupKit

L'objectif de *GroupKit* est de faciliter la construction des collecticiels synchrones [Roseman et al.96b]. La plate-forme fournit un ensemble de services pour la coopération notamment un système de gestion explicite de rendez-vous, un service de communication de groupe et des *widgets* spécialement conçus pour la présentation de la co-présence comme par exemple un pointeur partagé ou une barre de défilement multi-ascenseurs. En revanche *GroupKit* ne fournit aucune politique de gestion de rendez-vous, de participation ou de résolution de conflits. L'implantation des politiques de contrôle est laissée aux constructeurs des applications [Roseman et al.92].

La plate-forme est initialement conçue pour la construction de nouvelles applications. Or, étant développé en langage interprété (en occurrence *Tcl/TK*) il est possible d'adapter des applications développées en même langage pour les rendre coopératives. Les collecticiels construits à l'aide de *GroupKit* ont une architecture totalement dupliquée. Chaque exécution d'un collecticiel constitue une *session* à part entière. *GroupKit* fournit un mécanisme de gestion de sessions qui met en jeux un composant central appelé *registar*. Le *registar* tient à jour une liste des sessions courantes (fonctionne comme un répertoire de sessions). Il ne gère pas les politiques qui définissent comment est créée une session ou comment un utilisateur s'enregistre et s'intégre dans une session existante. À coté du *registar* central, il existe un *registar* client par utilisateur. Ce dernier permet la création, la suppression et l'intégration aux sessions existantes.

2.5.4 CoopScan

CoopScan est une plate-forme générique conçue pour le développement des collecticiels synchrones [Balter et al.96b]. La plate-forme est implantée selon une architecture complètement dupliquée. Elle est fondée sur l'intégration d'applications existantes au niveau de l'application. Aucun mécanisme de rendez-vous n'est fourni par la plate-forme. Un utilisateur qui démarre une nouvelle session prévient les utilisateurs potentiels de l'adresse de la nouvelle session par un outil extérieur (courrier électronique, téléphone). Deux modes de participation peuvent être appliqués : *la participation libre* et *la participation supervisée*. Dans le dernier mode un seul utilisateur, qui a le rôle du président de la session, a le potentiel de parrainer de nouveaux utilisateurs. Ce même utilisateur peut changer le mode de la participation pendant le déroulement de la session.

Deux rôles fonctionnels sont définis : le rôle *président* et le rôle *participant*. Les deux rôles ont les mêmes droits d'accès mais le *président* a le privilège de déterminer la configuration de la session ; c'est à dire la politique de gestion de la concurrence et le mode de participation. Il se réserve le droit de récupérer le droit de parole à n'importe quel instant. À la différence de *XTV*, le président peut démissionner au cours de la session. *CoopScan* implante un modèle de coopération

synchrone. L'interaction est faite selon un mode `WYSIWIS-relâché`. Les participants ne partagent pas forçément les mêmes applications. Lorsqu'un utilisateur lance une nouvelle application, les autres utilisateurs seront prévenues. Le choix de participer à une application est laissé aux utilisateurs eux-mêmes.

La figure (2.10) montre l'interface de contrôle d'une session *CoopScan*. Cette interface affiche d'une manière permanente les informations suivantes : la liste des participants, la liste des applications employées par l'utilisateur local (la liste `In Tools`), la liste des applications employées par les autres participants (la liste `On Tools`), la nom de l'actuel détenteur du droit de parole et le nom du président de la session.

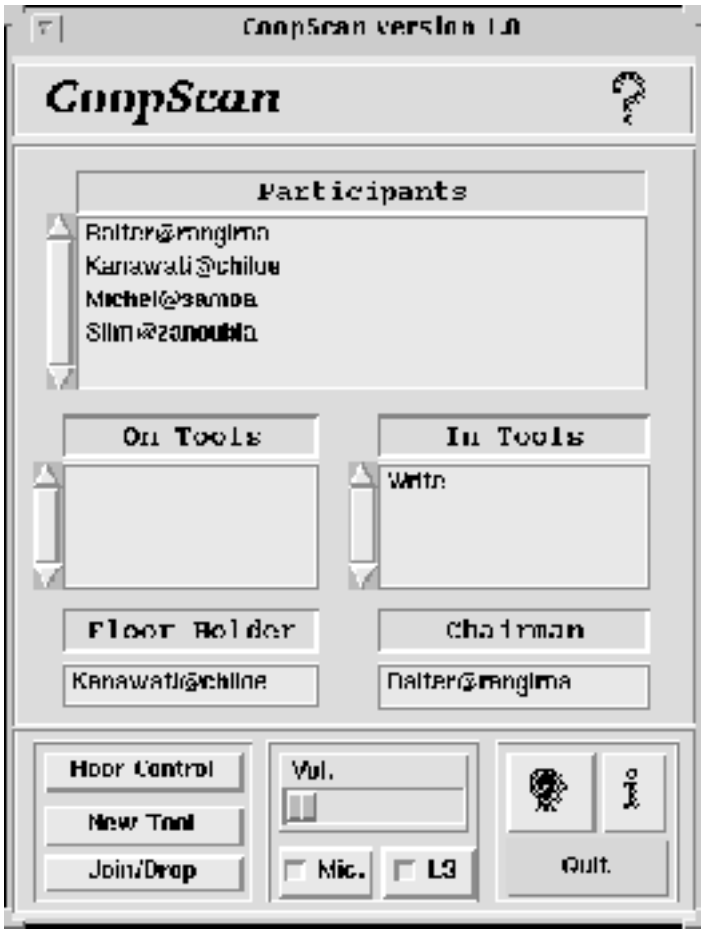


FIG. 2.10 - : Interface de contrôle de CoopScan.

CoopScan fournit une variété de protocoles de tour de rôle. Un seul jeton contrôle l'accès à toutes les applications utilisées dans une session. Le président a le potentiel de changer dynamiquement la politique de gestion de la concurrence.

2.5.5 La plate-forme MEAD

La plate-forme *Mead* est implantée selon une architecture hybride [Bentely et al.92b]. Un serveur central, appelé `Object Store Server (OSS)`, maintient l'ensemble des objets constituant l'espace partagé. Chaque participant a une vue de l'espace partagé présentée par un agent autonome dit `User Display (UD)`. Chaque `UD` est contrôlé par un ensemble d'agents nommés `User Display Agents (UDA)`. Un `UDA` définit une vue particulier sur le `OSS`. Une vue comprend un ensemble d'objets, des méthodes de présentation de ces objets et des opérations sur les objets. Deux `UDA` peuvent avoir des objets en commun mais ils peuvent les présenter différemment et permettre d'appliquer des opérations différentes sur eux. Un `UDA` peut être partagé par deux ou plusieurs `UD`. Les `UDA` peuvent être rajoutés et retirés dynamiquement aux `UD`. Cette architecture permet de réaliser des formes variés de partage d'information entre les utilisateurs. Trois principales formes de partage sont définies :

1. Partage au niveau de la présentation ou ce qu'on appelle habituellement le couplage `WYSIWIS`. Pour implanter cette forme de partage entre deux utilisateurs il suffit que leurs écrans d'affichage (les `UD`) soient associés aux mêmes (`UDA`).

2. Partage au niveau des vues selon lequel deux utilisateurs partagent les mêmes objets mais sans avoir forçément les mêmes présentations ou les mêmes droits sur ces objets. Par exemple deux utilisateurs peuvent partager le même tableau de calcul ; le premier perçoit le tableau sous forme graphique tandis que le deuxième le voit sous forme de tableau.

3. Partage au niveau des objets où les différents utilisateurs ont accès à des ensembles des objets différents (avec possibilité de recouvrement entre les différents ensembles).

Outre la définition des schéma de partage et de coopération différentes, la hiérarchie (`OSS, UDA, UD`) permet de personnaliser le modèle de la coopération entre chaque deux utilisateurs. Autrement dit, les différents participants n'interagissent pas entre eux de la même manière. Cette hiérarchie implante implicitement un système de définition et de gestion de rôles fonctionnels dont le seul inconvénient est qu'il est peu compréhensible par les utilisateurs finales (les rôles sont codés dans les relations `UD-UDA` définies par le programmeur de l'application).

2.5.6 GroupDesign

GroupDesign est un collecticiel de dessins structurés. L'application est implantée selon une architecture dupliquée [Karsenty94a]. Elle est fondée sur l'intégration d'applications existantes au niveau de l'application. Aucun mécanisme de rendez-vous n'est fourni par l'application. La participation à une session est

libre. *GroupDesign* ne fournit pas des rôles fonctionnels d'une manière explicite mais elle permet d'attribuer aux différents utilisateurs de droits différents sur des zones différentes des documents partagés. L'application supporte les deux modes d'interaction : synchrone et asynchrone. Les utilisateurs ont le choix de transiter d'un mode de travail `WYSIWIS-strict` à un mode `WYSIWIS-relâché`. Une façon pour travailler d'une manière asynchrone consiste à verrouiller une zone du dessin. Le verrouillage interdit l'accès en écriture à cette zone. Il peut aussi interdire l'accès en lecture, dans ce cas la zone serait affichée sous forme d'une zone noire avec une icône spéciale qui montre le type du verrou et le nom de l'utilisateur propriétaire.

Un point fort dans la conception de *GroupDesign* est la richesse et la variété des mécanismes qu'elle fournit pour décrire la co-présence. À chaque utilisateur est associée une couleur qui l'identifie. Afin de notifier un utilisateur des actions faites par un autre utilisateur l'application emploie un outil d'*écho*. Deux types d'échos sont utilisés : *écho graphique* et *écho audio*. Les échos sont activés lorsque les utilisateurs sont en mode d'interaction synchrone. L'écho graphique représente l'action d'un utilisateur sur les écrans des autres membres du groupe. Cet écho prend la forme d'une animation. Par exemple si un utilisateur redimensionne un rectangle, l'écho de cette opération consiste en une animation qui modifie progressivement la taille du rectangle (voir figure 2.11). L'écho audio prévient

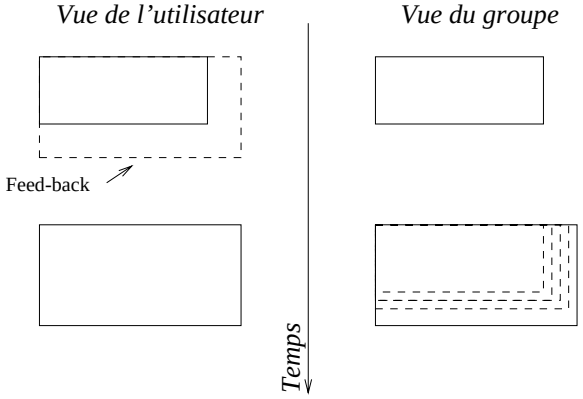


FIG. 2.11 - : Écho graphique dans GroupDesign.

les utilisateurs de l'occurrence d'une opération qui prend place à l'extérieur de leur fenêtre d'affichage. À chaque opération définie par l'application est associé un son spécifique. L'application permet aussi à un utilisateur de visualiser le champ de vision d'un autre utilisateur. De plus les objets dessinés ont toujours le couleur du dernier utilisateur qui l'a modifié. *GroupDesign* fournit un service d'historique de dessins qui permet de tracer l'évolution d'un diagramme et trouver qui a fait quelle modification. Deux principales politiques de gestion de la concurrence sont fournies :

1. Une politique de verrouillage pessimiste. L'acquisition et la libération de

verrous sont fait d'une manière explicite.

2. Une politique optimiste fondée sur l'algorithme *Oreste* [Karsenty et al.93a].

Une dernier caractéristique importante de *GroupDesign* est qu'elle permet la coopération entre des sites et des applications hétérogènes [Karsenty et al.93b].

2.5.7 ActiveMail

La plate-forme *ActiveMail* est fondée sur l'extension du courrier électronique [Goldberg et al.92]. Le modèle de rendez-vous fourni par *ActiveMail* réalise un compromis entre les deux approches explicite et implicite. Pour initialiser une nouvelle session de travail, un utilisateur envoie un message aux participants potentiels. Lorsqu'un utilisateur reçoit un message il a directement les moyens de se connecter à des applications partagées via des boutons de connexion intégrés dans le message. Dans le mesure où la participation est conditionnée par la réception d'un message, l'enregistrement est supervisé. *ActiveMail* supporte le développement de deux types d'applications coopératives : les collecticiels synchrones et les collecticiels asynchrones. Mais il n'offre pas la possibilité de changer dynamiquement la nature d'interaction pour la même application.Tous les utilisateurs ont des attributions identiques. L'implantation d'une politique de gestion de la concurrence est faite au niveau des applications intégrées dans la plate-forme. Aucune politique n'est imposée par la plate-forme.

2.6 Synthèse et discussion

Le tableau (2.6) résume les services offerts par chacune des plates-formes décrites ci-dessus pour résoudre les deux problèmes de la participation et de la coordination dans une activité coopérative. Les trois premières lignes rappellent les choix de construction de chacune des plates-formes. Les notations suivantes sont employées : • pour implantation convenable, ⊗ pour implantation partielle, – pour implantation absente et blanc pour implantation non précisée.

Le tableau 2.6 révèle deux caractéristiques importantes des travaux actuels :

1. La plupart des plates-formes existantes se contentent de supporter une seule type d'interaction : l'interaction synchrone ou l'interaction asynchrone. À l'origine de cette dichotomie se trouve, à notre avis, la diversité des intérêts des différentes disciplines informatiques participant à la recherche en *TCAO*. La construction d'interfaces-utilisateurs qui prennent en compte les besoins de la coopération synchrone est un *défi* majeur lancé aux chercheurs qdans le domaine de l'ingénérie des interfaces homme-machine [Salber95]. Le même type d'applications intéressent les chercheurs dans les domaines de réseaux, des protocoles de communication et des systèmes répartis en raison

	XTV	Sol	Mead	GroupKit	CoopScan	ActiveMail	GroupDesign
Support	IHM	IHM	Donnée	IHM- Donnée	Donnée	Communication	Données
Architecture	Centralisée	Hybride	Hybride	Dupliquée	Dupliquée	Centralisée	Dupliquée
Applications	Int. SF	Nouvelles	Nouvelles	Nouvelles	Int.-App.	Adapt.	Int. App.
Rdv. explicite	⊗			⊗	-	●	-
Rdv. implicite	-			-	-	⊗	-
Intégration du travail individuel	-			-	-	-	-
Enregistrement	●	-	-		●	-	●
Participation dynamique	●				-	●	●
Rôles fonctionnels	-	⊗	⊗	-	-	-	⊗
Interaction synchrone	●	●	●	●	●	-	●
Interaction asynchrone	-	-	-	-	-	●	⊗
Co-présence	⊗	-	-	⊗	⊗	-	●
Configuration dynamique	-	⊗	⊗		●	-	●
Ouverture	-	-	-	●	⊗	-	⊗

TAB. 2.6 - : Satisfaction des services requis pour la gestion de la participation et de la coordination par les plates-formes étudiées.

des contraintes temporelles exigées par ces applications. D'autre part, les chercheurs dans les domaines de documents électroniques, d'automatisation du bureau et de bases de données traitent des problématiques plus proches de la coopération asynchrone.

2. La majorité des plates-formes prêtent peu d'attention à la relation entre le travail individuel et le travail en groupe. La plupart des travaux conçoivent le travail en groupe comme un mode de travail particulier et indépendant du travail individuel. La majorité des plates-formes existantes étant construit par des équipes informatique mono-disciplinaire, les constructeurs semblent être prises par le défi technique que lance le travail en groupe.

Nous rappelons que notre objectif de l'étude des travaux existants est de fixer les choix de construction les plus opportunes pour le développement d'une plate-forme pour le *TCAO* qui offre l'ensemble des services identifiés dans la section (2.3.3). Le tableau précédent montre qu'aucune des plates- formes étudiées n'offre pas la totalité des services requis. L'incomplétude fonctionnelle d'une plate-forme peut être le résultat d'une manque d'effort de développement ou bien *innée* dans les choix de construction de la plate- forme. Évidement, ce que nous intéresse ici est l'identification des choix de construction qui limitent la complétude fonctionnelle des plates-formes. Nous énumérons dans la suite les choix de construction

que nous identifions d'être limitatifs (au niveau fonctionnel) :

Avoir le support au niveau de l'IHM ne permet pas d'implanter un système implicite de gestion de rendez-vous, ni de supporter la transition facile entre les deux modes de travail : individuel et de groupe. C'est le cas de *XTV* et de *Sol*. Les fonctions de la coopération étant exclusivement rattachées au système de gestion de l'interface la sémantique des données manipulées par l'application ne serait pas prise en compte. Par suite il est difficile de distinguer les données personnelles des données partagées. De plus cette approche nécessite l'existence simultanée de tous les utilisateurs dans l'espace du collecticiel.

Avoir le support au niveau de la communication cette approche repose sur l'idée de l'échange asynchrone de l'information. Elle est par conséquent non adaptée à supporter la coopération synchrone. D'un autre coté, cette approche favorise la gestion explicite de la coopération. Il est par exemple difficile d'imaginer l'emploi de cette approche pour implanter un modèle implicite de gestion de rendez-vous.

L'intégration au niveau de système de fenêtres est un cas particulier de la mise en œuvre des fonctions de la coopération au niveau de l'IHM. Cette approche présente donc les mêmes inconvénients présentés ci- dessus. Les informations échangées entre les utilisateurs, selon cette approche, ont un niveau sémantique très bas (fenêtre, ascenseur, pointeur) ; toute entité d'un niveau sémantique plus haut est forcément partagée par les utilisateurs. Ceci implique l'impossibilité d'associer des rôles fonctionnels différents aux différents utilisateurs et que le seul type d'interaction possible est l'interaction synchrone stricte (`WYSIWIS`). D'autre part, les systèmes de fenêtres fondés sur le paradigme client-serveur tel que *X-Window* impose l'emploi des protocoles de tour de rôle pour gérer la concurrence. L'emploi des protocoles optimistes peut générer des séquences non autorisés par le protocole client- serveur et en conséquence faire arrêter le système. Par exemple dans le cas de *X-Window* une séquence formée de deux clics souris (`MouseDown`) non séparés par un événement de type souris relâchée viole le protocole *X* et entraîne l'arrêt du système.

La mise en œuvre selon une architecture centralisée force le choix des politiques restrictives pour la gestion de la concurrence. Il est difficile d'implanter une politique qui permet l'accès simultané à l'espace partagé.

2.7 Conclusion

L'objet de ce chapitre est d'étudier et de comparer les différentes approches pour la construction de collecticiels. Dans un premier temps nous avons présenté

une étude de définition et de classification de collecticiels. Trois classes sont identifiées : les collecticiels procéduraux, les collecticiels à immersion et les collecticiels contractuels. Dans un deuxième temps nous avons proposé une nouvelle classification des principales approches de construction des plates-formes pour le *TCAO*. La classification proposée est fondée sur les trois axes suivants :

- Le niveau du support qui peut être au niveau de l'IHM, au niveau de données ou au niveau de la communication.
- L'architecture de mise en œuvre qui peut être centralisée, dupliquée ou hybride.
- La nature des applications supportées qui peuvent être des nouvelles applications ou des applications existantes. Dans le dernier cas, nous avons distingué deux approches de réutilisation d'applications existantes : par adaptation et par intégration. L'intgration d'applications peut se faire au niveau du système de fenêtre ou au niveau de l'application.

Des exemples représentatifs des différentes approches de construction évoquées ci-dessus sont décrits. Nous les avons comparé en fonction de leurs complétude fonctionnelle vis à vis des services requis pour résoudre deux problèmes fondamentaux: la participation à une activité coopérative et la coordination à l'intérieur d'une activité. Notre étude de comparaison nous a permet de montrer que certains choix de construction ne sont pas pertinents pour la construction d'une plate-forme générique pour le *TCAO*. Les principaux choix **à éviter** sont :

1. avoir le support au niveau de l'IHM,
2. avoir le support au niveau de la communication,
3. la mise en œuvre selon une architecture centralisée et
4. l'intégration d'applications existantes au niveau de système de fenêtres.

Chapitre 3

Colt, un environnement de coopération

3.1 Introduction

La plupart des collecticiels contractuels[1] existants souffrent de deux défauts qui sont à notre avis à l'origine de leur faible notoriété dans les organisations réelles :

1. **La coopération entre collecticiels :** Peu de mécanismes sont offerts par les plates-formes actuelles pour coordonner et orchestrer l'exécution d'un groupe de collecticiels. Pourtant dans la vie réelle des organisations et des entreprises les activités sont rarement isolées l'une de l'autre [Schmidt et al.93].

2. **L'intégration du travail individuel :** Les collecticiels existants prêtent peu d'attention à l'intégration des modes du travail individuel et en groupe.

Pour remédier aux défauts précédents, le concept d'Environnement de Coopération (EdC) est introduit [Schmidt et al.93, Navarro et al.93]. Un EdC fournit à un groupe organisé d'individus un support pour orchestrer le lancement et l'exécution de collecticiels variés en fonction de besoins du groupe. Comme le fait remarquer *Croisy* dans [Croisy94] le concept d'EdC est une généralisation directe de celui du collecticiel. Si nous associons un collecticiel à une relation $N \rightarrow 1$ où N personnes interagissent avec une application un EdC serait associé à une relation $N \rightarrow M$ où N utilisateurs travaillent ensemble en utilisant M applications.

Nous présentons dans ce chapitre un nouvel environnement de coopération appelé *Colt* (abréviation de *Collaboration Terrain*). *Colt* est un environnement intégré pour le travail individuel et le travail coopératif. Les utilisateurs partagent un espace d'information où chacun a le droit de *voir* et de *se mouvoir* selon des *rôles*

[1] voir la définition dans 2.2.4

qui lui sont attribués. Les utilisateurs peuvent créer et participer à des activités coopératives. Une activité est définie par l'ensemble des règles de contrôle qui y sont appliquées et par les documents qui y sont manipulés. Chaque activité définit un sous-espace de l'espace d'information partagé. Une activité dans *Colt* peut être persistante ou non. Une activité est persistante si tous les acteurs peuvent se déconnecter puis revenir plus tard et retrouver l'activité dans le même état observé par le dernier des acteurs déconnectés. Récemment quelques travaux se sont intéressé à la construction d' EdC. Nous citons en particulier les travaux fondés sur la double métaphore *pièce-outil* tel que *TeamRooms* [Roseman et al.96b] et *Co-Learn* [Croisy94]. Le principe est de permettre aux utilisateurs de construire des *pièces* dans lesquelles ils partagent des *outils*. Contrairement à ces travaux nous découplons dans *Colt* la métaphore d'outil de celle de la *pièce* ; les utilisateurs rassemblent des *outils individuels* pour travailler ensemble dans une *pièce*. Les outils ne sont pas partagés. Nous montrons que notre approche permet une intégration plus simple du travail individuel et le travail en groupe et qu'elle permet de construire un système souple et adaptable.

Dans la section suivante nous précisons les objectifs fixés pour l'environnement *Colt*. La section (3.3) décrit les choix de construction adoptés pour atteindre les objectifs fixés. Une description informelle de l'environnement est donnée dans la section (3.4). L'architecture générale du système est exposé dans la section (3.5). Un premier prototype de *Colt* est réalisé sur une plate-forme *Unix* et est implanté en utilisant l'environnement graphique *TCL-TK*. Nous décrivons ce prototype dans la section (3.6). Une évaluation fonctionnelle de *Colt* est faite dans la section (3.7).

3.2 Objectifs

L'objectif principal de *Colt* est de fournir un système intégré qui assiste un ensemble organisé d'individus dans leur travail. Nous ne visons pas un type précis d'organisations mais nous cherchons plutôt à fournir un environnement générique et adaptable qui permet la construction et l'exécution des collecticiels contractuels variés. Les services requis d'un tel environnement sont présentés et analysés dans (2.3.3). Cinq groupes de services ont été identifiés à savoir : 1) la conscience de groupe, 2) le partage et l'échange d'informations, 3) la gestion des groupes d'utilisateurs, 4) la reconfiguration dynamique et 5) l'ouverture du système à toute évolution future. S'occuper de l'ensemble des services requis échappe au volume d'une seule thèse. Dans le cahier des charges de *Colt*, nous mettons l'accent sur les points suivants :

1. Fournir un modèle générique de rendez-vous.
2. Permettre le lancement et l'exécution de scénarios coopératifs variés. Par scénarios variés nous voulons dire des activités coopératives synchrones ou

asynchrones, planifiées ou fortuites, persistantes pu on persistantes.

3. Rendre aisé le passage entre le travail individuel et le travail en groupe.

Les critères cités ci-dessus sont inspirés de l'étude des deux problèmes de la participation et de la coordination dans les activités coopératives présentés dans la section (2.3).

3.3 Choix de construction

Nous exposons et nous argumentons dans cette section les choix que nous avons fait pour la mise en œuvre de *Colt*. Deux sortes de choix sont à faire :

1. Le choix de la stratégie de mise en œuvre que nous exprimons en fonction des approches de construction des collecticiels identifiées dans la section (3.3).

2. Le choix des métaphores à employer pour la conception de l'environnement et pour la présentation de l'interface utilisateur du système.

3.3.1 Stratégie de mise en œuvre

Dans (2.4) nous avons identifié trois critères pour la classification des approches de construction des collecticiels : 1) le niveau du support, 2) le schéma de mise en œuvre et 3) la nature des applications supportées. Nos choix pour la mise en œuvre de *Colt* sont fondés sur les résultats de l'étude de comparaison fonctionnelle des différentes approches faite dans (2.6), à savoir :

- **Le niveau du support est au niveau des données**. Les deux autres niveaux identifiés (le niveau de l'interface utilisateur et le niveau de la communication) ne permettent pas l'implantation de certains modèles de coopération ou de fonctions de gestions de rendez-vous.

- **L'architecture de mise en œuvre est totalement dupliquée**. Ce choix est motivé par les nombreux avantages de l'architecture dupliquée notamment : la réactivité des applications, la réduction du trafic réseau, la disponibilité des données et une meilleur tolérance aux pannes [Roseman et al.92, Balter et al.96a].

- **Les applications construites sont nouvelles**. Nous supportons également l'adaptation et l'intégration des applications au niveau de l'application sous certaines contraintes que nous précisons ultérieurement. L'intégration d'applications au niveau de système de fenêtres n'est pas supportée en raison des nombreuses limitations fonctionnelles qu'elle induit.

3.3.2 Choix de métaphores

Les concepteurs des applications interactives ont recours aux métaphores pour identifier les fonctions et les services à fournir et pour munir l'interface utilisateur de l'application d'une représentation facile à interpréter et à utiliser. Souvent la même métaphore est employée pour les deux tâches à la fois. Nous présentons dans la suite les métaphores les plus employées pour modéliser les collecticiels :

1. *Les collecticiels sous forme de réunions* ou de sessions. La coopération entre deux utilisateurs nécessite qu'ils soient tous les deux présents au même instant dans l'espace défini par l'activité. Cependant l'interaction entre les utilisateurs peut être de nature synchrone ou asynchrone (voir 2.2.3). L'activité se termine avec le départ du dernier participant. Les œuvres produites dans le contexte de l'activité n'existeront plus après la terminaison de l'activité [2].

2. *Les collecticiels sous forme de procédures.* Selon cette métaphore de conception chaque participant a le potentiel d'exécuter un ensemble fini d'actions dont les résultats et les impacts sur les vues des autres utilisateurs sont déterminés à l'avance. Les utilisateurs peuvent travailler en même temps ou en temps différé. Bien que cette métaphore de conception convienne à l'implantation des collecticiels procéduraux elle est employée parfois pour modéliser des collecticiels contractuels dans lesquels les actions possibles sont bien limitées et déterminés à l'avance et où le travail à accomplir suit un plan structuré. C'est le cas par exemple d'une application de *vote* ou d'un système de prise de décisions.

3. *Les collecticiels sous forme de locaux partagés.* Chaque exécution d'un collecticiel prend place dans un *local* bien défini. Ainsi pour participer à une activité l'utilisateur doit se rendre au local qui lui est associé. Un local est défini par son adresse et son étendue (les données et les outils qu'il contient). Les utilisateurs peuvent accéder à un local quand ils veulent (du moment qu'ils ont l'autorisation de s'y rendre). Lorsque deux utilisateurs sont dans le même local en même temps ils peuvent interagir entre eux d'une manière synchrone ou asynchrone en fonction des outils dont ils disposent. Les œuvres contenues dans un local y restent tant que le local existe. Dans ce sens un local peut être vu comme une *session* persistante. La persistance permet aux utilisateurs de travailler et de faire évoluer leur tâche commune aussi bien en temps différé qu'en même temps ; le travail individuel trouve aussi sa place dans un système reposant sur la métaphore de locaux (être seul dans le local).

Comparée aux autres métaphores le local partagé est la métaphore la plus générique puisqu'elle permet de modéliser de types variés d'activités coopératives.

[2] Sauf s'ils sont explicitement sauvegardés par l'un des participants.

En conséquence la plupart des environnements de coopération qui se veulent génériques sont construits selon cette métaphore. Deux principales variations d'implantation de la métaphore de locaux partagés sont proposées dans la littérature du *TCAO*: la *pièce* et le *terrain*.

- La métaphore de la *pièce*: L'espace partagé est composé d'un ensemble de *pièces*. Une *pièce* correspond à une activité; elle contient les documents et les outils nécessaires pour mener cette activité. Pour manipuler un document il faut d'abord accéder à la *pièce* qui le contient. Lorsque deux utilisateurs se trouvent dans la même *pièce* ils peuvent travailler ensemble sur les documents contenus dans cette *pièce*. Le concept de *pièce* est donc un moyen pour structurer à la fois l'espace partagé et les activités des utilisateurs. Les utilisateurs peuvent créer de nouvelles *pièces* en fonction de leurs besoins. Un mécanisme de déplacement entre les différentes *pièces* doit être fournit par le système. Un premier exemple de la réalisation de cette métaphore est constitué par les systèmes dits *MUD* (pour Multi User Dungeons). Un autre exemple est le système *Co-Learn* [Croisy95] qui définit des *pièces* typées. Le type d'une *pièce* détermine la nature de l'activité qui peut y prendre place. D'autres systèmes comme *TeamRooms* [Roseman et al.96b] fournissent des *pièces* identiques que les utilisateurs façonnent et meublent en fonction de leurs besoins.

- Les locaux sous forme d'un *terrain*. L'espace partagé est représenté sous forme d'une structure abstraite (non conforme à une réalité physique) dotée d'un système d'aide à la navigation entre les données. Un exemple classique est de présenter l'espace partagé sous forme de structure hypertexte comme c'est le cas dans le système *BSCW* [Bentley et al.97]. Un autre exemple est constitué par les systèmes dits *PIT* (Populated Information Terrains) [Benford et al.95]. Un *PIT* consiste à donner à un système d'information (par exemple un système de fichiers) une configuration spatiale qui permet aux utilisateurs de visualiser les données partagées et de voir où sont les autres dans le système. Les différentes approches de configuration spatiales des *PIT* sont présentées dans [Benford et al.95].

Selon la métaphore de la *pièce*, la définition de l'activité (création d'une pièce) précède l'existence des données à manipuler dans l'activité tandis que selon la métaphore du *terrain* les données existent avant les activités. C'est l'accès simultané de deux ou plusieurs utilisateurs au même document qui peut engendrer une activité. Ainsi la métaphore de la *pièce* supporte l'exécution des activités planifiées tandis que celle du *terrain* supporte principalement les activités non planifiées. Le choix de la métaphore de la présentation de l'interface homme machine d'un système fondé sur la métaphore de locaux partagés dépend fortement du mécanisme de navigation fourni par le système. Dans le cas d'un *PIT* l'espace partagé

est souvent représenté sous forme d'un graphe où les nœuds représentent les documents et les arrêtes représentent des liens sur lesquels l'utilisateur peut *glisser* pour aller d'un document à un autre (comme dans le cas de la navigation dans un système hypertext). Différentes méthodes de navigation sont proposées dans les systèmes fondés sur la métaphore de la *pièce*. Dans *GroupKit* les *pièces* disponibles sont présentées dans une seule fenêtre. Chaque *pièce* est représentée sous forme d'un rectangle qui porte le nom de la *pièce* et les noms des utilisateurs qui sont dedans. Il suffit d'appuyer sur le rectangle pour entrer dans la *pièce* [Greenberg91]. La disposition des *pièces* dans l'espace de la fenêtre est aléatoire. *Co-Learn* emploie une stratégie semblable à la précédente sauf que les *pièces* sont organisées dans un sorte de plan d'immeuble [Croisy95]. L'existence d'un plan facilite l'orientation des utilisateurs dans l'espace. D'autres systèmes emploient la métaphore de la *porte* pour assurer le déplacement entre les différentes *pièces*. Le système *MILAN* [Condon92] présente une interface utilisateur sous forme d'un *village* ; les *pièces* sont organisées en immeubles qui sont à leur tour organisés en quartiers. Ainsi pour aller d'une *pièce* à une autre il faut passer par un niveau organisationnel plus élevé. Dans *Colt* nous empruntons une approche originale que nous résumons par les trois points suivants :

- La métaphore de conception est une combinaison des deux métaphores : la *pièce* et le *terrain*. Les utilisateurs partagent un espace d'information qui contient des documents partagés et des *pièces*. La combinaison des deux métaphores nous permet de supporter à la fois les activités planifiées et les activités non planifiées.

- La métaphore de la présentation de l'interface homme machine est la métaphore classique de surface du bureau. La surface du bureau d'un utilisateur porte tous les documents que l'utilisateur a le droit de voir, qu'ils appartiennent à une *pièce* ou au *terrain*. L'utilisateur accède à un document contenu dans une *pièce* (donc appartenant à une activité coopérative) de la même manière qu'il accède à un document individuel ou à un document partagé (appartient au terrain). L'uniformité d'accès et d'emploi des documents renforce l'intégration entre le mode de travail individuel et le travail en groupe.

- Contrairement à la majorité des systèmes implantant la métaphore de la *pièce*, nous découplons dans *Colt* le concept d'outil de celui de la *pièce*. Une *pièce* est une structure de contrôle et d'organisation qui contient un ensemble des documents que les utilisateurs peuvent manipuler tout en respectant des règles d'accès et d'interaction propres à la *pièce*. Pour accéder à un document chaque utilisateur emploie son propre outil individuel. Ainsi le même outil (application) est utilisé pour travailler individuellement ou en groupe. Ceci facilite l'emploi de notre environnement et le rend plus

flexible en permettant aux utilisateurs de coopérer tout en employant des outils hétérogènes.

3.4 Colt : présentation informelle

L'architecture de *Colt* fait intervenir deux entités conceptuelles dites *le Terrain* et *l'Agent Colt* (*AgC*). Le *Terrain* est un serveur auquel se connectent les utilisateurs via des *clients* qui sont les *AgC*. Il ne faut pas confondre ici le concept du *Terrain* dans *Colt* avec la métaphore du *terrain* présentée dans la section (3.3.2). Le *Terrain* implante la totalité de l'espace partagé : les documents partagés et les *pièces* disponibles dans le système. Il implante aussi les modules de contrôle des activités (coopératives ou individuelles) qui peuvent prendre place dans l'environnement. De son côté un *AgC* représente un utilisateur enregistré dans l'environnement. Il permet à l'utilisateur qu'il représente d'agir dans le *Terrain* en fonction des privilèges qui lui sont attribués. Ainsi le déploiement de *Colt* consiste à avoir un seul *Terrain* et autant d'*AgC* qu'il y a d'individus dans l'organisation cible. Lorsqu'un utilisateur se connecte au *Terrain*, son *AgC* lui montre la *vue* qui lui est attribué sur ce *Terrain*. La *vue* d'un utilisateur comprend les documents et les *pièces* disponibles pour cet utilisateur. Un document est disponible pour un utilisateur si ce dernier a au moins le droit de le lire. Une *pièce* est disponible pour un utilisateur si ce dernier est y déjà enregistré ou invité à s'y rendre (coopérant potentiel). Les documents contenus dans les *pièces* auxquelles l'utilisateur est invité ne figurent évidemment pas dans la *vue* de l'utilisateur.

La définition de la *vue* d'un utilisateur dépend de son rôle dans l'organisation. Un système de définition et d'attribution des rôles organisationnels est implanté par le *Terrain* (4.5.1.4). La figure (3.1) montre l'interface utilisateur d'un *AgC* construit selon la métaphore de surface du bureau. Contrairement aux implantations habituelles de cette métaphore, la surface du bureau dans *Colt* ne porte que des documents. Les applications disponibles (les outils) pour l'utilisateur sont regroupées sous le menu `Tools`. Ce choix est fait dans l'optique de faciliter la localisation des outils. Outre le menu `Tools` la barre de menu porte quatre entrées qui sont les suivantes :

1. Le menu `Activities` propose les commandes de création de nouvelle activités de groupe (*pièces*), de visualisation de la liste des activités dans lesquelles l'utilisateur est enregistrées et la liste d'activités auxquelles il est invité à participer.

2. Le menu `Organisation` propose les commandes de visualisation de la hiérarchie de l'organisation des utilisateurs (les rôles organisationnels et les relations entre eux). Ce menu propose à l'administrateur de l'environnement des commandes de gestion de l'organisation (création, modification et effacement des rôles ou des liens entre les rôles).

3. Le menu `People` propose les commandes de renseignement sur les utilisateurs enregistrés dans l'environnement.

4. Le menu `Manager` propose une commande qui permet à l'utilisateur de changer son mot de passe et une autre pour quitter l'environnement.

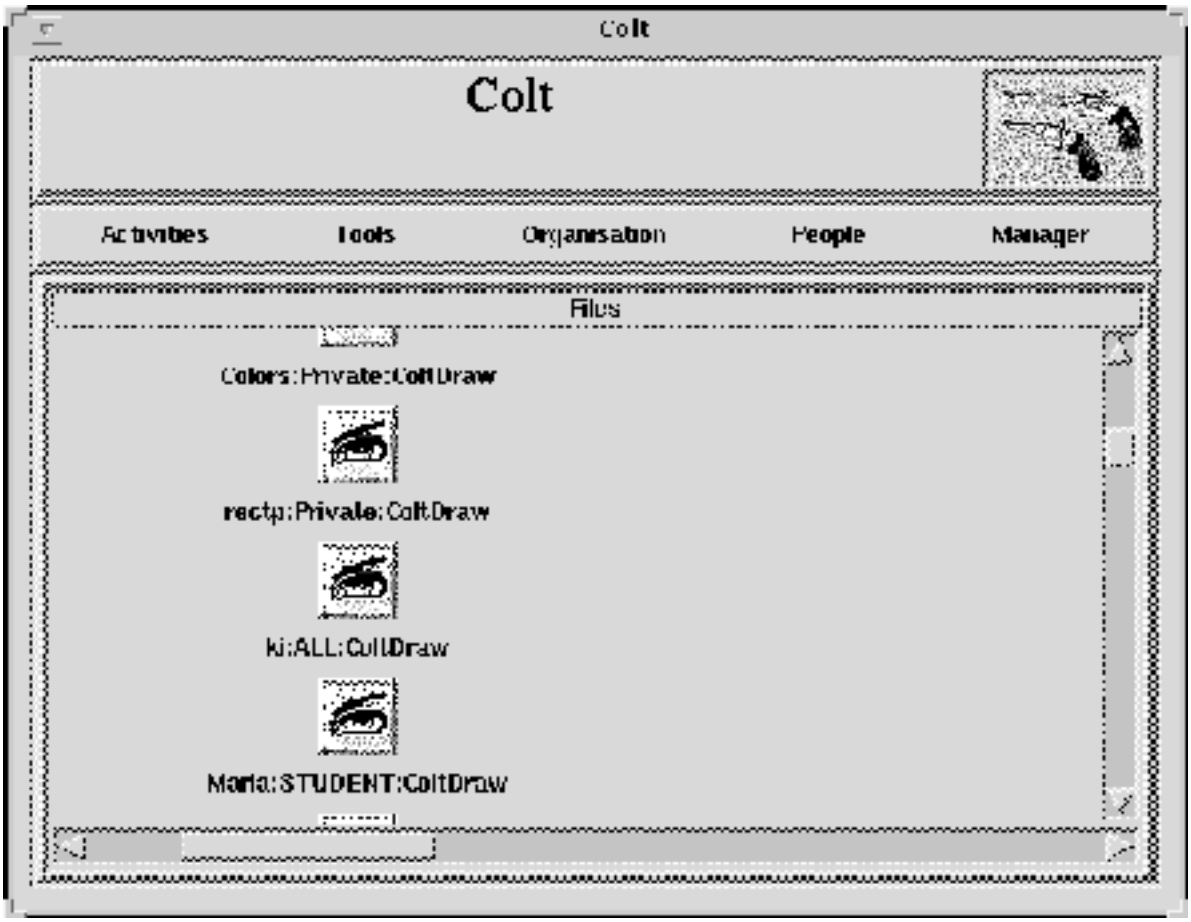


FIG. 3.1 - : L'interface utilisateur d'un agent Colt.

Les documents sont représentés sur le surface du bureau sous forme icônique. L'icône d'un document désigne l'outil qui peut être utilisé pour manipuler le document. Le mode d'affichage d'une icône change en fonction de l'état de l'emploi du document qu'elle représente.

De point de vue d'un utilisateur un document peut être dans l'un des trois états suivants :

1. *État de repos* où le document n'est pas utilisé par aucun utilisateur. Son icône est affichée en mode `plat`.

FIG. 3.2 - : Visualisation d'un document : Draw0 est au repos, Draw1 est occupé et Draw2 est en état d'exploitation.

2. *État d'occupation* où le document est chargé par un autre utilisateur mais pas par l'utilisateur local. Dans ce cas l'icône est affichée en mode `sur-élévé`.

3. *État d'exploitation* où le document est utilisé par l'utilisateur local. L'icône est affichée en mode `enfoncé`.

Les trois modes d'affichage d'icônes sont illustrés sur la figure (3.2).

À chaque document est attachée une *fiche* d'information que l'utilisateur peut afficher en appuyant deux fois (bouton droit de la souris) sur l'icône du document. Un exemple d'une fiche d'information est illustré à la figure (3.3)

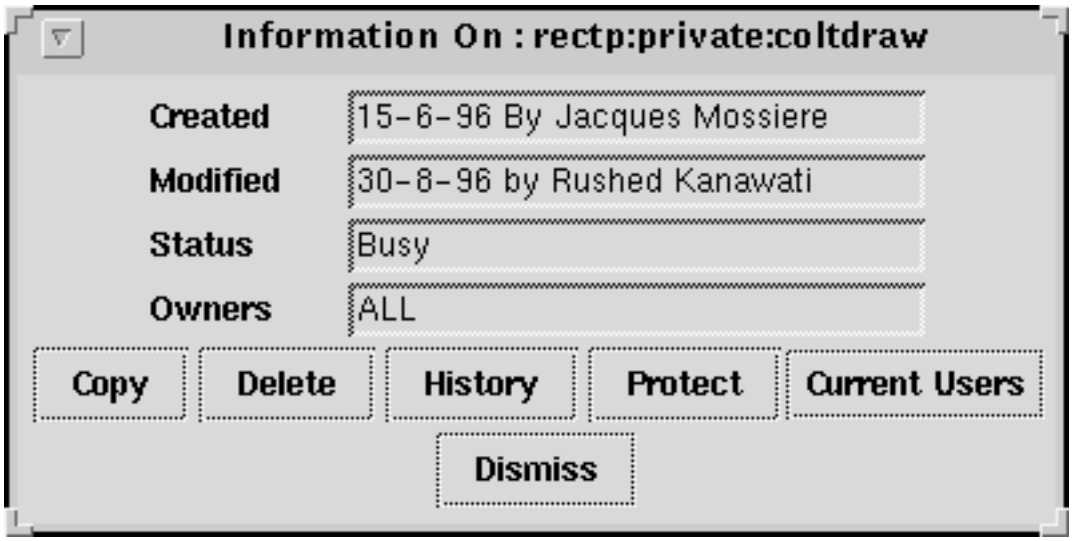


FIG. 3.3 - : Les opérations et les informations disponibles sur un document.

Outre la description du document (dates de création et de dernière sauvegarde, état d'emploi et les propriétaires du document) la fiche porte des boutons qui permettent d'appliquer les commandes usuelles sur les documents (copier, effacer et changer le mode de protection). La fiche porte deux autres boutons ; le premier (`History`) permet d'afficher l'historique d'accès au document tandis que le deuxième a pour effet d'afficher la liste des utilisateurs actuels du document.

Lorsqu'un utilisateur accède à un document l'événement est propagé à tous les autres utilisateurs connectés au *Terrain* et qui ont le même document sur leur *bureau*. Deux utilisateurs partagent un document dans l'un des deux cas suivants :

1. si le document appartient à une activité coopérative dans laquelle les deux utilisateurs sont enregistrés. Nous disons que le document est un *document de coopération*,

2. si le document n'appartient pas à une activité coopérative mais si les deux utilisateurs ont au moins le droit de le lire (par exemple les deux utilisateurs ont le même rôle organisationnel). Nous désignons ce type de documents par le terme *document public*.

Lorsque deux ou plusieurs utilisateurs accèdent en même temps à un document public ils vont se retrouver dans une situation de coopération synchrone où ils peuvent manipuler tous ensemble le document chacun en fonction des droits qu'il possède sur ce document. Nous appelons cette situation une *rencontre*. Les *rencontres* sont des activités de groupe non-planifiées. La création des rencontres repose sur un modèle de rendez-vous implicite.

L'environnement *Colt* fournit aussi un modèle explicite de rendez-vous implanté sous forme d'un répertoire d'activités. Les commandes de la gestion de ce répertoire sont regroupées sous le menu `Activities`. Afin de faciliter la tâche de la création de nouvelles activités, nous avons opté vers un mécanisme de création par instanciation des *modèles* prédéfinis. Un *modèle* décrit les protocoles de contrôle (politiques d'enregistrement, d'intégration et de médiation) qui peuvent être appliqués dans l'activité et les privilèges attribués aux participants à l'activité.

La définition des privilèges des participants repose sur un système de définition et d'attribution de *rôles fonctionnels*. Par exemple dans un *séminaire* nous définissons les trois rôles fonctionnels suivants : `orateur, public` et `animateur` (ou président). À chaque participant à une activité sont associés un ou plusieurs rôles fonctionnels. En fait, lors de la création d'une nouvelle activité, l'utilisateur doit spécifier qui parmi les utilisateurs enregistrés dans l'organisation peut avoir quel rôle fonctionnel.

Un utilisateur peut être enregistré dans plusieurs activités, mais il ne peut être actif que dans une seule activité à la fois. Ce choix nous évite de se préoccuper du problème complexe de fusion d'activités. Pour manipuler les documents disponibles dans une activité les utilisateurs emploient les mêmes outils qui sont disponibles sous le menu `Tools`. Cependant les participants à une activité disposent d'un autre type d'outils que nous appelons les *Outils d'Aide à la Conversation* (OAC). Un OAC est un outil de communication directe entre les participants. Des exemples sont : un tableau blanc partagé ou un canal audio-vidéo temps réel. Il est important de noter qu'un OAC ne peut être employé que dans le contexte d'une activité. Donc deux utilisateurs ne peuvent pas communiquer directement

sauf s'ils sont dans la même *pièce*. Ce choix découle de notre objectif de fournir un environnement de coopération contractuelle ; nous évitons l'immersion des utilisateurs dans l'environnement informatique (voir 2.2).

3.5 Architecture générale

3.5.1 Le Terrain

Le *Terrain* définit et gère le contexte de l'organisation cible de l'environnement. À ce titre il maintient les documents manipulés dans l'environnement, définit l'organisation et les utilisateurs qui la forment et décrit la nature des activités qui peuvent avoir lieu dans l'environnement. Les fonctions citées ci-dessus sont assurées par des agents spécialisés, appelés *les gestionnaires* (voir figure 3.4). La structuration du *Terrain* en gestionnaires coopérants nous permet de décrire son fonctionnement indépendamment du schéma de mise en œuvre adopté. Cinq gestionnaires spécialisés constituent le *Terrain* : 1) le gestionnaire de l'espace de travail, 2) le gestionnaire de l'organisation, 3) le gestionnaire des activités, 4) le gestionnaire de la protection et 5) le gestionnaire de la communication.

3.5.1.1 Le gestionnaire de l'espace du travail (GET)

Le gestionnaire de l'espace du travail (*GET*) assure le stockage permanent des documents utilisateurs et l'accès à ces documents. Il fournit une interface classique de manipulation de documents dont les principales fonctions sont données dans le tableau (3.1).

Fonction	Description
`NewDoc`	Création d'un nouveau document.
`OpenDoc`	Ouverture d'un document existant.
`WriteDoc`	Modification d'un document existant.
`CloseDoc`	Fermeture d'un document.
`DelDoc`	Détruire un document
`ChDocOwn`	Changement du propriétaire du document.
`GetDocInfo`	Donner les informations contenues dans le descripteur.

TAB. 3.1 - : Principales fonctions de manipulation de documents.

L'environnement *Colt* étant conçu pour fonctionner sur des systèmes d'exploitation existants, la gestion de la persistance des documents est gérée par le système de gestion de fichiers (SGF) fourni par le système sous-jacent. Par

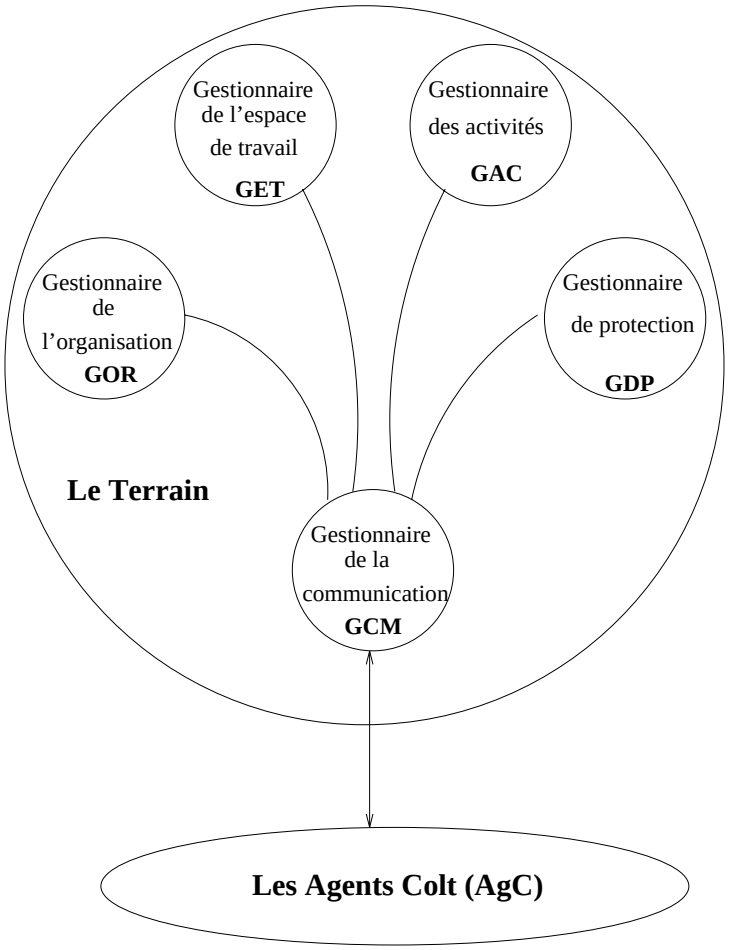


FIG. 3.4 - : Architecture générale du Terrain.

conséquent le *GET* est affranchi des tâches de bas niveau (gestion de la mémoire, localisation des documents, ...).

Le *GET* implante une couche intermédiaire entre *Colt* et le SGF employé. L'intérêt principal de cette couche est de *collecter* une partie des informations nécessaires pour la présentation de la co-présence des utilisateurs dans l'environnement. Le rôle d'interface que joue le *GET* entre *Colt* et le SGF sous-jacent lui permet de connaître, pour chaque document, l'état d'emploi, l'historique et la liste des utilisateurs actuels et potentiels. Afin de sauvegarder et d'exploiter les différentes informations citées ci-dessus le *GET* associe à chaque document *Colt* une structure de contrôle dite *descripteur de document* (DD). Les informations contenues dans un DD sont explicitées dans le tableau 3.2.

Nous décrivons dans la suite l'utilité et la fonction de chacun des champs constituant un DD. Les trois premiers champs dans un DD servent pour *désigner* et *localiser* les documents. La chaîne d'accès à un document est la suivante :

$$Nom \rightarrow_{AgC} Identificateur \rightarrow_{GET} \text{Adresse physique}$$

Champ	Contenu
Identificateur	Identificateur unique attribué par le GET.
Nom	Nom symbolique attribué par le créateur du document.
Adresse physique	Listes des *fichiers* contenant le document
Applications	Liste d'applications pouvant manipuler le document.
Propriétaire	identificateur d'un utilisateur ou d'un groupe ou d'une activité.
Historique	Nom du fichier contenant l'historique d'accès au document.
Les utilisateurs potentiels	Liste des utilisateurs **connectés** au Terrain ayant le droit d'accéder au document.
Les utilisateurs actuels	Liste des utilisateurs employant actuellement le document.

TAB. 3.2 - : Structure d'un DD.

L'agent colt d'un utilisateur (l'entité qui gère l'interface utilisateur) s'appuie sur une table de correspondance pour retrouver l'identificateur d'un document à partir de son *Nom* (voir 3.5.2). Le *GET* en consultant le DD adéquat *localise* les fichiers implantant le document désigné. Le *Nom* d'un document ne le caractérise pas d'une manière unique. Deux documents peuvent avoir le même *Nom*. Rien n'empêche que deux utilisateurs utilisent le même *Nom* pour nommer deux documents différents. Un problème se pose si deux documents ayant le même nom font partie d'une même *vue* d'un utilisateur. Pour résoudre ce problème l'*AgC* renomme *localement* l'un des deux documents. C'est ainsi que le même document peut être désigné différemment par différents utilisateurs.

Le champ *Applications* énumère les applications qui peuvent manipuler le document. Il définit le type du document . Le même document peut être manipulé par des applications différentes. Par exemple un dessin peut être édité par un éditeur graphique ou visualisé par un *browser* d'images. La liste des applications pouvant manipuler un document est déterminée automatiquement lors de la création du document en fonction des relations prédéfinies de compatibilité entre l'application qui crée le document et les autres applications disponibles dans l'environnement.

Chaque document a un seul propriétaire dont l'identité est donnée par le champ *propriétaire*. Le propriétaire a tous les droits sur le document qu'il possède y compris la définition des droits des autres utilisateurs sur ce document. La notion de propriété sert aussi à structurer l'espace des documents colt. Chaque propriétaire définit un groupe de documents.

Le propriétaire d'un document peut être un: 1) un utilisateur, 2) un groupe d'utilisateurs désigné par un rôle organisationnel ou 3) une activité coopérative. La structuration facilite la localisation d'un document en réduisant la recherche à un sous ensemble de documents. Cependant il faut trouver des critères qui permettent de structurer la liste des documents appartenant au même propriétaire. Le choix de tels critères est un problème ouvert [Coulouris et al.94].

Les trois derniers champs d'un DD portent les informations décrivant la *co-présence* des utilisateurs. Ils désignent respectivement, le fichier qui contient l'historique d'accès au document, la liste (UP) des utilisateurs connectés au *Terrain* ayant le droit d'accéder au document et finalement la liste (UA) des utilisateurs employant le document. La construction de la liste UP se fait au fur et à mesure de la connexion des utilisateurs au *Terrain* (voir 3.5.2.1). Lorsqu'un utilisateur accède au document, le GET transfère son identificateur de la liste UP à la liste UA. Ainsi l'intersection des deux listes est vide. Le contrôle de l'affichage de l'icône d'un document repose sur les informations contenues dans les deux listes UP et UA. Les règles suivants sont appliquées pour déterminer le mode d'affichage :

- L'icône est affichée sur les surfaces de bureaux des utilisateurs referencés dans les deux listes.
- L'icône est affichée en mode `enfoncé` pour tout utilisateur referencé dans la liste UA.
- Si la liste UA est non vide les utilisateurs referencés dans la liste UP affichent l'icône en mode `sur-élévé`.
- Si la liste UA est vide les utilisateurs referencés dans la liste UP affichent l'icône en mode `plat`.

Selon les règles précédentes le *GET* se contente de modifier le mode d'affichage de l'icône d'un document pour ses utilisateurs potentiels (UP) lors de l'occurence d'un des deux événements suivants : le passage de la liste UA de l'état vide à l'état non vide et lors du passage de la même liste de l'état non vide à l'état vide.

3.5.1.2 Le gestionnaire de l'organisation (GOR)

Le gestionnaire de l'organisation (*GOR*) a le rôle de définir et de présenter les utilisateurs qui forment l'organisation cible de l'environnement. La modélisation de l'organisation des utilisateurs repose sur la définition de deux espaces interconnectés appelés *l'espace organisationnel* et *l'espace utilisateurs*.

L'espace organisationnel (respectivement utilisateurs) est composé d'un ensemble de *rôles organisationnels* (respectivement utilisateurs). Un rôle organisationnel (RO) correspond à un groupe social reconnu dans l'organisation cible. Par exemple dans un milieu universitaire nous retrouvons les deux rôles classiques : `étudiant` et `professeur`. Un rôle utilisateurs (RU) désigne un utilisateur humain enregistré dans l'organisation. La définition des rôles est faite par l'administrateur de l'environnement *Colt*. L'administrateur peut définir un nouvel RO par spécialisation d'un ou plusieurs RO existants. C'est le cas par exemple du rôle `ATER` dans l'organisation illustrée à la figure (3.5) qui est défini par spécialisation des trois rôles `étudiant, chercheur` et `enseignant`. Les relations de spécialisation entre les RO définissent une hiérarchie sur l'espace organisationnel. Cette

hiérarchie facilite la gestion des RO en définissant des règles d'héritage entre les RO. Un RO hérite les attributions de ses *ascendants*.

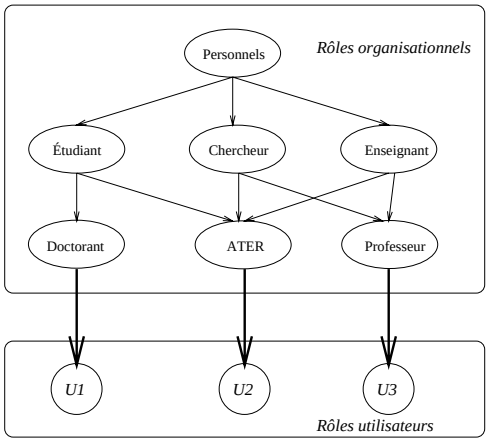


FIG. 3.5 - : Exemple d'un arborescence d'un répertoire des utilisateurs.

Aucune hiérarchie n'est définie sur l'espace utilisateurs. Par contre chaque RU est relié à un ou plusieurs RO qui définissent les groupes auxquels appartient l'utilisateur. Outre la description de la position de l'utilisateur dans l'organisation (à travers les RO qui lui sont associés), un RU porte des informations personnelles destinées à présenter l'utilisateur qu'il représente à ses collègues (nom, photo, coordonnées, etc). La figure (3.6) illustre un exemple de la présentation d'un utilisateur.

Un utilisateur peut demander au *GOR* la liste des utilisateurs ayant un certain RO, ou bien quels sont les rôles attribués à un utilisateur donné.

D'autre part, un utilisateur peut combiner les différents rôles pour *désigner* un nouveau groupe d'utilisateurs. La combinaison des rôles existants ne définit pas un nouveau rôle. Deux opérations de combinaison de rôles sont fournies : l'union de rôles (`ET`) et l'exception (`Sauf`).

La décomposition de l'organisation en deux espaces, organisationnel et utilisateurs augmentée des opérations de combinaison de rôles fournit un modèle de désignation des utilisateurs qui est à la fois efficace (grâce à la manipulation de groupes) et précis car il permet de distinguer les individus (grâce à l'emploi des rôles utilisateurs). La souplesse de notre modèle de désignation prend toute son importance lors du son couplage avec les deux systèmes de gestion de la protection et de gestion des activités coopératives (voir 3.5.1.3 et 3.5.1.4). Le concept du rôle est employé à la fois comme unité de désignation et comme unité de protection.

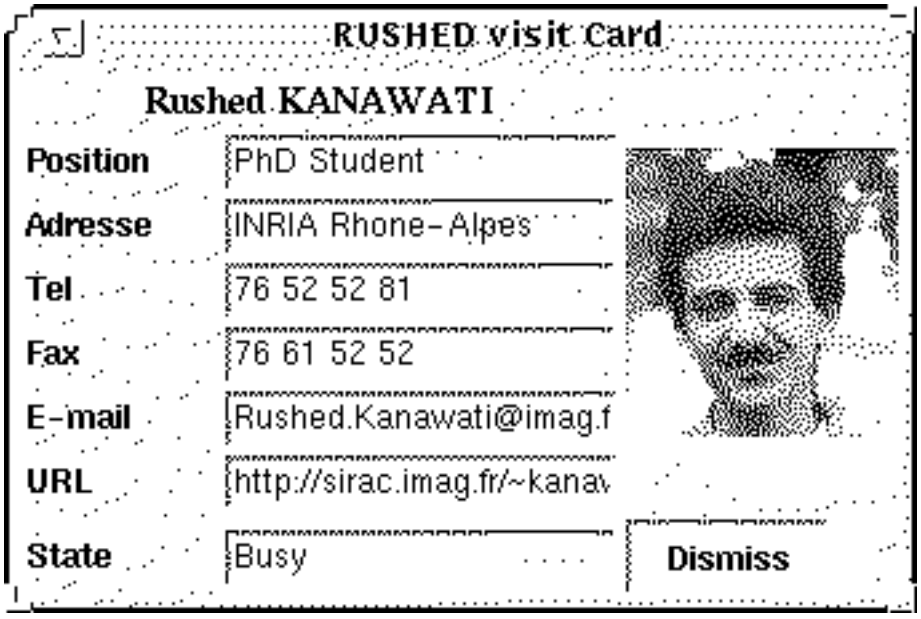


FIG. 3.6 - : Exemple de présentation d'un utilisateur.

3.5.1.3 Le gestionnaire des activités (GAC)

Le gestionnaire des activités (GAC) a la charge de gérer la spécification, la création et l'évolution des activités coopératives qui peuvent avoir lieu dans l'environnement. Selon l'axe du temps, nous distinguons deux types d'activités : les activités *persistantes* et les activités *éphémères*. Une activité est persistante si elle persiste lorsque tous les participants l'ont quitté. Une activité éphémère se détruit avec le départ du dernier participant. La propriété de la persistance permet d'effectuer des tâches dont l'accomplissement nécessite des durées de travail assez longues (plusieurs heures voir plusieurs jours) comme par exemple la rédaction d'un document. Nous détaillons dans la suite les trois fonctions du *GAC*.

1. Spécification d'activités : les modèles d'activités

La spécification d'une activité de groupe revient à définir les actions que chaque participant peut effectuer dans le contexte de l'activité, la nature des interactions entre les participants et les différentes politiques de contrôle qui gouvernent l'évolution de l'activité (politiques d'enregistrement, de participation et de médiation). Une approche générale pour résoudre ce problème consiste à fournir un langage spécifique de description des activités comme c'est proposé dans [Paoli et al.94],[Pons et al.96]. Les utilisateurs finals d'un environnement de coopération n'étant pas des programmeurs, il est difficile de leur apprendre à utiliser un tel dispositif pour spécifier des nouvelles activités de groupe. Une alternative s'impose. Dans *Colt* nous avons opté pour le choix de fournir des *modèles souples*

d'activités de groupe. La construction d'un modèle d'activité est à la charge de l'administrateur de l'environnement qui peut employer un langage évolué pour réaliser cette tâche.

Un *modèle* définit la nature de l'activité (persistante ou éphémère), les politiques de contrôle qui peuvent être appliquées et les actions que les utilisateurs peuvent effectuer dans le contexte de l'activité. La définition des privilèges des utilisateurs repose sur un système d'attribution de *rôles fonctionnels* (RF). Chaque RF est associé à un ensemble de règles d'accès qui déterminent les actions permises pour ce rôle. Nous expliquons la mise en œuvre du système de protection appliqué dans une *pièce* dans la section (3.5.1.4). Un RF particulier que l'on retrouve dans tout modèle est le rôle *président*. Le président a la charge de choisir les politiques de contrôle à appliquer parmi les choix prévus par le modèle de *pièce*. Ce rôle est attribué automatiquement à l'utilisateur qui crée l'activité. Cependant, le président peut *démissionner* en désignant un successeur. Un seul utilisateur peut avoir ce rôle dans la même activité.

Les politiques de contrôle fixées par un modèle d'activité regroupent les stratégies d'enregistrement et d'intégration (accès aux pièces) et les stratégies de gestion de conflits. L'enregistrement des utilisateurs et le déroulement des activités dans une *pièce* doivent vérifier certaines contraintes que nous appelons *les conditions de la validité* de l'activité. La violation d'une condition de validité entraîne la terminaison ou la suspension de l'activité. Par suite, une activité peut être dans l'un des états suivants :

- *Inactive* si tout les participants sont déconnectés.
- *Suspendue* si certains participants sont connectés mais si une ou plusieurs conditions de validité de l'activité ne sont pas respectées.
- *Active* si toutes les conditions de validité sont remplies.

Actuellement nous utilisons un seul type de conditions de validité qui consiste à exprimer des limites supérieures et inférieures sur le nombre des utilisateurs connectés par rôle fonctionnel. En général les conditions de validité servent pour forcer le respect de règles et de code de travail comme par exemple forcer l'application du principe de séparation de pouvoirs où certains rôles fonctionnels ne peuvent pas être attribués à la même personne.

Un aspect important dans la spécification d'activités est la définition de la nature des interactions entre les utilisateurs. Dans les modèles fournis à présent cet aspect reste implicite. Tous les utilisateurs présents en même temps dans l'espace de l'activité interagissent entre eux d'une manière synchrone. L'élaboration des modèles implantant des schémas d'interaction plus sophistiqués reste une perspective.

Trois modèles d'activité sont fournis par le prototype actuel : le modèle d'un *séminaire* (voir tableau 3.3), le modèle d'une *réunion* et le modèle d'une *rencontre*.

Paramètres	Valeur
Nature	Persistante, Éphémère
Rôles fonctionnels	Orateur, Participant, Président
Stratégies d'enregistrement	Libre , Supervisé
Médiation	Par tour de rôle.
Conditions de validité	$\parallel$ Orateur $\parallel = 1$, $\parallel$ Président $\parallel = 1$, $\parallel$ Participant $\parallel \leq 20$
Outil de conversation	Tableau blanc, Bavardage.

TAB. 3.3 - : Modèle d'un séminaire.

1. Création d'une activité

La création d'une nouvelle activité est faite par instanciation d'un des modèles fournis par le système. L'instanciation d'un modèle résulte en la création d'une entité logicielle dite *contrôleur d'activité* (CA). En plus des paramètres définis par le modèle à partir duquel un CA est instancié, ce dernier comprend les informations suivantes :

- *l'en-tête de l'activité* est une chaîne de caractères qui décrit le but de l'activité. L'en-tête est rédigé par l'utilisateur qui crée l'activité. Il est visible pour l'ensemble des participants effectifs et potentiels de l'activité. En consultant l'en-tête, un participant potentiel (invité) peut avoir une idée sur l'activité et en conséquence être en mesure de décider s'il veut participer ou non à l'activité en question. Il est à noter que l'information contenue dans l'en-tête ne peut pas être traitée par le système.

- *La distribution des rôles fonctionnels.* Il s'agit d'associer à chaque rôle fonctionnel prévu dans le contexte de l'activité un groupe d'utilisateurs. La désignation des utilisateurs qui peuvent jouer un rôle fonctionnel se fait en associant à ce rôle une combinaison de rôles utilisateurs et organisationnels (voir 3.5.1.3). Cette association tient lieu d'invitation à participer à activité ; chaque utilisateur désigné directement (par son rôle utilisateur) ou indirectement (par un rôle organisationnel) est averti de la création de l'activité. Un utilisateur peut avoir plusieurs rôles fonctionnels à la fois. Nous montrons dans la section (3.5.1.4) comment résoudre les éventuels conflits entre les rôles.

- La liste des utilisateurs enregistrés et la liste des participants actuels.

- La liste des documents appartenant à l'activité.

3. Contrôle de l'évolution d'une activité

À chaque activité est attribué un identificateur unique qui la distingue des autres activités. Toutes les activités sont enregistrées dans une structure de données spécifique gérée par le *GOR*. Toute nouvelle activité commence par entrer dans l'état inactif. Dans cet état les utilisateurs potentiels, étant informés de l'existence de l'activité peuvent envoyer des requêtes d'enregistrement. Le traitement de ces requêtes est fait selon la politique fixée par le président de l'activité. Une activité sort de l'état inactif lorsque l'un des participants enregistrés envoie au *GOR* une requête d'intégration à cette activité. Si la requête d'intégration est acceptée le contrôleur de l'activité en question examine si toutes les conditions de validité de l'activité sont satisfaites. Le cas échéant, l'activité passe à l'état actif. Sinon elle passe à l'état suspendu. L'utilisateur qui a demandé l'intégration est mis en attente jusqu'à ce que l'activité passe à l'état actif. Il peut cependant retirer sa demande d'intégration (quitter l'activité).

Pendant le déroulement de l'activité (dans l'état actif) les participants communiquent via les outils d'aide à la conversation disponibles dans l'activité. Un participant peut apporter (par création ou par importation) de nouveaux documents à l'activité. Les documents sont manipulés par les outils de production. Les outils d'aide à la conversation étant des outils conçus spécialement pour les groupes, ils offrent leurs propres politiques de gestion de droit de parole. Par exemple, l'emploi de l'outil de bavardage est libre ; tout utilisateur peut envoyer un message texte à tout autre participant sans qu'il ait le droit de parole dans l'activité. Le choix de séparer le contrôle des outils de production de ceux d'aide à la conversation est justifié par la différence fonctionnelle entre les deux types d'outils.

3.5.1.4 Le gestionnaire de protection (GDP)

Le principal but d'un système de protection est d'empêcher l'accès et la manipulation des données par des utilisateurs non autorisés. Une première phase dans le processus de contrôle d'accès consiste à authentifier les utilisateurs. L'authentification repose sur la procédure classique de présentation de mots de passe.

Comme nous l'avons évoqué dans la section (3.3), le *Terrain* est composé d'un espace partagé et d'un ensemble de *pièces*. Par conséquent un utilisateur ne peut être que dans l'une des deux situations suivantes: 1) dans le *Terrain*, ou 2) dans une *pièce*. Les mécanismes d'expression, d'administration et d'évaluation de droits d'accès sont similaires dans les deux situations. Dans le suivant nous présentons le système de protection du *Terrain*. Ensuite nous citons les points de différence entre ce système et celui de la protection d'une *pièce*.

Protection du Terrain

Expression de droits d'accès

Nous utilisons le modèle d'organisation présenté dans (3.5.1.2) comme un support d'attribution et d'expression de droits d'accès au *Terrain*. Chaque rôle, utilisateur ou organisationnel, est associé à un ensemble de règles de protection. Une règle de protection, r a la forme suivante :

$$r = [\pm, A, O]$$

Le signe au début d'une règle détermine si le droit A donné sur le(s) document(s) O est positif ou négatif. L'emploi de droits négatifs facilite d'attribution et la révocation sélectives des droits. Par exemple pour donner à un rôle R le droit de lire tous les documents d'un groupe G sauf pour un document particulier $d \in G$ il suffit de lui donner le droit de lecture sur le groupe G et de lui interdire le lecture du d (lui attribuer un droit négatif de lecture sur d).

Un droit A est exprimé en fonction des cinq opérations de base suivantes :

1. *Administrer* (A) donne la capacité de modifier le profil de protection des objets désignés. Ce droit peut être attribué sur des groupes de documents ou sur des documents individuels.

2. *Importer* (I) donne la capacité d'introduire un nouveau document dans le groupe de documents désigné. Naturellement ce droit s'applique uniquement aux groupes de documents.

3. *Exporter* (E) donne la capacité d'extraire un document du groupe désigné.

4. *Éditer* (W) donne la capacité de modifier (y compris effacer) le document ou le groupe de documents désigné.

5. *Lire* (L) donne la capacité de lire le document ou le groupe de documents désignés.

Pour faciliter l'attribution des droits, des règles implicites d'inférence sont définies sur les droits définis ci-dessus. Les règles d'inférence sont les suivantes : $(A \rightarrow I, E.)$, $(E, I \rightarrow W.)$, $(W \rightarrow L.)$, $(E \rightarrow L.)$.

L'interprétation des règles précédentes est directe. Par exemple la première règle implique qu'en attribuant le droit d'administration d'un groupe de documents à un rôle on lui attribue implicitement les droits d'importation et d'exportation à ce même groupe. Ces règles d'inférence sont utilisées par la fonction d'évaluation de droits d'accès que nous exposons plus loin dans cette même section.

Notre système d'expression de droits d'accès permet de définir des règles de protection qui sont synthétiques et précises. Par exemple pour interdire aux *étudiants* de modifier les données des *enseignants* il suffit d'ajouter au rôle organisationnel *étudiant* la règle suivante : (-, W, Enseignant). Cette règle est valide pour tout *étudiant* qu'il soit déclaré avant ou après l'introduction de cette règle. En même temps nous pouvons exprimer des exceptions ; par exemple permettre à un `étudiant` privilégié de modifier les documents d'un enseignant ou un document particulier en ajoutant à son rôle utilisateur une règle de type (+ , W, Doc) où Doc est le document en question.

L'attribution des rôles et la définition des règles au niveau de chaque rôle utilisateur ou organisationnel est de la responsabilité de l'administrateur de *Colt*. Cependant un utilisateur a la possibilité de *partager* avec d'autres les privilèges qui lui sont attribués. Deux principales fonctions sont fournies à cet effet :

1. La fonction de délégation de droits permet à un utilisateur de déléguer à un autre certains de ses propres privilèges. La fonction de délégation prend la forme suivante : `Déléguer (RU, R, +A, D)`, qui veut dire que l'utilisateur dont le rôle utilisateur est RU délègue au rôle R le droit A sur l'objet D. Le rôle R peut être un rôle utilisateur ou organisationnel. Dans le cas où R est un rôle organisationnel la nouvelle règle de protection (le droit délégué) est ajoutée à tous les rôles utilisateurs qui sont associés au rôle organisationnel désigné et pas au niveau du rôle R lui même. Un utilisateur ne délègue que des droits positifs. Évidement il ne peut pas déléguer des droits qu'il ne possède pas. Un utilisateur qui possède un droit par voie de délégation ne peut pas le déléguer à d'autres utilisateurs. Les droits déléguées peuvent être révoqués en appliquant l'opération inverse `Révoquer` $(RU, R, +A, D)$.

2. La fonction d'attribution de droits permet à un utilisateur d'attribuer à d'autres des droits sur des objets sur lesquels il a le droit *d'administration*. Cette fonction prend la forme suivante `Attribuer(RU, R, ± A , D)` qui veut dire que l'utilisateur RU attribue au rôle R le droit A sur l'objet D. Le droit attribué peut être positif ou négatif. Il est acquis *définitivement* par les utilisateurs ayant le rôle R. Ceci veut dire que l'utilisateur récepteur peut attribuer ou déléguer le droit reçu à d'autres utilisateurs. La fonction d'attribution n'est valide que si l'utilisateur émetteur possède le droit d'administration sur les objets désignés. D'autre part, dans le cas d'attribution d'un droit négatif, il ne faut pas que l'utilisateur récepteur soit un des *propriétaires* des objets désignés. Par exemple, un utilisateur qui a le rôle *étudiant* ne peut pas attribuer un droit négatif sur un document du groupe *étudiant* à un autre utilisateur qui a le même rôle organisationnel.

Toute modification de droits d'accès d'un utilisateur prend effet **immédiatement**. Le gestionnaire de protection prévient l'agent colt de l'utilisateur concerné pour qu'il agisse en conformité avec les nouvelles règles qui lui sont attribuées.

Par exemple si on interdit à un utilisateur de lire un document, l'agent colt de cet utilisateur va être averti pour qu'il retire le document en question de la vue associé à l'utilisateur. Si cet utilisateur est en train de lire ce document l'*AgC* informe l'application à travers laquelle le document est manipulé pour qu'elle *quitte* ce document. Une explication est donnée à l'utilisateur sous forme de boite de dialogue.

Évaluation des droits d'accès La fonction d'évaluation de droits d'accès est définie comme suit :

$$Eval : U \times A \times O \rightarrow (False, True)$$

où U est l'utilisateur qui demande d'appliquer l'opération A sur le document O. La fonction *Eval* renvoie 1 si la requête d'accès est accordée, 0 sinon. À Chaque utilisateur est associé un graphe, dit *graphe de privilèges*, qui est composé de son rôle utilisateur et des rôles organisationnels qui lui sont associés. La construction du graphe de privilèges est faite en remontant les liens entre le rôle utilisateur et la hiérarchie organisationnelle. La figure (3.7) illustre le graphe de privilèges de l'utilisateur U_2 dans l'organisation illustrée à la figure (3.5).

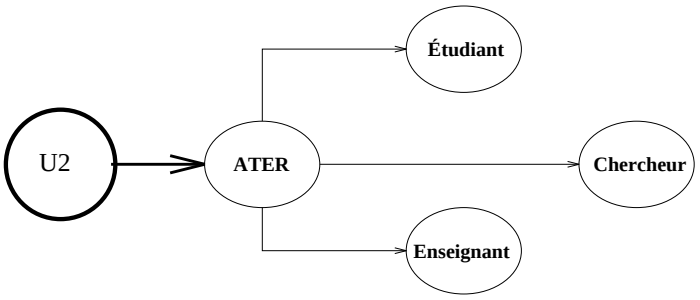


FIG. 3.7 - : Graphe de privilèges associé à l'utilisateur U_2 de l'organisation illustrée à la figure (3.5)

Pour évaluer une requête d'accès, la fonction *Eval* applique un simple mécanisme d'*inférence* sur les règles de protection portées par le graphe de privilèges de l'utilisateur qui demande l'accès. L'évaluation d'une requête en fonction d'une règle de protection peut donner l'un des trois résultats suivants : `Acceptée, Refusée, Non Décidée`. La procédure de l'évaluation est la suivante. Étant donnée une requête d'accès $Q = (a, d)$ dont le but est d'appliquer l'opération a sur le document ou le groupe de documents d et étant donné une règle de protection de la forme $R = (Signe, A, D)$, le résultat de l'évaluation est la suivante :

1. La requête *est acceptée* si $d \in D$ **et** $A \rightarrow a$ **et** Signe == +.
2. La requête est *refusée* si ($d \in D$ **et** $a \rightarrow A$ **et** Signe == -).

3. L'évaluation donne le résultat `Non Décidé` dans les autres cas.

La fonction *Eval* examine chacune des règles contenues dans le graphe de privilèges jusqu'à l'obtention d'une acceptation ou d'un refus explicite de la requête. Si toutes les règles sont épuisées sans avoir une réponse explicite (acceptée ou refusée) la requête est refusée. Évidement, le résultat de la fonction *Eval* dépend de l'ordre de l'exploration du graphe de privilèges et de l'ordre de l'examen des règles de protection dans chacun des rôles. Pour éviter tout ambiguïté d'interprétation, nous fixons les règles d'évaluation suivantes :

- Dans un rôle, la règle qui s'applique au groupe d'objets le plus spécifique contenant l'objet désigné par la requête est la règle prioritaire. Ainsi la règle qui désigne un document par son identificateur est prioritaire sur les règles qui désignent le groupe auquel appartient le document.

- Le graphe de privilège est exploré en profondeur d'abord. Ceci donne la priorité au rôle le plus à droit dans le graphe. Ainsi dans l'exemple illustré à la figure (3.7) le graphe est exploré dans l'ordre suivant : U_2 , ATER, Étudiant, Chercheur, Enseignant.

Nous faisons l'hypothèse ici que les rôles sont *cohérents* dans le sens qu'ils ne contientent pas de règles de protection contradictoires.

Exemples d'évaluation des requêtes d'accès

Pour illustrer la procédure de l'évaluation d'une requête d'accès nous prenons le cas de l'utilisateur U_2 dont le graphe de privilèges est illustré à la figure (3.7). Le tableau (3.4) donne les règles de protection contenues dans chacun des rôles faisant partie du graphe précèdent. Dans la spécification des règles nous utilisons la notation d^i_G pour désigner un document d^i appartenant au groupe G. Par exemple d^1_{ATER} et d^2_{ATER} sont deux documents appartenant au groupe *ATER*.

Rôle	**Règles**
U_2	R_1 : (+, A , U_2), R_2 : (-, L, $d^1_{Enseignant}$).; R_3 : (+, W, $d^2_{Chercheur}$)
ATER	R_4 : (+, A, ATER). R_5 : (+, I, Étudiant). R_6 : (+, I, Enseignant)
Étudiant	R_7 : (+, A, Étudiant)
Chercheur	R_8 : (+, A, Chercheur)
Enseignant	R_9 : (+, A, Enseignant)

TAB. 3.4 - : Exemples de règles de protection.

Dans la suite nous énumérons quelques exemples de requêtes d'accès avec leur résultats d'évaluation selon la procédure décrite ci-dessus.

- $Q_1 = (W, d^1_{U_2})$ est autorisée grâce à la règle R_1.
- $Q_2 = (L, d^2_{Enseignant})$ est autorisé car c'est la règle R_7 qui s'applique.
- $Q_3 = (L, d^1_{Enseignant})$ est refusée car c'est la règle R_2 qui s'applique.
- $Q_4 = (L, d^1_{Chercheur})$ est autorisée car c'est la règle R_3 qui s'applique.
- $Q_5 = (W, d^1_{Doctorant})$ est refusée car il y aucune règle qui l'autorise.

La protection d'une pièce Le mécanisme de protection est semblable au mécanisme de protection du *Terrain*; il suffit de remplacer les rôles organisationnels par les rôles fonctionnels définis dans le contexte de la *pièce*. Quatre différences principales néanmoins existent entre les deux mécanismes de protection :

1. L'ensemble des rôles fonctionnels a une structure plate ce qui simplifie la fonction de l'évaluation des requêtes d'accès.
2. Les droits sur les objets sont exprimés en fonction d'opérations qu'on peut appliquer sur les données au lieu de simples opérations d'administration de lecture et d'écriture des documents. Par exemple dans une *pièce* où les utilisateurs élaborent ensemble un dessin, il peut y avoir besoin de spécifier qu'un rôle fonctionnel a le potentiel de changer les couleurs des objets sans avoir le droit de les effacer. Pourtant les deux opérations de changement de couleur et d'effacement d'objets sont des opérations d'écriture. L'emploi de droits à grains fins est dicté par le fait que dans une *pièce* les utilisateurs sont plus *proches* les un des autres. Ils ont, par conséquent, besoin d'une grille plus fine pour définir leurs marges de manœuvre.
3. Le champ d'objet dans une règle de protection désigne soit un document de coopération appartenant à l'activité courante soit tous les documents existant dans la pièce.
4. Le *président* de l'activité est le seul propriétaire de tous les documents manipulés dans la *pièce*. Par conséquent il est le seul à pouvoir changer les droits des autres participants sur les documents contenus dans l'activité qu'il préside.

3.5.1.5 Le gestionnaire de la communication (GCM)

Le gestionnaire de la communication fournit des mécanismes de transfert de données entre les différents composants actifs (gestionnaires et agents) de l'environnement *Colt*. Le *GCM* fonctionne comme un *courtier de messages*. Les différents composants lui adressent des messages dont les destinataires sont spécifiés

par leurs identificateurs ou par un critère associatif. L'emploi de critères associatifs est restreint à la désignation des agents colts (AgC). À la réception d'un message, le *GCM* fait la résolution des adresses des destinataires et s'occupe de l'acheminement du message. Pour effectuer la résolution des adresses le *GCM* maintient à jour une liste des adresses de toutes les entités actives dans l'environnement. Trois types de critères de désignation des agents peuvent être employés :

- Les critères fondés sur la combinaison des rôles organisationnels et utilisateurs. Par exemple envoyer un message à tous les utilisateurs qu'ont le rôle organisationnel *Étudiant*. C'est le cas typique d'un message envoyé par le gestionnaire des activités pour informer les agents connectés au *Terrain* de la création d'une nouvelle activité.

- Les critères fondés sur la possession d'un privilège. Par exemple l'envoi d'un message à tous les agents connectés qui ont le droit de lire un document donné. C'est le cas typique d'un message envoyé par le gestionnaire de l'espace de travail pour informer les utilisateurs de l'ouverture ou de la fermeture d'un document.

- Les critères fondés sur l'appartenance à une activité donnée. C'est le cas de la diffusion d'un message à tous les participants à une activité.

Le *GCM* coopère avec les autres gestionnaires notamment avec le *GET*, le *GAC* et le *GDP* pour trouver la liste des identificateurs des agents désignés à partir d'un critère associatif, puis il se réfère à sa propre liste pour la résolution des adresses des destinataires.

Le *GCM* supporte donc les deux paradigmes de communication : point à point et communication de groupe. Un problème important que nous n'avons pas abordé est le problème de l'ordonnancement des messages diffusés à un groupe de composants. Pour l'instant, nous fournissons un support minimal d'un service de communication de groupe, à savoir la communication fiable à ordre FIFO.

Outre l'acheminement de messages, le *GCM* offre aussi des mécanismes pour le téléchargement d'informations plus complexes telles que des documents. En fait, il implante un protocole de transfert de fichier (style ftp) qui permet d'échanger des documents entre le *Terrain* et les agents colts.

3.5.2 L'agent-colt (AgC)

Un agent colt est une entité logicielle qui représente un utilisateur enregistré dans l'environnement. Il est chargé de 1) présenter et gérer la vue attribuée à l'utilisateur et 2) fournir et contrôler l'emploi des outils de production. Les deux fonctions d'un *AgC* sont détaillées ci-après.

3.5.2.1 Présentation et gestion des vues

À chaque utilisateur est attribué une *vue* sur le *Terrain* qui comprend les informations suivantes :

1. La liste des documents accessibles pour l'utilisateur. Chaque document pour lequel l'utilisateur a au moins le droit de lecture fait partie de cette liste.

2. Les activités dans lesquelles l'utilisateur est enregistré.

3. Les activités auxquelles l'utilisateur est invité.

Les documents sont représentés sur l'interface utilisateur sous forme icônique (voir 3.4) tandis que les activités sont représentées sous forme de deux listes : la liste des activités courantes et la liste des invitations. Par suite pour visualiser la vue attribuée à l'utilisateur l'*AgC* maintient les structures des données suivantes :

- Une liste des documents contenus dans la vue. Une entrée dans cette liste contient les informations suivantes : le nom utilisateur du document, le propriétaire du document, l'application de manipulation du document (pour afficher l'icône adéquat) et l'état d'utilisation du document (détermine le mode d'affichage de son icône).

- Deux listes d'activités ; l'une contient les activités dans lesquelles l'utilisateur est enregistré et l'autre contient les activités auxquelles l'utilisateur est invité. Une entrée dans une liste d'activité contient les informations suivantes : l'identificateur de l'activité, son en-tête et son état actuel.

La vue d'un utilisateur change d'une manière dynamique en fonction de création et de destruction de documents et d'activités dans l'environnement et en fonction de changement de droits (par délégation, révocation ou attribution) de l'utilisateur. Toute modification apportée à la vue d'un utilisateur connecté au *Terrain* est immédiatement notifiée à l'*AgC* de cet utilisateur. La version initiale de la vue d'un utilisateur est *calculée* lors de son connexion au *Terrain*. La procédure de la connexion d'un utilisateur au *Terrain* est schématisée sur la figure (3.8).

3.5.2.2 Définition et gestion d'un outil de production

Un outil de production (OP) est composé des trois parties suivantes :

La présentation gère les interactions de l'utilisateur avec l'outil.

Le noyau fonctionnel implante les fonctions fournies par l'outil et gère les données associées à l'outil.

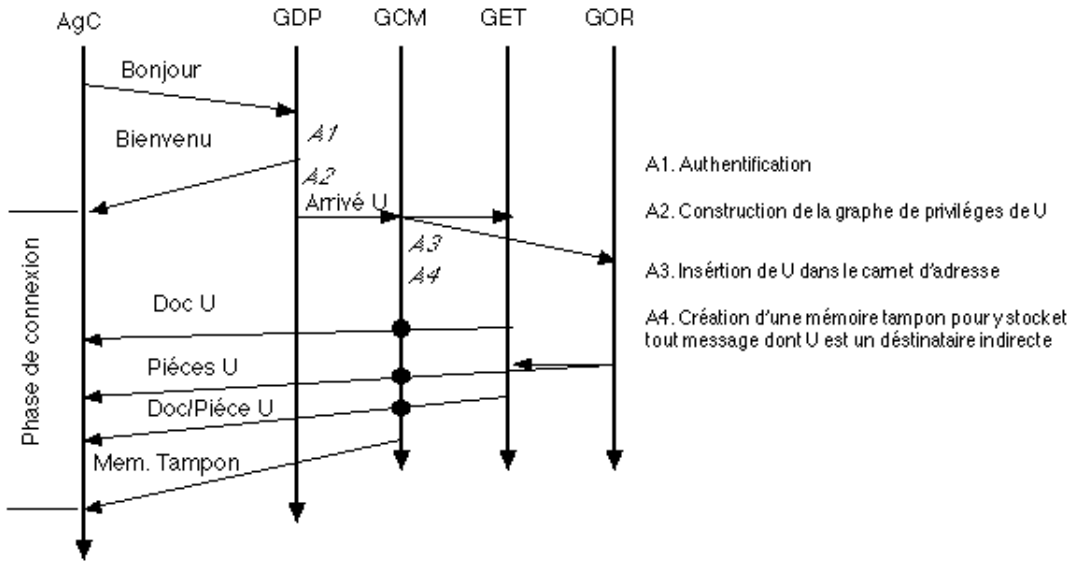


FIG. 3.8 - : Protocole de la connexion d'un AgC au Terrain.

L'adaptateur est l'unité de contrôle. Il *conditionne* le comportement de l'outil en fonction de son contexte d'utilisation et en fonction des droits que possède l'utilisateur sur le document manipulé par l'outil.

La figure (3.9) montre la disposition des trois parties constituant un OP et les interactions entre elles. L'*adaptateur* contrôle le comportement de l'outil en fonction des instructions qu'il reçoit de la part de l'*AgC*. Ce dernier possède les connaissances nécessaires pour préciser le contexte de l'emploi d'un outil et par suite son comportement. À cet effet, l'*AgC* maintient une structure de données spéciale qui contient pour chaque document employé par l'utilisateur local les informations suivantes : 1) les droits de l'utilisateur sur le document et 2) l'état d'utilisation de ce document (individuel ou en groupe). Les informations précédentes sont obtenues lors de l'accès au document en question.

L'adaptateur traduit les droits attribués à l'utilisateur sur un document en inhibant les commandes non permises (griser les commandes non autorisées). Dans le cas où le document est manipulé par un groupe (dans une rencontre ou dans une pièce), l'adaptateur se charge de contrôler l'interaction de l'utilisateur avec les documents courants en fonction de la politique de gestion de droit de parole employée par le groupe. Par exemple dans le cas de l'utilisation d'une politique de tour de rôle (ou par circulation d'un jeton) pour la gestion de droit de parole, l'*AgC* informe l'adaptateur s'il possède le *jeton* ou non. Si l'*AgC* possède le jeton l'adaptateur débloque l'outil et permet ainsi à l'utilisateur de modifier le document. Lorsqu'un utilisateur applique une commande sur le document, le noyau fonctionnel envoie la commande appliquée à l'adapatateur qui se charge de l'envoyer aux autres sites participant à l'activité. L'adaptateur traduit la commande

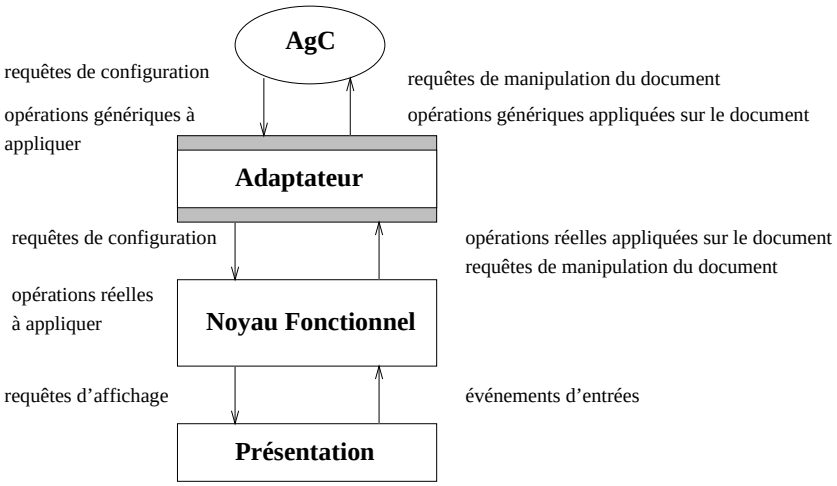


FIG. 3.9 - : Structure d'un outil de production dans l'environnement Colt.

en termes d'*opérations génériques*. Les opérations génériques sont les mêmes que celles employées pour spécifier les privilèges attribués aux rôles fonctionnels. À la réception, l'adaptateur de l'outil concerné par la commande traduit les opérations génériques en opérations réelles que le noyau fonctionnel peut exécuter. Le passage par des opérations génériques permet d'employer des outils différents pour manipuler le même document. Par exemple lorsqu'un utilisateur modifie à travers un éditeur de texte un document représentant une page WEB qu'un autre utilisateur est en train de lire via un *browser* (les deux utilisateurs sont donc dans la même pièce) l'adaptateur du *browser* traduit l'opération en opération de rechargement de la page lue. Le principe d'échange d'opérations génériques peut être vu comme une généralisation de la coopération par des outils hétérogènes via l'implantation des protocoles inter-applications comme le propose [Karsenty et al.93b]. Au lieu d'implanter un protocole spécialisé par pair d'outils susceptibles d'être utilisés pour manipuler un document, nous implantons un protocole par outil qui traduit les opérations de cet outil en opérations génériques.

L'environnement *Colt* intègre actuellement deux outils de production : un éditeur graphique nommé *ColtDraw* et un *browser* d'images nommé *ColtSlideShow*. Les deux outils sont développés en modifiant des applications de démonstration jointes à la distribution de la version 4.0 de *Tk*. Noter que le schéma de construction d'OP décrit ci-dessus nous permet aussi de construire de nouveaux OP par intégration d'applications existantes au niveau de l'application comme nous l'avons expliqué dans (3.3.3).

3.6 Réalisation

3.6.1 L'environnement de développement

Un premier prototype de l'environnement *Colt* a été développé sur un ensemble de stations *Sun Sparc* utilisant le système d'exploitation *SunOs 4*. Les développements sont réalisés en utilisant le langage de programmation *Tcl 7.0* et l'environnement graphique *Tk 4.0* [Welch95]. Le choix de *Tcl-Tk* est motivé par les arguments suivants :

- *Tcl* est un langage interprété. Il favorise à ce titre le prototypage et le test rapide des applications. Il permet aussi l'évolution facile (l'ouverture) et le changement dynamique de comportement des applications.
- L'environnement graphique *Tk* offre un large éventail de *widgets* graphiques nécessaires pour le développement des interfaces utilisateurs pour les applications interactives (fenêtres, boutons, listes et menus, etc). Les *widgets* fournies sont très adaptables. De plus *Tk* est un environnement extensible. Il est facile de créer de nouveaux *widgets* graphiques ou d'adapter le comportement des *widgets* existants.
- L'environnement *Tcl-Tk* est disponible pour une grande variété de plates-formes telle que *Unix*, *Linux*, *MacOS* et *MS-Windows*. Par conséquent *Colt* sera facilement portable à pour autres types de plates-formes.
- Il existe plusieurs interfaces du langage *Tcl* avec d'autres langages évolués notamment le *C*. L'existence d'une interface simple avec des langages compilés remédie au défaut principal des langages interprétés à savoir leurs performances en termes de temps d'exécution. Il est facile de remplacer les fonctions *Tcl* qui représentent le goulot d'etranglement de l'application par des fonctions *C* alliant ainsi la performance à la simplicité de la programmation.
- Un dernier argument mais pas le moins important est que l'environnement *Tcl-Tk* est disponible dans le domaine public. Il bénéficie du support d'une communauté d'utilisateurs particulièrement active qui accroît sans cesse le nombre d'applications et d'extensions de cet environnement et qui fournit un support de documentation *On-line* à travers un groupe de *News* très sollicité[3] et plusieurs serveurs *WEB*.

3.6.2 Mise en œuvre

La mise en œuvre de l'environnement *Colt* a été fait de la manière suivante. Un site central exécute un processus *unix* qui implante les fonctions du *Terrain* et

[3] comp.lang.tcl

chaque site utilisateur exécute un processus qui implante les fonctions de l'*AgC*. La communication entre les processus est faite en utilisant les *sockets* unix en mode connecté (en utilisant le protocole TCP/IP) [Comer92]. *TCP/IP* garantit la fiabilité de la communication et la réception des messages en ordre *FIFO*. La diffusion d'un message à un groupe de N processus est faite en maintenant N connexions *TCP/IP* entre l'émetteur et les récepteurs.

La description de la configuration de l'environnement (les utilisateurs, l'espace du travail, les règles de protection,..., etc) est conservé dans de fichiers textes maintenus sur le site central. Chaque modification apportée à la configuration de l'environnement est immédiatement enregistrée dans le fichier adéquat. Un document *Colt* correspond à un ou plusieurs fichiers unix (selon le type de l'application employée pour la création et la manipulation du document). Par exemple un document créé par l'application *ColtDraw* est un fichier texte qui contient la liste des instructions *Tcl* nécessaires pour redessiner les objets sauvegardés dans le document, tandis qu'un document généré par l'application *ColtSlideShow* correspond à plusieurs fichiers *unix*: un fichier par image contenue dans le document, un fichier contenant les commentaires sur les images et un autre qui contient la liste des fichiers unix précédents constituant le document.

Le processus *Terrain* crée lors de son lancement une *socket* d'écoute lié à un port de communication dont le numéro est connu par tous les agents de l'environnement. Le paire constitué par le numéro de port et l'adresse du site central identifie d'une manière unique l'adresse du *Terrain*. Pour accéder au *Terrain* un utilisateur exécute sur son site local un processus *AgC*. Ce dernier commence par afficher sur l'écran de l'utilisateur une boîte de dialogue qui demande à l'utilisateur d'entrer son nom et son mot de passe. Après l'introduction des informations demandées, l'agent établit un canal *TCP/IP* avec le *Terrain* à travers lequel il envoie la requête de connexion. Si la requête de connexion est valide le *Terrain* envoie à l'agent colt la vue attribuée au nouvel arrivant selon le protocole de connexion décrit dans (3.5.2). L'*AgC* se charge de la présentation graphique de la vue attribuée à l'utilisateur (voir figure 3.1). Le canal de communication est maintenu tout au long de la connexion de l'*AgC* au *Terrain*. Il sert à véhiculer les messages entre les deux entités.

L'activation d'un outil correspond à l'exécution d'un autre processus Unix qui implante l'outil. L'*AgC* communique avec les outils qu'il emploie via des canaux de communication *TCP/IP*. Ces canaux servent pour véhiculer les informations entre les adaptateurs des outils et l'*AgC*. Une seule instance d'un outil par site utilisateur peut être activé à la fois.

Chaque activité de groupe est contrôlée par une entité spécifique dite le contrôleur de l'activité. Le passage d'une activité de l'état inactif à l'état suspendu ou actif se traduit par le lancement d'un nouveau processus unix sur le site du *Terrain*. Ce dernier processus établit des canaux TCP/IP avec le *Terrain* et avec chaque *AgC* représentant un utilisateur intégré dans l'activité. La figure (3.10) illustre le schéma d'implantation du prototype actuel.

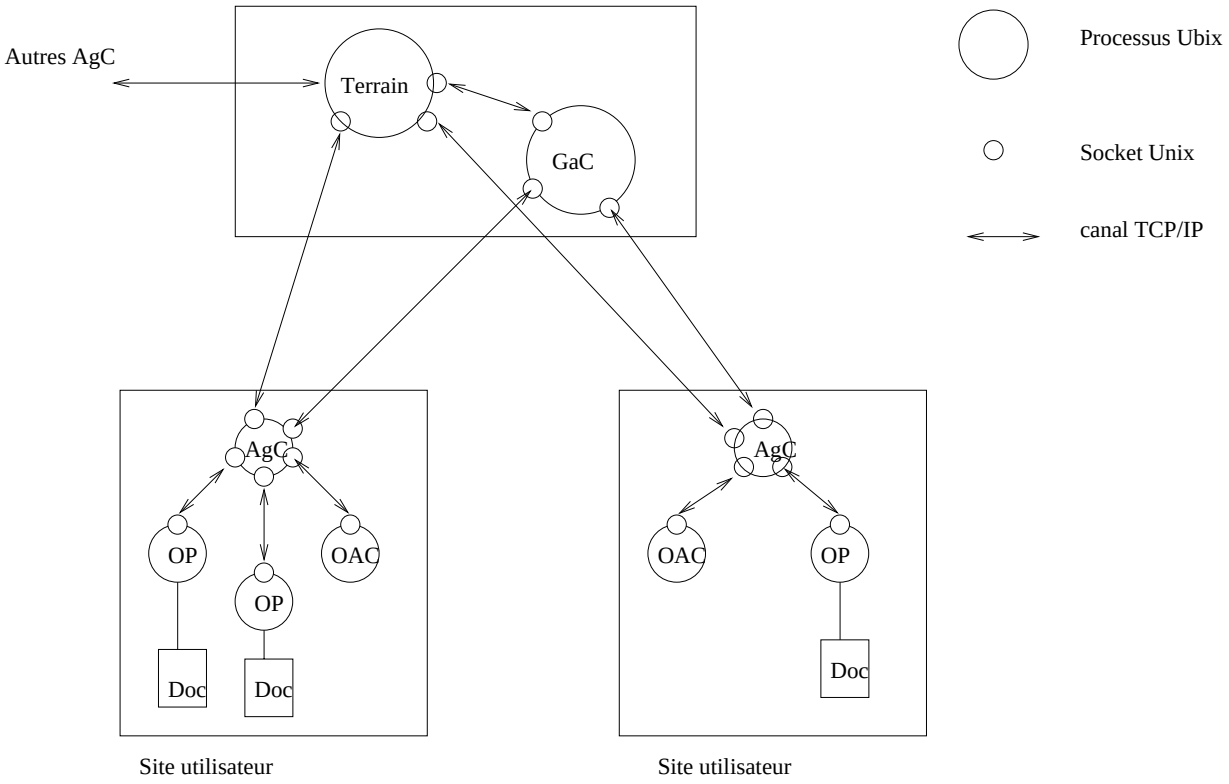


FIG. 3.10 - : Implantation de l'environnement Colt.

3.7 Discussion et comparaison

Dans un premier nous présentons une brève comparaison entre l'environnement *Colt* et deux autres *EdC*. Le premier, *EVC* [Benford et al.95] est fondé sur la métaphore du *terrain* tandis que le deuxième, *TeamRooms* [Roseman et al.96a] qui repose sur la métaphore de la *pièce*. Ensuite nous présentons une comparaison plus générale entre notre approches et les différentes plates-formes présentées dans la section (2.5).

3.7.1 Environnements virtuels coopératifs

Un environnement virtuel coopératif (EVC) marie les deux domaines de la *réalité virtuelle* et du *TCAO* [Benford et al.95]. Le principe est de doter un système d'information (base de données, système de fichiers) d'une représentation visuelle abstraite qui englobe aussi les utilisateurs eux mêmes. Une représentation est dite abstraite si elle ne repose pas sur une métaphore de la réalité physique. Plusieurs approches de représentations abstraites sont proposées dans la littérature. La représentation proposée dans [Benford et al.95] est fondée sur une interprétation graphique des propriétés extrinsèques et intrinsèques des objets

visualisés. D'autres approches sont l'approche statistique et la structure hypertexte. Chaque représentation des données et des utilisateurs est augmentée d'une méthode de navigation et de déplacement qui permet aux utilisateurs de se mouvoir dans l'espace. Chaque utilisateur a une vision de l'espace partagé qui dépend de sa position dans l'espace. Il *voit* aussi les autres utilisateurs qui sont dans son champ de vision. Si deux utilisateurs accèdent en même temps au même objet, des liens de communication directes entre eux s'établissent directement permettant ainsi leur coopération (synchrone). Les utilisateurs peuvent communiquer directement entre eux en clinquant tout simplement sur l'objet représentant l'autre utilisateur. Ainsi un *EVC* est un système hybride qui regroupe des fonctions des systèmes de coopération à immersion et des systèmes de coopération contractuels (2.2.4). Cependant les *EVC* actuels ne supportent pas totalement la coopération contractuelle. Ils implantent un modèle de rendez-vous implicite. Les activités coopératives dans un *EVC* sont les analogues des rencontres supportées par *Colt*. Les *pièces* n'existent pas dans un *EVC*. Un autre défaut des *EVC* est de négliger les aspects organisationnels du travail coopératif. Tous les utilisateurs partagent tout l'espace d'information (pas de rôles organisationnels). En fait la prise en compte de l'aspect organisationnel pose un problème intéressant au niveau de la représentation de l'espace et des utilisateurs : Un utilisateur U_y voit-il un autre U_x si ce dernier manipule un objet O_i proche d'un objet O_j si l'objet O_j figure dans la vue de U_y mais pas l'objet O_i ? Un autre problème est celui de l'implantation des méthodes de navigation : la représentation spatiale des données impose t-elle des contraintes sur les *déplacements* des utilisateurs dans l'espace? Par exemple dans le cas d'une représentation de type hypertexte faut-il suivre les liens ? ou l'utilisateur peut-il sauter d'un objet à un autre sans respecter la topologie de l'espace? Un dernier problème posé par le concept d'*EVC* est l'absence d'un espace privé et l'immersion des utilisateurs dans l'espace partagé? Des mécanismes de protection de la *vie privée* doivent être fourni sinon le système ne peut pas être utilisable [Salber95].

Les différences majeures entre notre approche dans *Colt* et celle du *EVC* sont les suivantes : 1) l'absence d'une représentation explicite des utilisateurs et l'absence d'une présentation spatiale de l'espace partagé (le PIT) ce qui nous évite de traiter les problèmes de la protection de la vie privée et de la navigation dans l'espace. 2) les *EVC* ne supportent pas la coopération organisée. Le concept de *pièce* n'étant pas supporté, aucune structure de contrôle de la coopération n'est mis en place outre l'établissement des communications directes lorsque deux utilisateurs accèdent au même objet partagé.

3.7.2 TeamRooms

TeamRooms est un environnement de coopération fondé sur la métaphore de la *pièce* [Roseman et al.96b]. Pour coopérer les utilisateurs se connectent à un serveur central qui maintient une liste des *pièces* disponibles dans l'environne-

ment. Le concept de *pièce* est semblable à celui employé dans *Colt*. Les pièces sont persistantes. Chaque *pièce* contient un ensemble d'outils d'aide à la coopération (outil de bavardage, tableau blanc partagé) et des outils de production et offre un ensemble de mécanismes de présentation de la co-présence (pointeur partagé, liste des utilisateurs présents, ..., etc). La différence principale entre une *pièce Colt* et une *pièce TeamRooms* est que les outils de production dans *TeamRooms* sont des collecticiels ; ils sont construits à l'aide de la plate-forme *GroupKit* [Roseman et al.92] développée par la même équipe de recherche qui a développé *TeamRooms*.

Chaque outil de production (collecticiel) implante son propre modèle de contrôle (protocoles de contrôle et rôles fonctionnels). Par conséquent le contrôle de la coopération varie d'un outil à un autre dans la même *pièce*. Une pièce se résume à une collection de collecticiels.

Une autre conséquence du choix de collecticiels comme outils de production est que chaque utilisateur doit être associé à un ou plusieurs rôles fonctionnels par outil disponible dans la *pièce*. Les rôles fonctionnels ne sont plus définis en fonction de la nature de la tâche à réaliser (selon le type de la pièce) mais en fonction des actions qu'un outil peut effectuer. Or compte tenu qu'il est impossible d'imaginer lors de la conception tout contexte possible d'utilisation d'un outil les rôles fonctionnels définis par un outil ne peuvent qu'être génériques. Ceci peut poser un problème lors de la définition d'une nouvelle tâche (pièce). Par exemple un collecticiel d'édition coopérative peut proposer les rôles fonctionnels suivants : administrateur, rédacteur et lecteur. Par exemple, la mise en page d'une publication requiert la définition d'un rôle d'*imprimeur* qui doit avoir la possibilité de changer l'apparence du document sans pouvoir changer son contenu ni sa structure logique. Il est impossible dans *TeamRooms* de définir un tel rôle sans modifier le collecticiel d'édition employé. Par contre la définition du rôle décrit ci-dessus est directe dans *Colt* ; il suffit de créer ce rôle fonctionnel et lui associer les opérations nécessaires pour la mise en page.

3.7.3 Comparaison générale

L'environnement *Colt* présente une solution complète pour supporter le travail coopératif *contractuel*. Il réalise l'ensemble des objectifs que nous avons fixés dans le paragraphe (3.2) à savoir : l'implantation d'un modèle générique de rendez-vous, l'intégration du travail individuel et du travail coopératif et le support d'exécution des scénarios coopératifs variés. Les objectifs sont atteints grâce à la souplesse fonctionnelle et structurelle du système. Selon l'axe structurel *Colt* permet de définir et de réajuster dynamiquement les rôles des utilisateurs, de définir autant d'activités de groupe que l'on souhaite et d'utiliser dans ces différentes activités les outils que l'on a besoin. Sur l'axe fonctionnel les utilisateurs peuvent définir et changer dynamiquement la configuration d'une activité. Le tableau (3.5) dresse une comparaison fonctionnelle entre *Colt* et les plates-formes

	XTV	Sol	Mead	GroupKit	CoopScan	ActiveMail	GroupDesign	**Colt**
Rdv. explicite	⊗			⊗	–	●	–	●
Rdv. implicite	–			–	–	⊗	–	●
Groupe↔individuel	–			–	–	–	–	●
Enregistrment libre	●	–	–		●	–	●	●
Enreg. supervisé	–	⊗	⊗		⊗	⊗	–	●
Participation dynamique	●				–	●	●	●
Rôles fonctionnels	–	⊗	⊗	–	–	–	⊗	●
Interaction synchrone	●	●	●	●	●	–	●	●
Interaction asynchrone	–	–	–	–	–	●	⊗	●
Co-présence	⊗	–	–	⊗	⊗	–	●	●
Gestion de droits de parole	⊗				⊗	–	●	●
Configuration dynamique	–	⊗	⊗		●	–	●	●
Ouverture	–	–	–	●	⊗	–	⊗	⊗

TAB. 3.5 - : Comparaison fonctionnelle entre Colt et les plates-formes présentées dans la section 2.5.

présentées dans la section (2.5). Dans ce tableau nous utilisons la notation suivante : • pour implantation convenable, ⊗ pour implantation partielle, – pour implantation absente et blanc pour implantation non précisée. Le tableau (3.5) montre que *Colt* répond à l'ensemble des besoins demandés à une plate-forme pour le travail coopératif. C'est le choix de définir les fonctions de la coopération au niveau des données qui nous a permis, en premier chef, d'atteindre cette degré *complétude* fonctionnelle.

D'autres travaux ont exploré l'axe de définition de la coopération au niveau des données. Un premier exemple est la plate-forme *Mead* présentée dans la section (2.5.5). Nous rappelons que la conception de *Mead* repose sur la définition d'un serveur d'objets partagés (`OSS`) sur lequel à chaque utilisateur est attribué une vue personnalisée (`UD`). La vue d'un utilisateur est définie par des agents spéciaux (les `UDA`). Les `UDA` peuvent être partagées par plusieurs utilisateurs ce qui rend possible l'implantation et la cohabitation des formes variées de coopération. La souplesse de *Mead*, comparée à celle de *Colt*, est bien limitée. Elle est contrôlée par deux *variables* qui sont les relations `UD-UDA` et les relations `UDA-OSS`. Ainsi *MEAD* propose deux paramètres de contrôle (les deux relations citées ci-dessus) pour *décrire* ce que dans *Colt* est exprimé par les rôles utilisateurs, les rôles organisationnels et les modèles d'activités qui englobent la définition des rôles fonctionnels et des politiques de contrôle, d'enregistrement et de participation variées. Un *UDA* compile les concepts des rôles organisationnels, utilisateurs et fonctionnels dans une seule entité logicielle. Il faut rappeler que *MEAD* est à l'origine conçu pour réaliser un système coopératif de contrôle aérien [Bentely et al.92a].

Ayant une application cible bien définie justifie la réduction de la souplesse du système au profit de la simplicité du contrôle. La plupart des dimensions façonnables dans *Colt* comme par exemple la structure de l'organisation, la natures des tâches à réaliser, les relations entre les membres de l'organisation, sont plus au moins connues à l'avance dans le cadre d'une organisation spécifique telle que un groupe de contrôleurs aériens.

Les deux plates-formes *CoopScan* et *GroupDesign* qui sont fondées sur le principe d'intégration d'applications existantes au niveau de l'application implantent à leur tour les fonctions de la coopération au niveau des données grâce à l'exploitation de la sémantique des commandes utilisateurs. Les deux systèmes ne se sont pas intŕessé au volet organisationnel du travail coopératif mais ils s'appuient sur une représentation explicite des activités à réaliser (définition explicite des sessions de travail en groupe). Dans *CoopScan* la propagation des actions entre les sites repose sur l'implantation d'un protocole spécifique par application employée entre l'agent local et ses représentants sur les autres sites. *GroupDesign* étend l'expérience de *CoopScan* en implantant des protocoles inter-applications ce qui permet d'utiliser des applications hétérogènes dans la même session. Dans *Colt* nous généralisons l'approche de *GroupDesign* en employant le concept d'opérations génériques. Au lieu d'implanter un protocole d'échanges d'actions par paire d'applications intégrées, nous fournissons un seul module de traduction entre les opérations effectives des applications et les opérations génériques qui les représentent. La plate-forme *GroupDesign* présente, par rapport à *CoopScan* et à *Colt*, l'avantage de fournir des outils de *fragmentation* des documents employés et des commandes pour spécifier les droits d'accès des utilisateurs aux différentes fragments. La fragmentation dynamique des documents permet d'envisager des formes plus variées de coopération comme par exemple être en coopération synchrone sur un fragment du document et en asynchrone sur un autre fragment du même document. L'environnement *Colt* a aussi ses limits. Le premier est le manque d'un langage évolué pour la spécification des activités coopératives comme c'est dans [Paoli et al.94]. De plus, les modèles d'activités fournis par *Colt* sont très simples. Les interactions possibles entre les utilisateurs sont limitées : si les utilisateurs sont en même temps dans la même *pièce*, ils sont forcément en mode d'interaction synchrone. Un autre problème non traité par *Colt* est celui des apartés qui peuvent avoir lieu dans une activité [Villemur95].

3.8 Conclusion

Le concept d'environnement de coopération représente la dernière étape de l'évolution des systèmes interactifs. Du paradigme d'interaction $1 \leftrightarrow 1$ c'est à dire le dialogue homme-application, les systèmes interactifs ont évolué vers le paradigme $1 \leftrightarrow N$ où un seul utilisateur interagit avec plusieurs applications à la fois, puis vers le paradigme $M \leftrightarrow 1$ qui représente les collecticiels où un en-

semble d'utilisateurs interagissent entre eux via le partage d'une application. En fin, un *EdC* implante le paradigme $M \leftrightarrow N$ où plusieurs utilisateurs travaillent ensemble en utilisant plusieurs applications. Dans ce chapitre, nous avons proposé une approche originale pour la construction d'un environnement d'*EdC* qui soit adaptable tant au niveau structurel qu'au niveau fonctionnel. Notre approche consiste à : 1) implanter un espace partagé par les utilisateurs où chacun peut voir et se mouvoir selon des rôles organisationnels qui lui sont attribués et 2) fournir des mécanismes de représentation et de manipulation explicite des activités coopératives (gestion explicite des pièces partagées). La conception de *Colt* présente plusieurs points innovant, en particulier l'abondon du principe de partage d'applications pour coopérer au profit du reassemblage des outils individuels dans l'espace d'une activité. Un autre point intéressant est l'implantation d'un modèle souple pour la définition et l'évaluation des droits d'accès et de règles de protection de l'espace partagé. Une nouveauté présentée par ce modèle est l'introduction de la notion du rôle utilisateur comme un rôle à part entière auquel nous pouvons associer des privilèges exceptionnels (positifs ou négatifs). Une comparaison avec d'autres plates-formes et environnements de coopération nous a montré la richesse fonctionnelle de notre proposition. La faisabilité du concept est démontrée par le développement d'un prototype réduit. Cependant un problème qui reste ouvert et dont souffrent la plupart des collecticiels, est celui de l'utilisabilité de l'environnement. L'étude d'utilisabilité nécessite des compétences pluridisciplinaires qui n'étaient disponibles lors du prototypage de notre environnement.

Chapitre 4

Contrôle de la concurrence dans les collecticiels synchrones

4.1 Introduction

Nous étudions dans ce chapitre le problème de gestion des accès concurrents aux données partagées dans les collecticiels synchrones. Nous considérons ici les collecticiels synchrones à architecture totalement dupliquée. Nous rappelons que la plupart des collecticiels synchrones sont mis en œuvre selon ce type d'architecture qui présente les avantages suivantes :

- Un court temps de réponse puisque tous les traitements sont faits en local.
- Un trafic réseau réduit puisque seules les requêtes des utilisateurs sont véhiculées à travers le réseau.
- Une bonne disponibilité des données et une bonne résistance aux pannes due à la duplication des ressources.

Selon cette architecture chaque utilisateur possède sa copie locale des données. Il est évident que si rien n'est prévu pour contrôler l'accès et les mises à jour des copies locales les différentes copies peuvent devenir mutuellement incohérentes. L'incohérence rend difficile l'avancement de la tâche du groupe. La résolution de ce problème repose sur l'emploi d'un mécanisme qui contrôle les modifications concurrentes des objets dupliqués.

Le problème posé ci-dessus est le même que celui de la gestion des *caches*[1] dans les systèmes répartis [Balter et al.91]. De nombreux protocoles sont proposés dans la littérature des systèmes répartis pour résoudre ce problème. Malheureusement ces protocoles ne satisfont pas les besoins des collecticiels synchrones. À

[1] Un cache est un dispositif matériel ou logiciel qui conserve une copie d'une information dite primitive, l'accès à la copie du cache étant plus rapide que l'accès à l'information primitive.

l'origine de cette inadéquation se trouve l'aspect *interactif* des collecticiels. Nous développons ce point dans le paragraphe suivant.

L'inadéquation des protocoles classiques de gestion de la concurrence
La mise en œuvre de l'interface utilisateur d'une application interactive repose généralement sur le principe de la *manipulation directe* [Shneiderman83]. Selon ce principe, pour modifier un objet l'utilisateur interagit directement avec une *représentation* de cet objet sur l'interface utilisateur. Par exemple dans un éditeur graphique, pour changer la taille d'un rectangle l'utilisateur sélectionne d'abord le rectangle à modifier puis déplace l'un de ses angles jusqu'à l'obtention de la taille désirée.

Selon le principe de la manipulation directe toutes les *copies* dupliquées sont en état de lecture *permanent*. Ainsi les protocoles classiques qui garantissent la cohérence des copies dupliquées au lecture ne trouvent pas leur place dans le contexte des collecticiels. Une autre conséquence de ce principe est que toute modification d'une copie locale est immédiatement visible sur l'interface utilisateur sur le site local. Les protocoles qui peuvent produire des séquences de d'annulation d'opérations, suite à l'annulation d'une transaction par exemple, peuvent alors générer une confusion chez l'utilisateur [Greenberg et al.94].

D'autre part, toute mise en œuvre du principe de la manipulation directe est contrainte de vérifier deux propriétés fondamentales : 1) l'honnêteté de la représentation des objets et 2) la réduction du temps de réaction à toute action de modification d'un objet. Une représentation est honnête si elle est conforme à l'état interne de l'objet représenté. L'honnêteté implique que toute modification d'un objet sur un site doit être notifiée au plus vite possible aux autres sites. La deuxième propriété implique que le temps de réponse du collecticiel doit être le plus court possible. Par conséquent, les collecticiels synchrones posent des contraintes temporelles plus sévères que celles posées par les applications réparties classiques (transparentes).

Une dernière propriété requise pour les collecticiels et qui est mal supportée par les protocoles classiques de gestion de la concurrence est celui la présentation de la co-présence des utilisateurs dans l'espace du collecticiel. Les protocoles classiques sont fondés sur le principe de la transparence de la concurrence où chaque utilisateur a l'illusion qu'il est le seul à utiliser le système. Or la présentation de la co-présence des utilisateurs nécessite de prévenir chaque utilisateur de la position et des actions faites par tout autre participant. Par exemple, lorsqu'un utilisateur verrouille un objet en vue de le modifier, les autres utilisateurs du système doivent être prévenus du verrouillage de cet objet et de l'identité de l'utilisateur qui l'a fait. Ayant les informations précédentes, les utilisateurs seront en mesure de négocier entre eux la possession des droits d'accès à chaque objet du système.

Nous présentons le problème de gestion des accès concurrents et les principales propriétés requises d'un protocole de gestion de la concurrence, dit aussi

protocole de gestion de droits de parole ou encore protocole de médiation, dans la section (4.2). Une classification des protocoles existants est donnée dans la section (4.3), les différents mécanismes et politiques de mise en œuvre des protocoles identifiés sont résumés dans la section (4.4). Une comparaison entre les différentes approches en termes de leur satisfaction des propriétés énumérées dans (4.2.1) est faite dans la section (4.4.3). Nous savons à l'avance qu'il n'existe pas un protocole qui peut être employé dans tout type d'activité coopérative. L'étude menée dans [Greenberg et al.94] montre qu'un collecticiel synchrone doit implanter plusieurs protocoles de gestion de droit de parole. Nous ajoutons à l'ensemble des protocoles proposés dans la littérature du *TCAO* un nouveau protocole que nous appelons *LICRA* (4.5). *LICRA* est un protocole complètement automatique (ne nécessite pas l'intervention des utilisateurs) ; par conséquent il est facile à utiliser par les utilisateurs finals. Il permet au collecticiel d'avoir un temps de réponse optimal. L'approche empruntée par *LICRA* est la suivante : chaque utilisateur peut modifier sa copie locale des données à n'importe quel moment. Toute opération est exécutée immédiatement sur la copie locale ce qui permet d'avoir un temps de réponse optimal. Chaque opération exécutée sur un site est diffusée aux autres sites participant à la session. À la réception d'une opération *LICRA* exécute une procédure qui garantit que toutes les copies des données sur tous les sites soient les mêmes lorsque toutes les opérations générées dans la session sont reçues et traitées par tous les sites. En contre partie de la simplicité et de l'efficacité du protocole proposé se trouve une *lourde* procédure d'*installation*: La procédure automatique de la résolution des conflits s'appuie sur l'exploitation des connaissances sémantiques des opérations échangées entre les sites.

4.2 Définition du problème

4.2.1 Modélisation d'un collecticiel synchrone

Nous décrivons dans cette section un modèle abstrait d'un collecticiel synchrone à architecture dupliquée. Ce modèle nous sert à clarifier les propriétés requises d'un protocole de gestion de la concurrence que nous présentons dans la section qui suit.

Nous utilisons le terme *session* pour désigner l'exécution d'un collecticiel synchrone. Une session S à l'instant t est définie par: $S_t = \{\Delta_t, \Phi_t\}$ où Δ_t est l'ensemble des sites impliqués à l'instant t et Φ_t l'ensemble des fonctions fournies par le collecticiel au même instant. Les compositions des deux ensembles précédents varient avec le temps. La composition de Δ change en fonction de la connexion et de la déconnexion des utilisateurs tandis que la composition de Φ change en fonction du lancement et de la terminaison d'applications.

Un site $\delta_i \in \Delta_t$ est défini par le couple: $\delta_i = (P_i, O_i)$ où i est l'identificateur du site, P_i est le processus du site et O_i la copie locale des données manipulées

par le collecticiel. L'ensemble O_i comprend des objets passifs qui représentent les données utilisateurs. Le processus d'un site exécute quatre types d'actions :

1. *Génération d'une opération* qui correspond à la demande de l'application d'une opération à travers l'interface-utilisateur.

2. *Exécution d'une opération* qui correspond à l'application d'une opération générée localement ou reçue d'un autre site sur la copie locale de données (O_i).

3. *Diffusion d'une opération* qui correspond à la diffusion d'une opération générée localement vers les autres sites.

4. *Réception d'une opération* qui correspond à la réception d'une opération envoyée par un autre site.

La figure (4.1) illustre le modèle d'un site.

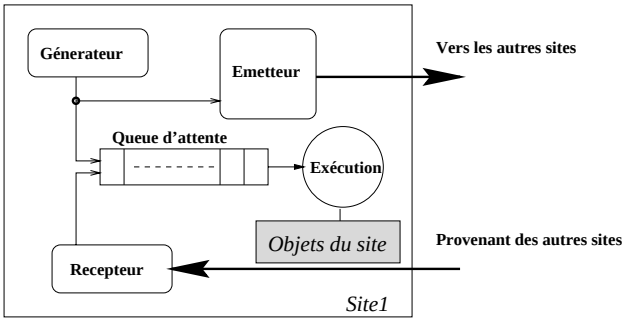


FIG. 4.1 - : Modélisation d'un site dans un collecticiel synchrone à architecture dupliquée.

Le déroulement d'une session est fait de la façon suivante : lorsqu'un utilisateur demande l'application d'une commande, via l'interface homme-machine, le processus de son site génère une ou plusieurs opérations. Les opérations générées sont insérées dans une file locale d'opérations en attente d'exécution. Parallèlement, le processus du site diffuse les opérations générées aux autres sites dans la session. Les opérations reçues des autres sites sont également insérées dans la même file d'attente. L'ordre d'insertion et d'extraction des opérations de la file d'attente dépend du protocole de gestion de la concurrence employé. Nous présentons les différentes politiques et schémas de mise en œuvre de protocoles de gestion de la concurrence dans la section (4.3).

4.2.2 Propriétés

Nous énumérons dans la suite les principales propriétés requises d'un protocole de gestion de la concurrence dans les collecticiels synchrones.

La correction L'objectif principal d'un protocole de gestion de la concurrence est de préserver la cohérence des données dupliquées. La vérification de cette propriété n'a de sens qu'aux instants où toutes les opérations générées par tous les sites sont reçues et *traitées* par tous les autres sites. De tels instants, dénotés par Q_T, sont appelés les *instants de repos*. La propriété de la correction s'écrit alors de la manière suivante :

$$\forall t \in Q_T\,, \forall \delta_i\,, \delta_j \in \Delta_t\ : O_i\ =\ O_j$$

L'interactivité L'interactivité est mesurée par deux paramètres : *le temps de réponse* et *le temps de notification*. Le premier (respectivement le dernier) correspond à la durée qui sépare le lancement d'une action de la visualisation de ses effets sur l'écran de l'utilisateur local (respectivement les écrans des utilisateurs distants). Un collecticiel a un temps de réponse optimal si toute opération locale est exécutée immédiatement. Le temps de notification, entre deux sites, (dénoté T_N), est donné par la formule suivante : $T_N = T_e + T_r + T_l$ où :

- T_e est le temps nécessaire pour préparer l'émission de l'opération.
- T_r est le temps qui sépare l'émission d'un opération de sa réception sur le site distant. Autrement dit T_r est le temps du passage de l'opération dans le réseau.
- T_l est le temps qui sépare la réception d'une opération de son exécution sur le site récepteur.

Le temps T_r dépend des caractéristiques du réseau employé. Par conséquent, un protocole de gestion de la concurrence doit chercher à minimiser T_e et T_l. L'interactivité se résume donc à minimiser le passage des opérations dans les files d'attente sur les sites participant à la session.

La faisabilité Il s'agit de réduire le surcoût du fonctionnement du protocole. Cette propriété exprime des contraintes sur la consommation des autres ressources comme l'espace mémoire et le débit du réseau de communication. Par exemple il faut éviter d'employer de structures de données dont la taille augmente avec le temps (ex. les logs d'exécution) ou au moins prévoir de mécanismes de réduction qui limite la croissance monotone de ce type des structures de données.

L'ergonomie de l'interface-utilisateur Une liste des propriétés ergonomiques qu'il est souhaitable qu'une interface-utilisateur d'une application interactive vérifie est donnée dans [Abowd et al.92]. La mise en œuvre d'un protocole de gestion de la concurrence doit respecter ces propriétés ergonomiques. Nous citons deux propriétés particulièrement importantes :

- *La pro-activité* caractérise les retours qui indiquent à l'utilisateur les actions interdites. Par exemple dans le cas de l'emploi d'une politique fondée sur le verrouillage, les objets verrouillés doivent être mis en relief afin qu'un autre utilisateur n'essaie pas de les modifier.
- *La stabilité du temps de réponse* dénote la capacité du système à entretenir un même temps de réponse pour un traitement donné. Autrement dit il est souhaitable que le surcoût introduit par le protocole de gestion de la concurrence soit constant.

La simplicité de l'emploi L'effort que chaque utilisateur doit fournir pour la coordination d'un travail coopératif doit être nettement inférieur au profit apporté par l'utilisation du collecticiel [Grundin88]. Par conséquent il est souhaitable que l'intervention des utilisateurs dans le contrôle de l'exécution d'un protocole de gestion de la concurrence soit aussi réduit que possible. À une extrémité de ce critère nous trouvons les protocoles automatiques, à l'autre extrémité il y a les protocoles manuels.

4.3 Éléments d'un protocole de gestion de la concurrence

Nous décomposons un protocole de gestion de la concurrence en trois couches, dites modèles : 1) le modèle de la concurrence, 2) le modèle de la cohérence et 3) le modèle de la communication.

1. Le modèle de la concurrence décrit le degré de parallélisme supporté par le protocole. Trois degrés de concurrence sont identifiés :

1. *Tout ou rien* où un seul utilisateur a le droit de modifier l'espace partagé à un instant donné.

2. *Accès parallèles* où plusieurs utilisateurs peuvent modifier des zones différentes et indépendantes dans l'espace partagé ou bien appliquer des opérations non contradictoires sur un même objet. Un exemple est l'application sur un objet graphique les deux commandes : *ChangerCouleur* et *ChangerTaille*.

3. *Accès simultanés* où plusieurs utilisateurs peuvent modifier en même temps les mêmes données (chacune sur son site local).

2. Le modèle de la cohérence décrit les contraintes imposées aux mises à jour des objets dupliqués. Des définitions formelles de différentes critéres de cohérence sont données dans [Raynal et al.95]. Nous citons dans la suite trois types de cohérence les plus employés :

1. *La cohérence faible* selon laquelle la mise à jour de toute copie doit avoir lieu au bout d'un temps borné ; néanmoins aucune contrainte n'est imposée sur l'ordre de mise à jour.

2. *La cohérence causale* complète la cohérence faible par la contrainte suivante : les modifications de toutes les copies doivent être faites selon l'ordre causal des mises à jour. L'ordre causal est défini comme suit. Si une opération Op_m est générée sur un site δ_i après l'exécution d'une opération Op_n alors l'exécution de Op_m prend lieu après l'exécution de Op_n sur tous les autres sites.

3. *La cohérence forte* (dite aussi cohérence sequentielle) complète la cohérence faible par la contrainte suivante : les modifications de toutes les copies doivent être faites dans le même ordre.

3. Le modèle de la communication décrit les propriétés du protocole de communication de groupe sur lequel s'appuie le protocole de gestion de la concurrence. Deux propriétés décrivent le comportement d'un protocole de communication de groupe : 1) la fiabilité et 2) l'ordre de délivrance de messages.

Un protocole de communication est *fiable* s'il garantit que tout message émis par un site est reçu correctement par tous les autres sites au bout d'un temps fini et borné. En ce qui concerne l'ordre de livraison de messages plusieurs relations d'ordre peuvent être définies. Trois relations d'ordre classiques sont les suivantes [Raynal90] :

1. *L'ordre FIFO* selon lequel deux messages consécutifs émis par le même site sont reçus par tous les autres sites dans leurs ordre d'émission.

2. *L'ordre causal* selon lequel la livraison de messages est faite selon l'ordre causal.

3. *L'ordre total* selon lequel les processus des sites reçoivent dans le même ordre les opérations générées dans la session.

Les deux modèles de la concurrence et de la cohérence constituent le cahier de charge du concepteur d'un protocole de gestion de la concurrence tandis que

le modèle de la communication fait partie des caractéristiques de l'environnement de l'exécution du collecticiel. Le problème de la conception d'un nouveau protocole est alors exprimé de la manière suivante : étant donné un modèle de communication, comment faire pour réaliser tel modèle de concurrence et tel modèle de cohérence tout en respectant au mieux les propriétés citées dans la section (4.2.1) ?

La simple combinaison des deux modèles de la concurrence et de la cohérence définit huit classes fonctionnellement différentes de protocoles de gestion de la concurrence. Quelle classe est la plus adaptée pour les collecticiels synchrones ? Plusieurs études dans le domaine du *TCAO* ont montré qu'il n'existe pas un seul protocole qui soit adapté à tous les collecticiels synchrones ou même à toutes les exécutions du même collecticiel. À titre d'exemple, nous énumérons dans la suite quelques paramètres qui peuvent influencer le choix du protocole à appliquer.

- *Le nombre de participants.* Si le nombre de participants est petit, il y a plus de chance que les utilisateurs arrivent à coordonner leurs actions en s'appuyant simplement sur les retours visuels indiquant la position de chacun d'eux. Dans ce cas un protocole à haut degré de concurrence peut être employé sans provoquer beaucoup de conflits. La probabilité de l'occurence d'un conflit augmente avec l'augmentation du nombre de participants. Pour un grande nombre il est préférable d'appliquer un protocole à faible degré de concurrence.

- *La nature de la session.* Dans une session de télé-enseignement ou d'une séminaire l'orateur est le seul à pouvoir adresser le parole à l'ensemble des participants. Pour intervenir, un utilisateur demande normalement l'autorisation de l'orateur. Les politiques à forte degré de concurrence ne trouvent leur place dans pareilles sessions.

- *La nature des applications employées.* Par exemple dans un collecticiel de vote il est possible d'employer un protocole à cohérence faible puisque l'ordre de mise à jour des votes n'influence pas le résultat final. Par contre dans un collecticiel d'édition il est souhaitable que les utilisateurs perçoivent les mises à jour selon un ordre causal afin de mieux comprendre l'évolution du document.

4.4 Mise en œuvre d'un protocole de gestion de la concurrence

Nous identifions deux niveaux distincts dans la mise en œuvre d'un protocole de gestion de la concurrence : le niveau des mécanismes employés et le niveau de la politique d'exploitation des mécanismes choisis.

4.4.1 Les mécanismes

Deux types de mécanismes sont généralement employés : le verrouillage et l'ordonnancement des opérations.

1. Le verrouillage Le principe est de lier la capacité de modifier un objet à l'acquisition d'un verrou *exclusif* sur cet objet. La gestion des verrous nécessite l'introduction des fonctions suivantes :

1. `DemanderVerrou (Objet)`. L'exécution de cette fonction est une condition nécessaire mais non suffisante pour verrouiller un objet ; la requête de verrouillage peut être rejetée. Par exemple lorsque deux utilisateurs demandent en même temps de verrouiller le même objet, le système d'allocation de verrous est contraint de préférer l'un à l'autre.

2. `AttribuerVerrou(Objet,U)`. Cette fonction verrouille l'objet $Objet$ au profit de l'utilisateur U. À partir de ce moment et jusqu'à la libération de l'objet désigné, seul U a le droit de modifier cet objet.

3. `LibérerVerrou(Objet)`. Cette fonction permet de relâcher le verrou associé à un objet. L'objet désigné devient de nouveau libre. D'autres utilisateurs peuvent alors le verrouiller.

La plupart des systèmes coopératifs implantant de protocoles de verrouillage rendent visible à chaque utilisateur le contenu de la table d'allocation de verrous. L'affichage du contenu de la table d'allocation revient à présenter sur l'interface utilisateur les objets verrouillés et l'identité des détenteurs des verrous [Baecker et al.93, Karsenty94a].

2. La détection de dépendances Le principe est de définir une relation d'ordre sur l'ensemble des opérations générées dans la session et de garantir que sur chaque site l'exécution de la session donne le même résultat que l'exécution des opérations selon l'ordre défini.

Une opération Op_x dépend directement d'une opération Op_y (dénoté $Op_y \mapsto Op_x$) si Op_x est générée sur un site immédiatement après l'exécution de Op_y sur le même site. Sur l'exemple illustré à la figure (4.2) nous avons $Op_1 \mapsto Op_2$ et $Op_1 \mapsto Op_3$. Deux opérations Op_x et Op_y sont en concurrence (dénotées $Op_x \parallel Op_y$) si aucune d'elles ne dépend de l'autre. Ainsi dans l'exemple précédent nous avons $Op_2 \parallel Op_3$.

Plusieurs mécanismes sont élaborés afin d'ordonner les opérations dans un système réparti. Parmi ceux-ci nous citons l'emploi d'horloges logiques [Lamport78] et les horloges vectorielles [Fidge88]. Un mécanisme d'estampillage définit la structure de l'estampille, la fonction de mise à jour de l'estampille et une relation de comparaison entre estampilles. La relation de dépendance entre opérations

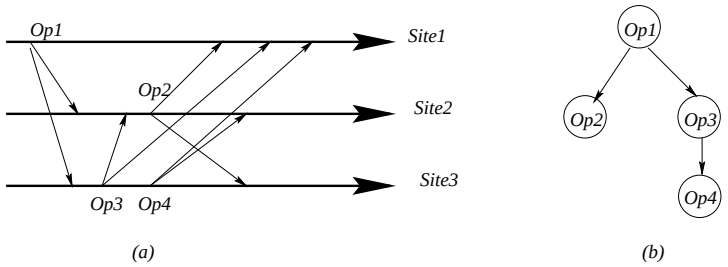


FIG. 4.2 - : Exemples de relations de dépendance et de concurrence entre opérations.

n'étant pas une relation d'ordre global le système est complété par une fonction de priorité entre les sites afin de pouvoir ordonner les opérations concurrentes.

4.4.2 Les politiques

Bien évidement la définition d'une politique dépend de la nature des mécanismes à employer (verrous ou estampilles). Cependant nous identifions trois critères de classement de politiques qui sont les suivants : 1) *la démarche*, 2) la *résolution* et 3) *l'automatisme*.

1. La démarche Nous identifions deux démarches possibles : la démarche pessimiste et la démarche optimiste. La démarche pessimiste repose sur le principe d'éviter les conflits. Chaque opération est *soupçonnée* d'être en conflit avec une autre. Une opération n'est exécutée qu'après avoir contrôlé l'absence des conflits. Dans le cas d'utilisation de verrous, une politique pessimiste ne permet pas la modification d'un objet avant l'acquisition définitive d'un verrou sur cet objet. Un protocole pessimiste qui repose sur une technique de détection de dépendances *force* l'exécution effective des opérations selon l'ordre employé.

Une démarche optimiste consiste à procéder par certification d'absence de conflits après l'exécution des opérations. En cas de détection d'un conflit la politique fait appel à un mécanisme de repérartion (ou de restauration de la cohérence) qui ramène le système à un état cohérent.

2. La Résolution Il s'agit de définir le degré de la concurrence supporté par la politique (voir 4.3).

3. L'automatisme Il s'agit de définir le degré d'implication des utilisateurs dans le contrôle de déroulement du protocole. Théoriquement toute fonction de manipulation des mécanismes employés par un protocole peut être déclenchée par le système ou par un utilisateur. Dans le premier cas nous disons que la fonction est déclenchée d'une manière *implicite* tandis que dans le deuxième cas nous disons que la fonction est déclenchée d'une manière *explicite*.

Pour fixer les idées, prenons l'exemple d'une politique qui repose sur un mécanisme de verrouillage et dont la taille de verrou est égale à la taille de l'espace tout entier (cas d'un protocole de tour de rôle). La tableau (4.1) illustre quelques variations possibles des protocoles de tour de rôle selon la nature de l'utilisation des fonctions de gestion du verrou. Dans ce tableau le lettre E (respectivement I) désigne une manipulation explicite (respectivement implicite).

	Demande	Libération	Attribution
`Animation`	E/I	E	E
`FIFO`	E	E/I	I
`Désignation`	E	E	E
`Anneau`	I	E/I	I
`Temps partagé`	I	I	I

TAB. 4.1 - : Exemples de protocoles de tour de rôle.

Dans certains cas, le choix du mode d'utilisation d'une fonction est réglée d'avance. Par exemple dans un protocole à base de détection de dépendance il est peu utile que l'estampillage des opérations soit fait par les utilisateurs !

Plus un protocole est automatisé, plus son utilisation est simple, mais moins il est souple. Les décisions faites par le système dans la cadre de l'automatisation d'un protocole peuvent ne pas correspondre à l'attente des utilisateurs. Par exemple dans un protocole à base de verrouillage quelle taille choisir pour un verrou? Quand faut-il demander un verrou? et quand faut-il le libérer? Une autre fonction dont l'automatisation est très controversée est la fonction de restauration de la cohérence des copies que tout protocole à politique optimiste doit utiliser. La restauration automatique de la cohérence des copies s'appuie forcément sur une fonction de *priorité* entre les sites. La simple suppression des contributions faites par un site *défavorisé* peut priver le groupe des contributions intéressantes. En contre partie, la restauration manuelle de la cohérence nécessite l'implantation d'un système de gestion et de présentation de versions de données [Condon92], [Moran et al.93]. Lorsque deux utilisateurs modifient en même temps un même objet le système crée deux versions de cet objet, une par modification. Or si les utilisateurs ne décident pas rapidement quelle version il faut adopter, le nombre de versions va augmenter d'une manière exponentielle suite à des modifications concurrentes des versions initiales ; la gestion et la représentation du document deviennent de plus en plus lourdes.

4.4.3 Exemples

4.4.3.1 Tour de rôle

Un protocole de tour de rôle implante un modèle de concurrence de type tout ou rien ; le modèle de la cohérence peut être la cohérence faible ou la cohérence forte. Le principe est d'associer la capacité à modifier l'espace partagé à l'acquisition d'un verrou unique dit le *jeton*. Le fait qu'un seul utilisateur peut modifier l'espace partagé à un moment donné rend équivalents les deux modèles de la cohérence causale et la cohérence forte. Une stratégie de négociation de la possession du jeton (ou négociation de droit de parole) décrit le protocole de circulation du jeton entre les participants à la session. L'implantation d'une stratégie fait appel aux trois mécanismes suivants : 1) *Demande du jeton*, 2) *Libération du jeton* et 3) *Attribution du jeton*. Chacun des mécanismes précédents peut être implanté d'une manière explicite ou implicite. Le tableau (4.1) illustre les principales stratégies de négociation de possession du jeton.

La validité d'un protocole de tour de rôle repose sur l'existence d'un jeton unique, qui peut être mise en défaut par la panne d'un site (au moment où il devient le jeton) ou du système de communication. Une autre caractéristique requise d'un protocole de tour de rôle est l'absence de la famine qui se traduit par la condition suivante : toute demande d'acquisition du jeton doit être traitée au bout d'un temps borné.

La complexité de la mise en œuvre d'un protocole de tour de rôle dépend des caractéristiques du système cible (fiabilité des sites) et du modèle de communication employé. Les protocoles de tour de rôle présentent l'avantage d'être à la fois simples à employer par les utilisateurs et simples à mettre en œuvre. Leurs principal inconvénient est le mode de concurrence très restrictive qu'ils implantent.

4.4.3.2 Verrouillage implicite, l'exemple d'*Ensemble*

Ensemble est un éditeur graphique coopératif synchrone [NW et al.92]. Il propose un protocole de gestion de droit de parole qui implante un modèle de concurrence d'accès parallèle et un modèle de cohérence forte.

La mise en œuvre du protocole est faite sur un modèle de communication fiable à ordre causal. Le protocole repose sur l'emploi des verrous. La politique de gestion de verrous suit une démarche pessimiste. Il présente la particularité d'être entièrement automatique.

Le principe de fonctionnement du protocole est le suivant. À chaque participant est attribuée une *couleur* qui le distingue des autres participants. Lorsqu'un utilisateur demande via l'interface utilisateur la sélection d'un objet le processus du site local envoie une requête de verrouillage de l'objet désigné à un *serveur de verrouillage*. Ce dernier maintient une table d'allocation de verrous aux utilisateurs. Si la requête de verrouillage est acceptée le serveur informe l'ensemble des

sites de l'identité de l'utilisateur ayant verrouillé l'objet. La couleur de marque de sélection de l'objet est celle de l'utilisateur qui possède ce verrou. Un utilisateur ne peut pas modifier un objet sélectionné par un autre utilisateur. Lorsqu'un utilisateur désélectionne un objet, le serveur de verrous libère l'objet désigné. Les autres utilisateurs sont en mesure de le verrouiller.

L'avantage de ce protocole est qu'il permet aux utilisateur d'interagir avec l'éditeur dans les situations de travail en groupe de la même façon que dans le cas d'utilisation individuelle de l'éditeur.

4.4.3.3 Verrouillage explicite, l'exemple de *Mace*

MACE est une application coopérative de traitement de texte [Nastos92]. Il implante un protocole de gestion de la concurrence à base de verrouillage. La gestion des verrous est presque entièrement manuelle. Pour verrouiller une zone du texte, un utilisateur doit placer deux verrous qui délimitent la zone à verrouiller. Les verrous sont présentés sous forme d'une petite icône de verrou portant le nom du propriétaire. Ici, la taille du verrou et les demandes de verrouillage et de libération sont faites d'une manière explicite. Par conséquent les utilisateurs peuvent négocier entre eux les limites et les tailles des zones accessibles pour chacun d'eux.

4.4.3.4 Le pèlrin

Ce protocole est proposé dans le cadre de la plate-forme coopératif *CAliF* [Guyennet et al.97]. Il implante un modèle de concurrence d'accès parallèls et un modèle de cohérence faible. La mise en œuvre du protocole est faite sur un modèle de communication fiable. Un souci principal du *Pèlrin* est d'alléger le trafic réseau. Dans cette optique le protocole emploi un jeton, appelé le pèlrin, dont la structure de données porte à la fois les commandes de négociation des propriétés des objets manipulés dans le système et les mise à jours effectués. Chaque objet est à un instant donné la propriété d'un seul site qui est le seul à pouvoir la manipuler (accès parallèls). Les sites coopérants sont placés sur un anneau virtuel sur lequel le pèlrin circule automatiquement. Arriver sur un site le pèlrin subit le traitement suivant avant d'être expédié sur le site voisin :

1. Lecture : Le site applique les mises à jour portés par le pèlrin et traite les demandes d'acquisition de propriété des objets en son possesion.

2. Écriture : Le site ajoute au jeton les modifications qu'il appliqué aux objets locals depuis le dernier passage du pèlrin puis ajoute à celui-ci ses réponses aux réquêtes d'acquisition d'objets et ses propres requêtes de demandes d'acquisition d'objets.

Selon ce protocole les seuls messages qui transitent sur le réseau sont les messages de circulation du pèlrin. Le protocole atteint son objectif de réduction du

trafic réseau sur le compte des propriétés requises de l'interface homme-machine du système. En effet le temps de réponse de l'interface n'est plus optimal (du au temps d'attente d'acquisition des verrous) et il n'est plus stable (la même action peut prendre des durée variées selon la taille du pèlrin et du nombre des sites participants. Le temps de notification est aussi penalisé par le choix d'un anneau undirectionel.

4.4.3.5 Oreste

Le protocole *Oreste* est proposé dans le cadre de la plate-forme *GroupDesign* [Karsenty et al.93b]. Il a le profil suivant :

- Le modèle de la concurrence est celui des accèdes simultanés.
- Le modèle de la cohérence est la cohérence faible.

Oreste repose sur l'emploi d'un mécanisme de détection de la dépendance. La politique implantée par le protocole suit une démarche optimiste. La restauration de la cohérence des copies est faite d'une manière automatique. La mise en œuvre du protocole est faite sur un modèle de communication fiable. La réception des messages échangés entre les sites est fait dans un ordre quelconque.

Le principe de fonctionnement d'*Oreste* est le suivant. Chaque opération générée par un site est immédiatement exécutée sur le site local et en même temps envoyée aux autres sites. Chaque site maintient deux listes d'opérations : une liste d'historique d'exécution (H_i) et une liste d'opérations en attente d'exécution (Q_i). Seules les opérations qui s'appliquent à des objets non encore crées sur le site de réception sont mises en attente sur ce site. Des *horloges vectorielles* sont employées pour détecter la causalité entre les opérations générées dans la session [Raynal et al.96].

Lorsqu'un site envoie une opération, il envoie avec elle son horloge logique. À la réception d'une opération (Op) le site récepteur compare l'horloge logique de Op avec l'horloge locale. Trois cas de figure peuvent avoir lieu :

1. L'objet auquel s'applique Op n'est pas encore créé sur le site local (l'opération de création de l'objet n'est pas encore reçue). Op est alors ajoutée à la liste Q_i.
2. L'objet auquel s'applique Op n'existe plus. Op est alors ajoutée à l'historique de l'exécution (H_i) sans devoir être exécuté.
3. L'objet désigné par Op existe. Deux cas peuvent se présenter : l'horloge logique de Op est supérieure ou égale à l'horloge locale ; dans ce cas Op est immédiatement exécutée et ajoutée à H_i. Dans le cas contraire l'algorithme *défait* toutes les opérations exécutées sur le site local (dans H_i) et qui sont *plus récentes* que Op, exécute Op puis *refait* toutes les opérations défaites.

4.5 L'algorithme LICRA

4.5.1 Présentation informelle

Le profil Nous proposons dans la suite un nouveau protocole pour la gestion de la concurrence nommé *LICRA* (abréviation de Lock-free Interactive Concurrence Control Resolution Algorithm). Ce protocole implante un modèle de concurrence à accès simultanés et un modèle de cohérence causale. Il s'appuie sur un mécanisme de détection de dépendances. La mise en œuvre de *LICRA* repose sur une politique optimiste dont le déroulement est complètement automatique.

La conception de *LICRA* présente plusieurs originalités parmi lesquelles nous citons les deux points suivants. *LICRA* est le seul protocole qui combine les trois caractéristiques suivantes : a) permettre les accès simultanés (grâce à sa politique optimiste), b) réaliser un modèle de cohérence causale et c) être complètement automatique. Peu de protocoles existants implantent une politique optimiste. Les deux systèmes *Milan* et *Tivoli* [Condon92] implantent des protocoles optimistes où la restauration de la cohérence est à faire par les utilisateurs eux-mêmes (restauration manuelle). Le protocole optimiste *Oreste* (voir 4.4.2.4) est complètement automatique mais il implante un modèle de cohérence faible. Ainsi *LICRA* remplit un vide dans la grille des protocoles de gestion de la concurrence dans les collecticiels synchrones.

D'autre part, le mécanisme de la restauration automatique de la cohérence employé dans *LICRA* repose sur une procédure originale fondée sur le principe de *transformation d'opérations*. La transformation d'opérations évite aux utilisateurs la confusion que peut générer l'emploi d'un mécanisme de restauration automatique de la cohérence fondé sur le principe d'exécution réversible. L'exemple (4.1) illustre le principe de la transformation d'opérations tout en le comparant avec un mécanisme à exécution réversible.

Le principe de la transformation d'opérations a été introduit par *Ellis* et *Gibbs* dans leurs proposition d'un protocole, nommé *dOPT* [Ellis et al.89]. *dOPT* est la première *tentative* de développement d'un protocole de gestion de la concurrence qui soit optimiste, automatique et qui ne repose pas sur le principe d'exécution réversible. Malheureusement *dOPT* s'avère *incorrect* comme l'ont montré *Allison* et *Livesely* dans [Allison et al.94] (voir l'annexe B).

Exemple 4.1 Soit une session d'édition coopérative impliquant deux sites δ_1 et δ_2. Les deux sites manipulent une chaîne de caractères S. Au début S est vide. Le site δ_1 génère l'opération $Op_1 = \text{Insérer}(S, 1, A)$ et en même temps le site δ_2 génère $Op_2 = \text{Insérer}(S, 1, B)$. Pour fixer les idées nous supposons que δ_1 est prioritaire sur le site δ_2 ce qui implique que la séquence correcte d'application des deux opérations est la suivante : Op_1, Op_2. La valeur correcte de S est $S = BA$.

Cas d'un protocole optimiste à exécution réversible : Conformément à la nature optimiste du protocole chaque site exécute immédiatement l'opération qu'il

génère. Nous avons $S = A$ (respectivement $S = B$) sur δ_1 (respectivement δ_2). À la réception de Op_2 sur δ_1 le site exécute simplement Op_2 ce qui donne le résultat $S = BA$ sur ce site. À la réception de Op_1 sur δ_2 le site défait Op_2, exécute Op_1 puis exécute Op_2 pour avoir $S = BA$. L'utilisateur sur le site δ_2 n'a aucun moyen de savoir si toutes les trois dernières opérations sont faites par l'autre utilisateur ou si elles sont induites par le fonctionnement du protocole.

Cas d'un protocole optimiste à base de transformation d'opérations : Le scénario est le même que dans la cas précèdent sauf pour la traitement de la réception de Op_1 sur δ_2. Ici le site δ_2 **transforme** Op_1 en Op_1^t = Insérer$(S, 2, A)$ et il exécute l'opération transformée. Le résultat est le même que dans le premier cas mais avec moins de perturbation de l'utilisateur sur le site δ_2. Ce dernier voit en effet une seule mise à jour de son interface utilisateur suite à l'exécution d'une opération envoyée par l'autre utilisateur.

L'intérêt principal d'un mécanisme de transformation d'opérations est de cacher aux utilisateurs le fonctionnement interne du protocole *réduisant* ainsi la confusion que peut induire un modèle de concurrence à accès simultanés. La contrepartie d'un tel mécanisme est la nécessité de définir des rgles de transformation d'opérations ; une tâche parfois difficille et délicate.

Le fonctionnement Intuitivement *LICRA* fonctionne comme suit. À chaque opération est attribuée une étiquette qui la désigne d'une manière unique dans l'espace de la session. À la génération d'une opération Op le processus du site générateur exécute Op immédiatement et la transmet aux autres sites. Chaque opération envoyée est accompagnée d'une information dite *le contexte de génération*. Ce dernier définit l'état des objets du site lors de la génération de Op. Le contexte de génération d'une opération est constitué par l'ensemble des identificateurs des opérations non causalement liées entre elles et qui sont exécutées immédiatement avant la génération de Op. La procédure de calcul du contexte de génération d'une opération est donné dans la section (4.5.2.2).

À la réception d'une opération Op le site récepteur compare le contexte de génération de Op avec son propre *historique* d'exécution. Deux cas peuvent avoir lieu :

- Toutes les opérations référencées dans le contexte de génération de Op sont exécutées sur le site récepteur. L'opération Op peut alors être exécutée mais avant elle a éventuellement besoin d'être **transformée**. Deux raisons motivent la transformation d'une opération Op :

  1. Une ou plusieurs opérations référencées dans le contexte de génération de Op ont déjà subi des transformations sur le site récepteur. L'opération Op doit évidemment subir les mêmes transformations.

2. Le site récepteur a exécuté de nouvelles opérations après l'exécution des opérations référencées dans le contexte de génération de Op. Par conséquent Op doit être transformée afin de prendre en compte les effets d'exécution de ces opérations.

- Il existe une opération référencée dans le contexte de génération de Op qui n'est pas encore reçue **ou** qui est en attente d'exécution. Nous disons que Op arrive en avance ; elle est par conséquent mise en attente.

Les deux règles précédentes assurent la mise à jour des copies selon l'ordre causal ; une opération n'est exécutée sur un site que si toutes les opérations qui la précèdent causalement sont déjà exécutées sur ce site. Le fonctionnement du *LICRA* décrit ci-dessus révèle le besoin de garder sur chaque site l'historique d'exécution locale. L'historique étant une structure de données dont la taille est croissante avec le temps, il risque de saturer la mémoire disponible sur le site. Ce problème est d'ailleurs commun à tous les protocoles optimistes et automatiques. Un mécanisme de réduction de la taille de l'historique est prévu. Nous explicitons dans la section suivante le schéma de mise en œuvre du protocole proposé.

4.5.2 Mise en œuvre

4.5.2.1 Hypothèses de travail

Afin de simplifier la présentation de l'algorithme nous faisons les hypothèses suivants :

H_1 Le nombre de sites impliqués dans la session est constante.

H_2 La communication entre les différentes sites est faite selon un protocole fiable.

Nous montrons dans la section (4.5.6) comment traiter le cas de la connexion et déconnexion dynamiques de sites ainsi que le traitement de certain types de pannes.

4.5.2.2 Définitions

Nous définissons dans la suite les principaux concepts employés dans le protocole *LICRA*.

Une opération (Op) est définie par un quadruple $Op = (Site, \text{Séquence}, \phi, \Pi)$ où $Site$ est l'identificateur du site générateur de l'opération, Séquence est un compteur du nombre d'opérations générées par le site générateur, ϕ est la fonction appliquée par l'opération et Π est l'ensemble des paramètres de la fonction ϕ. Nous utilisons la notation $Op.x$ pour désigner le champ x d'une opération ; ainsi l'identificateur du site générateur d'une opération est désigné par $Op.Site$.

Les deux premiers champs constituent l'étiquette (ou l'identificateur) universelle de l'opération. Nous utilisons la notation $Op.ID$ pour designer le couple $Op.Site$ et Op.Séquence. Toute opération garde le long de la session la même étiquette attribuée par son site générateur. La transformation d'une opération agit sur les deux champs ϕ et Π.

Une liste d'opérations. Chaque site δ_i maintient deux types de listes d'opérations : une liste d'historique H_i et des **listes** d'attente. Une liste d'opérations L est constituée par une suite d'éléments ; un élément décrit une opération reçue sur le site local. Nous détaillons la structure d'un élément dans la section (4.5.3.1). Nous avons besoin à présent de définir trois fonctions principales de manipulation de listes qui sont nécessaires pour la définition des concepts cités ci-après. Les trois fonctions sont les suivantes :

$Succ(Op, L)$; Cette fonction renvoie l'élément qui suit Op (l'élément décrivant Op) dans la liste L. La fonction $Succ^*(Op, L)$ renvoie tous les éléments qui suivent Op dans L.

$Pred(Op, L)$; Cette fonction renvoie l'élément qui précède Op dans L. La fonction $Pred^*(Op, L)$ renvoie tous les éléments qui précèdent Op dans L.

$Last(L)$ Cette fonction renvoie le dernier élément dans L.

Le contexte de génération (C_xG) À la génération d'une opération Op le site générateur calcule le contexte de génération de Op selon l'algorithme suivant :

Algorithme 4.1 *CxG (Op);*

// Le site local est le site δ_i;
CxG = nil; // initialisation du contexte de génération
*Op_l = **Last** (H_i); // Op_l est la dernière opération exécutée avant la génération de Op*
***Si** ($Op_l.Site == i$) **Alors** // Op_l est générée par le site local*
    $CxG = Op_l.ID$
    ***return**(CxG);*
Sinon
    $CxG = Op_l.ID$
    *Op_x = **Pred** (Op_l , H_i);*
    ***Tant que** ($Op_x.CxG == Op_l.CxG$) **Faire** // Les deux opérations sont concurrentes;*
        $CxG = CxG + Op_x.ID$
        $Op_l = Op_x$
        *Op_x = **Pred** (Op_l , H_i);*

Un message (M). Les sites échangent entre eux les opérations qu'il génèrent sous forme de messages. Un message M est constitué par un couple $M = (Op, C_xG(Op))$, où Op est l'opération émise et $C_xG(Op)$ est son contexte de génération.

Le contexte de production (C_xP) Il contient les opérations exécutées sur le site récepteur après l'exécution de la dernière opération précédente l'opération Op. Soit L la liste contenant l'opération Op sur un site δ. Le contexte de production de Op sur δ est alors définie par l'expression suivante : $C_xP(Op) = Succ^*(Op, L)$.

Le contexte d'adaptation (C_xA) Le contexte d'adaptation d'une opération désigne l'ensemble des opérations en fonction desquelles une opération *reçue* sur un site doit être transformée. Le calcul du contexte d'adaptation est fait selon le formule suivant :

$$C_xA(Op) = [(\cup_{X \in C_xG(Op)} C_xA(X))/C_xG(Op)] \cup C_xP(Last(C_xG(Op)))$$

Derrière cette expression complexe se cache une définition simple : tout opération ayant servi pour transformer une opération qui précède Op doit servir pour transformer Op elle même. De l'ensemble précèdent nous excluons les opérations qui appartiennent déjà au contexte de génération de Op car l'effet de ces opérations est déjà prise en compte par Op. Puis nous ajoutons au contexte d'adaptation toute opération exécutée sur le site après l'exécution de la dernière opération précédant Op. L'exemple suivant illustre les différents concepts définis ci-dessus.

Exemple 4.2 *Illustration des concepts de contextes de génération, d'adaptation et de production*

La figure (4.3) illustre un échange de cinq opérations entre trois sites dans le cadre d'une session coopérative.

La tableau suivant donne le contexte de génération de chacune des cinq opérations. Le calcul de chacun des G_xG est immédiat selon l'algorithme cité ci-dessus.

Pour illustrer les deux concepts de contexte de production et d'adaptation nous prenons l'exemple de l'historique de l'exécution sur le *site3*. L'historique de l'exécution sur ce site est composé de cinq étapes :

1. **Génération de Op_2 :** Op_2 étant une opération locale elle est exécutée immédiatement. La liste d'historique devient $H_3 = (Op_2)$.
2. **Réception de Op_1 :** Nous avons $C_xG(Op_1) = nil \Rightarrow C_xA(Op_1) = (C_xA(nil)/C_xG(Op_1)) + C_xP(nil) = nil + Op_2 = Op_2$. L'opération Op_1 est transformée en fonction de Op_2. L'historique local devient $H_3 = (Op_2, Op_1)$.
3. **Réception de Op_3 :** Nous avons $C_xG(Op_3) = (Op_1, Op_2) \Rightarrow C_xA(Op_3) = ((C_xA(Op_1) + C_xA(Op_2))/(Op_1, Op_2)) + C_xP(Op_1) = nil + nil = nil$. Alors l'opération Op_3 est exécutée sans modification. Nous avons $H_3 = (Op_2, Op_1, Op_3)$.

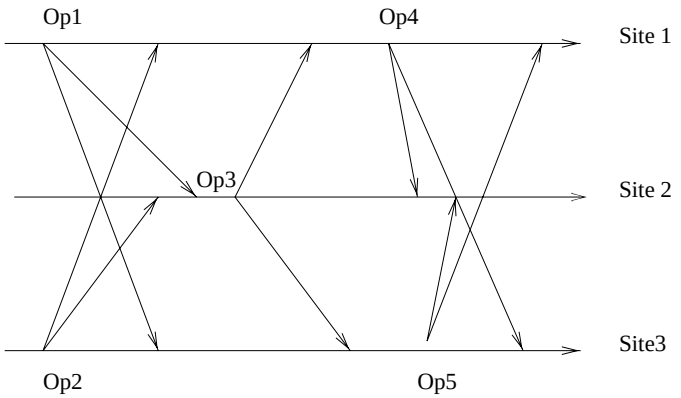


FIG. 4.3 - : Exemple d'une échange d'opérations entre trois sites.

Opération	C_xG
Op_1	-
Op_2	-
Op_3	$[Op_2, Op_1]$
Op_4	Op_3
Op_5	Op_3

TAB. 4.2 - : Les contextes de génération des opérations illustrées sur la figure (5.3)

4. **Génération de Op_5 :** Op_5 étant une opération locale elle est exécutée immédiatement. La liste d'historique devient $H_3 = (Op_2, Op_1, Op_3, Op_5)$.

5. **Réception de Op_4 :** Nous avons $C_xG(Op_4) = (Op_3)$, alors $C_xA(Op_4) = C_xA(Op_3)/(Op_3) + C_xP(Op_3) = nil + Op_5 = Op_5$. L'opération Op_4 est transformée en fonction de Op_4 puis rajoutée à l'historique.

4.5.2.3 L'adaptation d'une opération

La procédure d'adaptation d'une opération consiste à *transformer* cette opération en fonction de chacune des opérations contenues dans son contexte d'adaptation. Le principe de la transformation d'opération est le suivant : étant donné deux opérations Op_i et Op_j non causalement liées entre elles (donc générées par deux sites différents) ; deux sites peuvent recevoir ces deux opérations dans un ordre différent ; c'est notamment le cas des deux sites générateurs. Tout site ayant exécuté Op_i (respectivement Op_j) avant la réception de Op_j (respectivement Op_i) transforme Op_j (respectivement Op_i) dès sa réception en Op_j^t (respectivement en Op_i^t) de sorte que la condition suivante soit vraie :

$$Op_j^t \circ Op_i \;=\; Op_i^t \circ Op_j$$

Il est clair que le calcul de l'opération transformée dépend de la sémantique des deux opérations impliquées et nécessite aussi de définir une fonction de priorité entre les deux opérations. La fonction de priorité est nécessaire pour imposer un ordre de sérialisation des opérations. Pour cela nous définissons une fonction de priorité entres les sites participants à la session. La fonction de priorité est simplement l'ordre naturel définie sur l'ensemble des identificateurs (nombre entiers) des sites. Une opération prend la priorité de son site générateur. L'algorithme suivant résume la procédure d'adaptation d'une opération. Dans les procédures suivantes nous avons à transformer l'opération Op_1 qui arrive sur un site en fonction de Op_2 déjà exécutée sur le même site.

Algorithme 4.2 *Adaptation* $(Op, Op.C_xA)$*;*
Soit L = *Nombre d'éléments dans le contexte d'adaptation de* Op*;*
Pour $(i = 1 ; i++ ; i \leq L)$ ***Faire***
$Op =$ ***Transforme*** $(< Op , Op.Site > , < Op.C_XA[i].Op, Op.C_XA[i].Priorit >)$*;*

Exemple 4.3 *Définition de la fonction de transformation d'opérations - cas d'un éditeur de textes* Nous considérons dans cet exemple un éditeur de texte où le document édité est une simple chaîne de caractères. L'éditeur fournit les deux opérations suivantes :

- Insérer(Position,Lettre) : insérer le lettre Lettre dans la position Position. Par exemple l'application de Insérer (3 , 'L') sur la chaîne "COT" donne la chaîne "COLT".
- Supprimer(Position) : supprime la lettre à la position Position. Par exemple l'application de Supprimer(4) sur la chaîne "COLT" donne la chaîne "COL".

Nous citons dans la suite les règles de transformation à appliquer dans chacun des quatre cas possibles de croisement d'opérations. À chaque opération est associé une valeur P qui correspond à la priorité attribuée à l'opération.

Algorithme 4.3 *Transformer* $(Op_1 =< Ins(Pos_1, X), P_1 > , Op_2 =< Ins(Pos_2, Y), P_2 >)$*;*

Cas $P_1 < P_2$ ***Faire*** *// Op_1 est prioritaire à Op_2;*
Si $Pos_1 < Pos_2$ ***Alors*** $Op_1^{'} = Op_1$*; exit;*
Si $Pos_1 == Pos_2$ ***Alors*** $Op_1^{'} = Ins(Pos_1 + 1, X)$*; exit;*
Si $Pos_1 > Pos_2$ ***Alors*** $Op_1^{'} = Ins(Pos_1 - 1, X)$*; exit;*
Cas $P_1 > P_2$ ***Faire*** *// Op_2 est prioritaire à Op_1;*
Si $Pos_1 \leq Pos_2$ ***Alors*** $Op_1^{'} = Op_1$*; exit;*
Si $Pos_1 > Pos_2$ ***Alors*** $Op_1^{'} = Ins(Pos_1 - 1, X)$*; exit;*

Algorithme 4.4 *Transformer* $(Op_1 =< Ins(Pos_1, X), P_1 > , Op_2 = Sup(Pos_2) , P_2 >)$*;*

Cas $P_1 < P_2$ ***Faire*** *// Op_1 est prioritaire à Op_2;*
Si $Pos_1 \leq Pos_2$ ***Alors*** $Op_1^{'} = Op_1$*; exit;*
Si $Pos_1 > Pos_2$ ***Alors*** $Op_1^{'} = Ins(Pos_1 - 1, X)$*; exit;*

Cas $P_1 > P_2$ ***Faire*** *// Op_2 est prioritaire à Op_1;*
Si $Pos_1 \leq Pos_2$ ***Alors*** $Op_1^{'} = Op_1$*; exit;*
Si $Pos_1 > Pos_2$ ***Alors*** $Op_1^{'} = Ins(Pos_1 - 1, X)$*; exit;*

Algorithme 4.5 *Transformer* $(Op_1 =< Sup(Pos_1), P_1 >, Op_2 = Ins(Pos_2, Y), P_2 >)$*;*

Si $Pos_1 < Pos_2$ ***Alors*** $Op_1^{'} = Op_1$*; exit;*
Si $Pos_1 \geq Pos_2$ ***Alors*** $Op_1^{'} = Sup(Pos_1 + 1)$*; exit;*

Algorithme 4.6 *Transformer* $(Op_1 =< Sup(Pos_1), P_1 >, Op_2 = Sup(Pos_2), P_2 >)$*;*

Si $Pos_1 < Pos_2$ ***Alors*** $Op_1^{'} = Op_1$*; exit;*
Si $Pos_1 == Pos_2$ ***Alors*** $Op_1^{'} = NOP$*; exit;*
Si $Pos_1 \leq Pos_2$ ***Alors*** $Op_1^{'} = Sup(Pos_1 - 1)$*; exit;*

Exemple 4.4. *Transformation d'opération - cas de l'application ColtDraw*
ColtDraw est l'éditeur graphique fournit par l'environnement de coopération Colt. À la différence d'un éditeur de texte les opérations dans un éditeur graphique sont généralement commutatives ce qui simplifie la définition des règles de transformation. À titre d'exemple nous illustrons dans la suite les règles de transformation concernant le couple d'opérations suivant : `SetFillColor(Objet,couleur)` et `Supprimer(Objet)`.

Algorithme 4.7 *Transformer*$(Op_1 =< SetFillColor(o_1, C_1), P_2 >, Op_2 =< SetFillColor(o_2, C_2), P_2 >)$*;*

// Les deux opérations s'appliquent à des objets différents;
Si $o_1 \neq o_2$ ***Alors***
$Op_1^{'} = Op_1$*;*
exit
Sinon *// Les deux opérations s'appliquent au même objet*
Si $P_1 < P_2$ ***Alors*** $Op_1^{'} = Op_1$ ***Sinon*** $Op_1^{'} = NOP$*;*

Pour simplifier la présentation des autres règles de transformation nous ne considérons que le cas où les deux opérations s'appliquent sur le même objet graphique. Dans le cas contraire l'opération à transformer reste telle qu'elle est.

Algorithme 4.8 *Transformer* $(Op_1 =< SetFillColor(o_1, C), P_1 <, Op_2 = Sup(o_1), P_2 >)$*;*

$Op_1^{'} = NOP$*;*

Algorithme 4.9 *Transformer* $(Op_1 =< Sup(o_1), P_1 >, Op_2 =< SetFillColor(o_1, C), P_2 >)$*;*

$Op_1^{'} = Op_1$*;*

Les deux précédents exemples montrent que la complexité de la définition des règles de transformation dépend fortement de la sémantique de l'application. Dans certain cas il suffit de savoir les commandes appliquées pour avoir l'opération transformée ; dans d'autres il faut aller jusqu'à l'analyse des paramètres des opérations pour trouver le résultat de la transformation. Dans tous les cas, pour une application fournissant n opérations il faut écrire n^2 règles de transformations ; certaines sont plus faciles à formuler que d'autres.

4.5.3 Description de l'algorithme

4.5.3.1 Structures de données

Chaque site δ_i maintient les structures des données suivantes :

H_i est la liste d'historique d'exécution locale.

WOP est la liste des identificateurs des opérations attendues sur le site. Une opération Op_A est attendue sur un site si celui-ci a reçu une opération Op où $Op_A.ID \in C_cG(Op)$ et Op_A n'est pas encore reçue.

(L_i) est un ensemble de listes d'opérations en attente d'exécution.

V est un vecteur dont la dimension est le nombre de sites participants à la session. Une entrée $V[n]$ porte l'identificateur de la dernière opération reçue de la part du site δ_n. Le contenu de V servira à réduire la taille de la liste d'historique H_i évitant ainsi sa croissance monotone.

T_1 , T_2 sont deux horloges périodiques. Tous les T_1 instants le site local envoie à tous les autres sites un événement dont l'opération porte la fonction NOP. La fonction NOP n'affecte pas l'état des objets dans O_i. L'horloge T_1 garantit que chaque site reçoit au moins un message de tout autre site tous les T_1 instants. Ceci sert à pouvoir réduire la taille de H_i. La réduction effective du H_i est faite tous les T_2 instants.

4.5.3.2 Déroulement du protocole

Chaque site exécute le même algorithme. Pendant le déroulement d'une session deux événements peuvent arriver à un site : 1) la génération d'une opération, 2) la réception d'une opération. Nous décrivons dans la suite le comportement de l'algorithme dans chacun des deux cas précédents.

Cas de génération d'une opération Op Le site exécute Op immédiatement, prépare le message $M = (Op, Op.C_xG)$ puis l'envoie à tous les autres sites. Il ajoute aussi un nouvel élément décrivant Op à la fin de la liste de l'historique H_i.

Cas de réception d'un message M À la réception d'un message M, deux cas peuvent avoir lieu :

1. Il existe une opération Op_r référencée dans $M.C_xG$ qui ne soit pas encore reçue sur le site local. Dans ce cas le processus du site crée une nouvelle liste d'attente dont la tête est Op. L'opération Op_r est déclarée comme une opération attendue ; son identificateur est ajouté à la liste des opérations attendues WOP.

2. Toutes les opérations référencées dans $M.C_xG$ sont déjà reçues. Elle sont forcément stockées dans une même liste L. L peut être la liste d'historique H_i ou une liste d'attente. Dans les deux cas, le processus du site calcule le contexte d'adaptation de Op. L'opération Op est adaptée puis ajoutée à la fin de L. Si L est la liste d'historique, Op est exécutée sur le site sous sa forme transformée.

Dans les deux cas, après le traitement d'une opération reçue Op, le processus du site vérifie si l'opération Op est une opération attendue. Si c'est le cas, il existe forcément une liste qui contient les opérations en attente de Op. Le site est en mesure maintenant de traiter ces opérations. La liste d'attente est supprimée après le traitement de toutes les opérations qu'elle contient. L'identificateur de *op* est supprimé de la liste des opérations attendues : WOP.

Réduction de l'historique H_i Le principe est le suivant. Chaque site garde un vecteur de références V dans lequel l'entrée $V[k]$ porte l'identificateur de la dernière opération reçue de la part du site δ_k. À l'expiration d'une horloge périodique T_2, chaque site consulte le vecteur V pour déterminer l'opération Op_a : l'opération la plus ancienne dans H_i référencée dans V. Soit Op_A l'opération la plus ancienne dans le contexte d'adaptation de Op_a. Op_A est forcément dans H_i puisque $Op_a \in H_i$. Toute opération avant Op_A peut être supprimée puisque aucune *adaptation* ultérieure ne peut s'en servir.

Un problème que peut rencontrer l'algorithme précèdent est le problème du site inactif. Un site est inactif s'il se contente de recevoir des messages sans en produire. Par conséquent aucune réduction sur aucun site ne peut s'effectuer car du point de vue des autres sites, le site inactif est toujours soupçoné de produire une opération qui dépend de l'origine de l'historique de la session. Une première solution consisterait à envoyer systématiquement accusé de réception des événements reçus mais ceci mènerait aussi à une dégradation des performances du système en augmentant le nombre de messages à véhiculer. La solution adoptée est de forcer chaque site à diffuser d'une manière périodique un événement qui porte l'opération NOP. Cet événement porte ainsi l'identificateur de la dernière opération reçue **et** exécutée sur ce site.

4.5.4 Exemples

Nous présentons dans cette section quelque exemples d'applications du protocole *LICRA*. Dans l'ensemble des exemples présentés ci-dessous la fonction de priorité entre les sites est simplement la relation d'ordre normal sur les identificateurs des sites ; le site δ_1 est prioritaire sur le site δ_2 et ainsi de suite.

4.5.4.1 Échange simultané entre trois sites

Dans ce premier exemple nous considérons le cas d'échanges simultané de trois opérations entre trois sites. L'ordre de la réception des opérations sur chaque site est illustré à la figure (4.4).

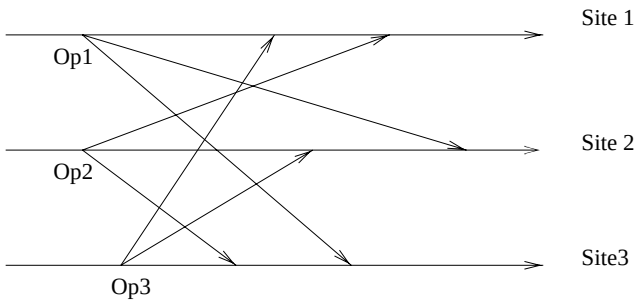


FIG. 4.4 - : Échange simultané entre trois sites.

Pour fixer les idées, nous considérons le cas d'édition d'une chaîne de caractères S. Au début la chaîne est vide. Nous considérons les opérations suivantes :

- Op_1 = `Insérer(1,A);`
- Op_2= `Insérer(1,B);`
- Op_3= `Insérer(1,C);`

Nous traçons dans la suite l'exécution des trois opérations sur les trois sites.

Exécution sur le site δ_1 L'opération Op_1 étant locale elle est exécutée immédiatement. Nous avons $S = A$ et $H_1 = (Op_1)$. Ensuite le site reçoit Op_3 dont le contexte de génération est l'*origine*, par conséquence le contexte d'adaptation de Op_3 est constitué par l'historique du site ; nous avons $C_xA(Op_3) = (Op_1)$. Or, d'après les règles de transformation l'opération Op_3 se transforme en opération identique. Nous avons $S = CA$ et $H_1 = (Op_1, Op_3)$. Enfin le site reçoit Op_2 dont le contexte de génération est l'opération origine. Elle doit être transformée en fonction de l'historique local, c'est à dire en fonction de Op_1 et Op_3. La transformation de Op_2 en fonction de Op_1 laisse l'opération Op_2 intacte tandis que la transformation en fonction de Op_3 la change en $Op_2^{'}$ = Insérer$(2, B)$. Nous avons alors $S = CBA$ et $H_1 = (Op_1, Op_3, Op_2^{'})$.

Exécution sur le site δ_2 Op_2 est d'abord exécutée. Nous avons $S = B$ et $H_2 = (Op_2)$. À la réception de Op_3 le site la transforme en fonction de Op_2 ; le site δ_3 est moins prioritaire que δ_2 l'opération Op_3 reste intacte conformément aux règles de transformation décrites ci-dessus. Nous avons maintenant $S = CB$ et $H_2 = (Op_2, Op_3)$. Lorsque le site reçoit Op_1, il la transforme en fonction de Op_2 et Op_3. L'opération Op_1 est transformée en $Op_1^{'}$ = Insérer$(3, A)$. Nous avons alors $S = CBA$.

Exécution sur le site δ_1 D'une manière similaire, aux autres sites, le site δ_3 exécute d'abord Op_3 transforme Op_2 en $Op_2^{'}$ = Insérer$(2, B)$ et l'exécute. Puis transforme Op_1 en $Op_1^{'}$ = Insérer$(3, A)$ et l'exécute. Le contenu de S passe du vide à B puis à CB en enfin à CBA.

4.5.4.2 Cas de concurrence partielle

Nous considérons ici un cas particulier d'échanges entre deux sites dit échange en concurrence partielle. Un ensemble d'opérations est dit en concurrence partielle si une opération est en concurrence avec une série d'opérations toutes causalement liées elles. Le cas le plus simple d'une situation de concurrence partielle est illustré à la figure (4.5).

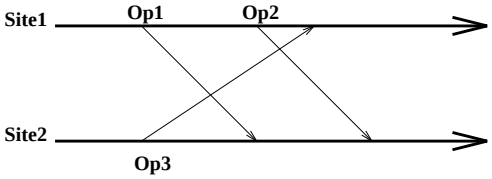


FIG. 4.5 - : Situation de concurrence partielle.

L'intérêt d'étudier le comportement de *LICRA* dans une situation de concurrence partielle est de montrer le succés du protocole là où le seul autre protocole à base de transformation d'opérations (le protocole *dOPT*) est montré incorrecte [Allison et al.94] (voir aussi annexe B). Pour fixer les idées considérons le cas d'une session impliquant deux sites qui utilisent l'application *ColtDraw*. Prenons le cas où $Op_1 = SetFillColor(o_1, Vert), Op_2 = SetFillColor(o_1, Bleu)$ et $Op_3 = SetFillColor(o_1, Rouge)$.

Exécution sur le site δ_1 Les deux opérations Op_1 et Op_2 sont exécutées séquentiellement. L'objet o_1 a la couleur bleue. À la réception de Op_3 le site l'adapte en fonction de deux opérations précédemment exécutées. Selon les règles de la transformation décrites dans l'exemple (5.4) l'opération Op_3 reste intacte ; à son exécution l'objet o_1 devient rouge. Les copies des données restent ainsi cohérentes entre elles. Dans l'annexe B, nous reprenons le même exemple pour montrer l'incorrection du protocole *dOPT*.

Exécution sur le site δ_2 L'opération Op_3 est exécutée immédiatement. L'objet o_1 a la couleur rouge. À la réception de Op_2 le site la transforme en fonction de Op_1. Le site δ_1 est plus prioritaire donc Op_2 est transformé en $Op_2^{'} = NOP$ (voir les règles de transformation données dans l'exemple 5.2). Le contexte d'adaptation de Op_2 sur le site δ_1 contient l'opération Op_1. Par conséquence Op_2 est transformée à son tour en NOP. L'objet o_1 reste en rouge sur le site δ_1.

4.5.5 Preuve de correction

Soit S une session de coopération synchrone. Le protocole *LICRA* étant fondé sur un modèle de communication fiable tous les sites dans S vont recevoir toutes les opérations générées dans la session ; l'ordre de la réception des opérations peut différer d'un site à un autre. *LICRA* est correct si quelque soit l'ordre de la réception des opérations sur les différents sites les deux conditions suivantes sont satisfaites :

C1. Tous les sites exécutent les opérations générées dans la session dans leurs ordre causal.

C2. Aux moments de repos les objets de tous les sites sont identiques.

La première condition est vérifiée par la construction même du protocole : une opération ne peut pas être exécutée sur un site (avec ou sans transformation) sauf si toutes les opérations qui la précédent sont déjà exécutées sur ce site. Il nous reste donc à prouver que *LICRA* satisfait la deuxième condition. Pour cela prenons deux sites quelconque δ_x et δ_y. Soit $G = (Op_1, Op_2, \ldots, Op_m)$ une séquence d'opération générées dans la session. Nous désignons par G_x (respectivement G_y) l'ordre de la réception de la séquence G sur δ_x (respectivement δ_y). G étant un ensemble fini d'opérations il est donc possible d'obtenir G_y en appliquant un nombre fini de *permutations* sur G_x. Par conséquent il nous suffit de montrer la véracité de la condition C2 dans le cas où les deux séquences G_x et G_y sont identiques à une permutation près. Soient Op_i et Op_j les deux opérations qu'il faut permuter dans G_x pour obtenir G_y. Deux cas peuvent avoir lieu : 1) une de deux opérations dépend de l'autre et 2) les deux opérations sont en concurrence.

Examinons d'abord le premier cas. Pour fixer les idées supposons que Op_j dépend de Op_i. Les deux sites exécutent la séquence dans G_x (ou G_y) qui précède Op_i de la même manière puisque cette séquence est identique sur les deux sites. À la réception de Op_j sur δ_y, l'opération va être mises en attente car il existe une opération qui la précède et qui n'est pas encore reçue (Op_i). Soit Op une opération appartenant au segment qui sépare Op_j de Op_i dans G_y. Deux cas sont possibles : soit Op dépend de Op_i ou soit elle est en concurrence avec elle. Dans le premier cas Op est ajoutée à la file des opérations en attente d'exécution. Dans le deuxième cas l'opération est traitée sur le site δ_y sans avoir subi les transformations qu'elle du subir en fonction de Op_i. Mais selon la définition même de la fonction de

l'adaptation lorsque le site δ_y reçoit Op_i elle la transforme en fonction de Op de sorte que son résultat d'exécution soit identique à l'exécution de Op transformée en fonction Op_i sur le site δ_x. Ce même raisonnement s'applique pour le cas de la concurrence entre Op_i et Op_j en remplaçant Op par Op_j. et Op_i en Op_j

4.5.6 Amélioration de l'algorithme

Nous présentons dans cette section quelque possibilités d'amélioration de l'algorithme *LICRA*. Nous étudions en particulier les points suivants :

- Simplifier la procédure d'adaptation d'une opération.
- Étendre l'algorithme afin de supporter la participation dynamique de nouveaux arrivants.
- Modifier l'algorithme pour pouvoir tolérer certains types de pannes de sites ou de perte de messages.

Les idées discutées ci-après ne sont pas implantées dans l'actuel protocole. Leur implantation est planifiée pour une nouvelle version du *LICRA*.

1. Simplifier l'adaptation Nous proposons dans la suit de simplifier la procédure d'adaptation en simplifiant : 1) la définition de la matrice de transformation d'opérations et 2) en simplifiant le parcours que fait un opération de son contexte d'adaptation.

Dans cette optique nous définissons la relation de commutation entre *fonctions* de la façon suivante : une fonction ϕ_1 commute avec une fonction ϕ_2 si l'application de deux fonctions dans les deux ordres possibles donne le même résultat. Deux opérations commutent si elles utilisent des fonctions qui commutent entre elles. Pendant l'adaptation d'une opération Op, on peut alors sauter immédiatement tout opération dans la contexte de l'adaptation qui commute avec Op. Une fonction particulière qui commute avec tout autre fonction est la fonction NOP. Lorsqu'une opération Op est transformée en NOP pendant son adaptation nous pouvons sauter immédiatement le reste du contexte de l'adaptation et terminer la procédure. La procédure d'adaptation devient la suivante :

Algorithme 4.10 *Adapter* $(Op, C_xA(Op))$;
Soit $Op_x = tête(C_xA(Op))$;
Tant que $Op_x.ID \neq nil$ ***Faire***
Cas $Commute(Op.\phi\, Op_x.\phi)$ ***Alors*** $Op_x = Suivant(Op_x, Op.C_xA)$;
Cas $Masquée(Op.\phi\, Op_x.\phi)$ ***Alors***
$Op.\phi = NOP$;
Ajouter(Op,L); // $Op.C_xA \subset L$
exit;

Défaut *// l'opération doit être transformée*
$Op = Transformer(Op, Op_x)$;
$Op_x = Suivant(Op_x, Op.C_xA)$;

2. Supporter la participation dynamique de nouvels sites Un service de gestion de la participation dynamique comprend deux volets :

1. un volet décisionnel correspondant à l'application d'une stratégie de contrôle d'appartenance,

2. et un volet technique correspondant au transfert au site du nouvel arrivant du contexte actuel de la session, une fois que la demande de connexion est approuvée.

Dans cette section, nous nous intéressons au deuxième volet uniquement. Deux solutions sont envisagées pour que *LICRA* devienne tolérant à la participation dynamique des sites.

La première solution consiste à *geler* la session et à forcer par conséquent un état de repos lors de la réception d'une requête de connexion. Une fois que le système est au repos, n'importe quel site peut envoyer au nouvel arrivant une copie de son état (données et historique). Cette approche est simple à mettre en œuvre. Cette solution est implantée dans le prototype actuel de l'environnement *Colt*. Cependant forcer l'état de repos ralentit le système. La session est gelée à chaque demande de connexion pour le temps nécessaire pour transmettre l'état courant. Ce temps peut être important si les données manipulées dans une session ont une taille importante.

Une deuxième solution qui permet la participation dynamique de nouveaux arrivants sans interrompre la session est alors proposée. Le principe de l'algorithme est le suivant. Soit δ_N le site du nouvel arrivant. Pour rejoindre une session, le site envoie une requête de participation à tous les membres actuels de la session. Un site est choisi parmi les sites existants pour jouer le rôle du *Parrain* du δ_N. Le parrain, dénoté δ_P, a la tâche de transférer son état au nouvel arrivant et de faire suivre les messages qu'il reçoit des autres sites à δ_N. Le site δ_N exécute le même algorithme *LICRA* à la différence près qu'il ne peut pas lui même générer des opérations. Nous disons que le site est en état *semi actif*. Tant que le site n'a pas reçu une copie de l'état du site du parrain, aucune opération reçue d'un autre site ne peut être exécutée. Par contre à la réception d'une opération d'un autre site δ_N envoie un événement d'acquitement à δ_P pour lui informer de l'établissement d'une liaison avec ce site. L'événement d'acquitement porte l'identificateur de l'opération reçue. Si cette opération est déjà reçue par le site parrain, ce dernier arrête la poursuite des événements (opérations) originaires du site acquitté et en relation avec δ_N. Lorsque le parrain arrête les envois poursuite de tous les autres sites il envoie un événement de fin de parrainage. À la réception de cet événement le site δ_N devient un site participant à part entière.

La figure (4.6) illustre le déroulement de l'algorithme pour un site qui participe à une session contenant deux autres sites. Remarquer que le site δ_N peut recevoir certaines opérations en double ; une fois par le site générateur et une deuxième fois par le biais du *parrain*. C'est le cas par exemple de l'opération Op_3 qui est reçue sur le site parrain avant que celui-ci ne reçoive l'acquitement du δ_N qu'il est en liaison avec le troisième site. La détection d'une réception redondante est triviale grâce à l'unicité des identificateurs des opérations.

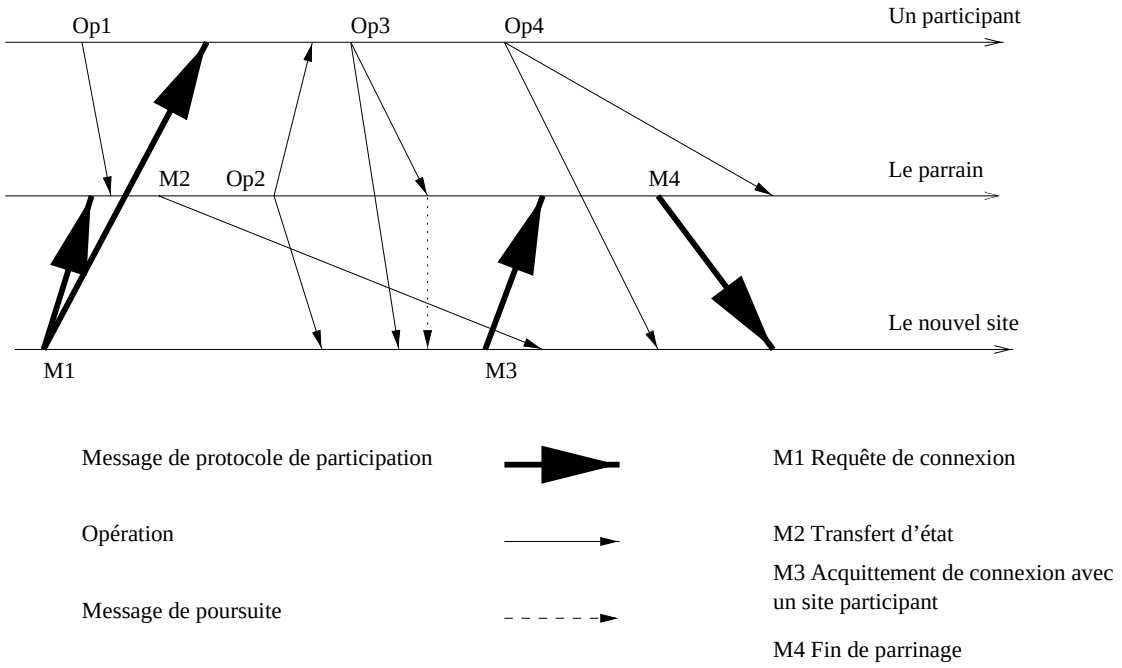


FIG. 4.6 - : Exemple de déroulement de l'algorithme de la connexion dynamique.

Tolérance aux pannes *LICRA* tel qu'il est présenté ci-dessus est fondé sur un modèle de communication fiable et à ordre *FIFO*. Nous montrons dans cette section comment relâcher davantage les contraintes sur le modèle de communication. Nous décrivons dans la suite comment l'algorithme *LICRA* peut être adapté pour faire face à certains types de pannes et de perte de messages. Nous examinons en particulier les deux cas suivants : 1) panne transitoire d'un lien de communication 2) et panne d'un site.

Panne transitoire d'un lien de communication Nous considérons ici le cas de la perte d'une opération envoyée par un site suite à une panne transitoire du réseau de communication. Pour fixer les idées, supposons que l'opération Op_e envoyée par le site δ_e n'a pas atteint un autre site δ_r. La panne étant transitoire, le site δ_r va forçément recevoir une opération Op_f qui elle dépend directement de Op_e. Noter que l'opération Op_f n'est pas nécessairement générée par le même site

δ_e. À la réception de Op_f par δ_r, ce dernier, conformément à *LICRA* va déclarer l'opération Op_e comme opération attendue (ajouter son identificateur à la liste WOP). Pour chaque opération attendue le site associe *une période d'expiration* (Time out) au bout de laquelle il réclame la retransmission de cette opération. Ainsi le site δ_r va demander au site émetteur de Op_f de lui transmettre l'opération Op_e. Le site générateur de Op_f a une copie de Op_e parce qu'il a généré Op_f.

Panne d'un site Nous considérons ici le cas de panne en silence d'un seul site. Considérons le cas où le site δ_e est tombé en panne immédiatement après l'émission d'une opération Op_f. Supposons par ailleurs que $Op_e \mapsto Op_f$. Trois cas peuvent avoir lieu :

- *L'opération Op_f n'est reçue par aucun site* : dans ce cas Op_f est ignorée.
- *Tous les sites ont reçu Op_f* : dans ce cas trois cas de figure peuvent avoir lieu :
  - Aucun site n'a reçu l'opération Op_e, dans ce cas l'opération Op_f est ignorée.
  - Tous les sites ont reçu Op_e. Dans ce cas l'opération Op_f peut être exécutée normalement.
  - Quelque sites ont reçu Op_e. Dans ce cas les sites qui ne l'ont pas reçu vont demander à un site qui a déjà Op_e de la retransmettre. Ceci est fait par diffusion d'un requête de retransmission de Op_e après l'expiration de la période l'attente.
- *Quelques sites ont reçu Op_f* : les sites qui reçoivent Op_f vont le traiter comme dans le cas précédent. Si l'opération Op_f n'est pas ignorée, un site qui ne l'a pas reçu va fatalement recevoir une opération Op_j de sorte que $Op_e \mapsto Op_j$; alors un tel site va recevoir Op_f comme c'est indiqué dans le cas précédent.

Cette extension de l'algorithme *LICRA* permet de garantir que tous les sites non défaillants vont pouvoir continuer à travailler malgré la défaillance d'un site. Le problème de tolérance aux pannes dans sa généralité est un problème complexe. Mais nous pensons que l'algorithme *LICRA* se prête bien à traiter plusieurs cas de pannes à cause de la décentralisation totale de son fonctionnement et à l'échange périodique des opérations NOP tous les T_1 instants. Les événements périodiques NOP peuvent être utilisés pour détecter l'occurrence des pannes des sites ou de réseau. Une étude complète des aspects de tolérance de pannes échappe au cadre de cette thèse.

4.6 Résumé

Nous avons étudié dans ce chapitre le problème de gestion de la concurrence pour les collecticiels synchrones. Ce type de collecticiels est souvent implanté selon une architecture totalement dupliquée afin de favoriser le temps de réponse et la résistance aux pannes. Dans une première partie nous avons défini les problèmes posés par la concurrence d'accès à l'espace partagé défini par un collecticiel synchrone et nous avons identifié les principales propriétés requises d'un protocole de gestion de la concurrence pour ce type d'applications. Un modèle général de description des protocoles de gestion de la concurrence est ensuite proposé. D'après ce modèle un protocole est divisé en trois couches dites aussi modèles : le modèle de la concurrence, le modèle de la cohérence et la modèle de la communication. À partir de ces trois couches nous avons présenté les différentes approches de construction de protocoles de gestion de la concurrence. Deux mécanismes de base sont identifiés pour la construction d'un protocole de gestion de la concurrence : le mécanisme de verrouillage et le mécanisme de détection de dépendances. Les deux mécanismes peuvent être exploités de différentes façons pour implanter une variété des protocoles. Une comparaison entre les différents variantes est faite en fonction de la satisfaction des propriétés requises d'un tel protocole. La conclusion de cette étude est qu'il n'existe pas un seul protocole qui soit compatible avec les besoins de tous les types de sessions de travail coopératif synchrone. Une plate-forme générique doit supporter une variété des protocoles et laisser le choix du protocole à appliquer aux utilisateurs.

Dans une deuxième partie de ce chapitre nous avons proposé un nouvel algorithme de gestion de la concurrence, nommé *LICRA*. L'algorithme *LICRA* est un algorithme optimiste fondé sur la détection de dépendances et la résolution automatique des conflits en utilisant un mécanisme de transformation d'opérations. Le mécanisme présente l'originalité d'employer un mécanisme de détection des dépendances directes entre les opérations au lieu de l'emploi des horloges logiques comme c'est le cas dans d'autres algorithmes optimistes.

Chapitre 5

Conclusion

5.1 Les principaux apports

Les travaux présentés dans ce mémoire ont permis d'une part d'élucider la problématique de la construction de collecticiels et d'autre part de définir une nouvelle architecture logicielle pour supporter le travail coopératif. Nous résumons nos principales contributions à la recherche en *TCAO* dans les deux sections suivantes.

1. La problématique des collecticiels. Tout collecticiel se décompose en trois axes : la communication, la coordination et la production. Cependant nous distinguons selon la nature de l'unité du contrôle trois classes disjointes de collecticiels :

Les collecticiels à immersion où le contrôle s'exerce sur les utilisateurs eux-mêmes. Ce type de collecticiels implante une extension virtuelle de l'espace physique (ex. monde virtuel).

Les collecticiels procéduraux où le contrôle s'exerce sur le routage de l'information entre les utilisateurs (ex. applications de flux de travail).

Les collecticiels contractuels où le contrôle s'exerce sur les primitives de partage et d'accès aux données sujet de la coopération (ex. éditeur coopératif).

Nous nous somme intéressé à la construction des collecticiels de troisième type. Une approche généralement empruntée pour construire ces applications consiste à définir une plate-forme qui fournit l'ensemble des services requis pour la coopération (communication, production et coordination). Nous avons montré que pour qu'une plate-forme soit générique elle doit fournir les deux services additionnels suivants :

- Permettre l'orchestration et l'emploi d'un groupe de collecticiels par le même groupe d'individus.

- Intégrer le travail individuel et le travail coopératif.

Nous avons utilisé le terme *environnement de coopération* (EdC) pour désigner une plate-forme qui fournit ces deux services. Pour identifier les choix à faire pour construire un EdC nous avons emprunté la démarche suivante :

1. Identifier les services qu'une plate-forme exemplaire doit fournir pour résoudre deux problèmes fondamentaux dans le travail coopératif à savoir : le problème de la participation d'un utilisateur à une activité coopérative et le problème de la coordination entre les participants à une activité.

2. Comparer les différentes approches proposées dans la littérature de *TCAO* pour la construction d'une plate-forme coopérative en fonction de leur satisfaction des services identifiés dans l'étape précédente.

À l'issue de la première étape, nous avons identifié les services suivants :

Services requis pour la participation

- Implanter les deux stratégies de gestion de rendez-vous : explicite et implicite.
- Permettre le passage souple entre les deux modes de travail individuel et en groupe.
- Implanter les deux stratégies d'enregistrement : libre et supervisé (implicite ou explicite.
- Permettre la participation dynamique de nouveaux arrivants ainsi que les départs anticipés.

Services requis pour la coordination

- Fournir un service de définition et de gestion de rôles fonctionnels.
- Supporter les deux types d'interaction : synchrone et asynchrone.
- Implanter un service de présentation de la co-présence. Ce service doit couvrir les quatre aspects de la description de la co-présence (Qui? Où? Quoi? et Comment?)
- Implanter des stratégies variées de gestion de la concurrence.

À l'ensemble de services cités ci-dessus s'ajoutent les deux fonctions générales suivantes :

- Permettre le changement dynamique de la configuration des activités.

- Permettre l'évolution future des fonctions offertes.

La classification proposée des approches de construction de plates-formes pour le *TCAO* repose sur les trois critères suivants :

1. Le niveau du support qui peut être au niveau de l'interface homme machine, au niveau de données ou au niveau de la communication.

2. L'architecture de mise en œuvre qui peut être centralisée, dupliquée ou hybride.

3. Les applications supportées qui peuvent être des nouvelles applications ou des applications existantes. Dans le dernier cas la transformation d'une application en collecticiel peut se faire par intégration (sans modification de code) ou par adaptation (avec modification de code).

La comparaison fonctionnelle des approches identifiées ci-dessus nous a révélé les choix suivants à suivre pour construire un EdC :

1. Définir le support de la coopération au niveau de données.

2. Construire les collecticiels à employer dans l'EdC selon une architecture complètement dupliquée.

3. Restreindre le réutilisation d'applications existantes à l'intégration selon l'approche haut niveau (niveau de l'application) ou par voie d'adaptation.

2. La solution : Colt Nous avons proposé une nouvelle architecture logicielle pour supporter le travail coopératif appelée l'environnement de coopération *Colt*. Les choix de construction de *Colt* s'appuient sur les résultats de notre étude de comparaison des approches de construction existantes dont le résultat est donné ci-dessus. Parmi les différents services requis d'un EdC nous avons mis l'accent dans l'étude de conception de *Colt* sur les services suivants :

1. Fournir un modèle générique de rendez-vous.

2. Permettre le lancement et l'exécution de scénarios coopératifs variés.

3. Permettre le passage facile entre le travail individuel et le travail en groupe.

La conception de *Colt* présente plusieurs innovations en particulier la combinaison des deux métaphores de conception : les locaux partagés et les pièces pour modéliser les activités coopératives. Un autre point intéressant dans la conception de *Colt* est la définition d'un modèle souple pour la spécification et l'évaluation des droits d'accès et de règles de protection de l'espace partagé. Une nouveauté présentée par ce modèle est l'introduction de la notion du rôle utilisateur comme un rôle à part entière auquel nous pouvons associer des privilèges exceptionnels

(positifs ou négatifs). Une attention particulière est faite en vue de doter l'environnement de stratégies variées pour contrôler les accès concurrents des utilisateurs aux données partagées. L'environnement *Colt* propose une famille de protocoles de tour de rôles et intègre le protocole *LICRA* (chapitre 4) que nous avons conçu et proposé dans le cadre de cette thèse.

Une étude de comparaison avec d'autres plates-formes et environnements de coopération nous a montré la richesse fonctionnelle de notre proposition. La faisabilité du concept est démontrée par le développement d'un prototype réduit sur un ensemble de stations Unix et en utilisant l'environnement de programmation TCL-TK.

5.2 Les perspectives

Les perspectives de ce travail sont nombreuses. Nous présentons dans la suite deux types de perspectives : des perspectives concernant l'environnement *Colt* et d'autres concernant la recherche dans le domaine du *TCAO*.

Des perspectives pour l'environnement Colt Nous souhaitons continuer le développement de l'environnement *Colt* selon les axes suivants :

1. Amélioration du mécanisme de spécification des activités Un premier axe consiste à fournir un langage adapté à pour la spécification des modèles d'activités coopératives. Nous souhaitons affiner les modèles d'activités pour pouvoir prendre en compte des formes plus variées d'interaction entre les utilisateurs. En particulier nous souhaitons mettre en œuvre des services de *fragmentation* des documents de coopération qui rendent possible d'attribuer à des utilisateurs différents des privilèges différents sur des zones différentes d'un même document.

2. Intégration des collecticiels procéduraux et des collecticiels à immersion Une approche envisageable pour l'intégration de collecticiels procéduraux consiste à doter l'environnement d'un mécanisme d'expression et d'évaluation de règles d'induction qui agissent sur les critères de démarrage, de validation et de terminaison des activités coopératives. Par exemple pour spécifier qu'un document doit être édité par un groupe et corrigé par un autre, il nous suffit de créer deux activités : la première correspond à la phase d'édition et la deuxième à la phase de correction. Le démarrage de la deuxième activité est subordonnée à la terminaison de la première activité.

Des perspectives pour la recherche en TCAO Un premier objectif consiste à compléter l'étude de comparaison des approches de construction des plates-formes pour les collecticiels en vue de proposer un véritable système d'aide à la

conception des collecticiels. La fonction d'un tel système est d'indiquer l'approche la plus appropriée pour la construction d'un collecticiel en s'appuyant sur les connaissance suivantes :

- Les limitations fonctionnelles des approches de construction de collecticiels. Avoir ce type de connaissance nous demande d'étendre l'ensemble des critères retenus pour la comparaison fonctionnelle cités dans (chapitr 2) pour qu'il englobe l'ensemble des services requis pour la coopération (voir annexe B).

- Le coût de fonctionnement d'un collecticiel. Il dépend de la configuration matérielle disponibles (stations et réseau d'interconnexion) et du schéma de mise en œuvre du collecticiel.

- Le coût de développement du système. Il dépend de la possibilité d'intégration ou d'adaptation des applications existantes et de la possibilité de réutilisation de codes et de l'emploi des services pré-définis pour la coopération.

Un autre objectif sur lequel l'auteur de cette thèse souhaite continuer son travail est l'étude des méthodes de construction des interfaces homme-machine pour le travail coopératif. Un problème central dans la construction d'un collecticiel (ou d'un EdC) est celui de la présentation de la co-présence des utilisateurs. Peu de travaux dans la littérature ont abordé le délicat problème de définition des nouvelles métaphores et de modèles d'interaction avec la machine qui prennent en compte les contraintes du travail en groupe.

Annexe A

Pluridisciplinarité du TCAO : Apport des sciences humaines

Ayant pour *clientèle* des groupes humains, le concepteur de collecticiels est contraint à *comprendre* la nature et la dynamique du travail en groupe et à pouvoir analyser et mesurer l'impact des collecticiels sur le comportement des groupes cibles. Par conséquent la contribution des sciences humaines, qui ont pour objet d'étude l'homme et la société, est primordiale. La pluridisciplinarité complique davantage le développement des collecticiels. Elle demande d'établir un dialogue et une synergie qui réunie différentes domaines de compétences cpmme l'informatique, la sociologie et la psychologie.

Établir un dialogue *positif* au sein d'une telle synergie demande à tous les partenaires de montrer une soupleless et une ouverture à l'égard des autres disciplines. Un dialogue entre des individus qui ont des cultures différentes est une tâche réputée par sa difficulté. Un premier obstacle est la construction d'un référentiel commun où chacun des partenaires peut interpréter correctement les *actes* et les contributions des autres. La construction d'un tel référentiel est d'autant plus difficile que la divergence initiale entre les référentiels propres à chaque partenaire est grande.

L'interférence des sciences sociales ou sciences humaines avec des sciences informatiques n'est pas une exclusivité du domaine de *TCAO*. La *psychologie cognitive* est déjà au centre des études de conception et d'évaluation des interfaces homme-machine. La science de la *gestion* et la *sociologie* sont d'importance extrême pour la conception des systèmes d'automatisation de bureau (un domaine considéré par certain comme le prédécesseur directe du *TCAO*).

La particularité des collecticiels est qu'ils font appelle à une grande nombre des disciplines variées à la fois. De nombreuse études ont montré que la faible notoriété des collecticiels est due à des raisons sociales plutôt qu'à des raisons technologiques. Dans [Grundin88] l'auteur attribue *l'échec* de l'adoption des col-

lecticiels dans les organisations actuelles aux facteurs suivants :

1. La disparité de la distribution du *profit* apporté par l'utilisation d'un collecticiel.

2. La tendance presque monotone des *directeurs* de favoriser l'utilisation d'applications qui servent leurs propres intérêts. La difficulté de satisfaire les contraintes précédents dépende essentiellement du modèle de communication employé. Par exemple si le modèle de communication garantit une communication fiable et en ordre total les trois conditions sans se soucier de leurs effets sur les autres membres de l'organisation.

3. La grande difficulté d'évaluer les avantages et les inconvénients de l'introduction d'un collecticiel aux organisations.

Dans [Markus et al.90] les auteurs renforcent l'analyse de l'échec cité ci- dessus. Ils montrent que le profit apporté à un utilisateur est une fonction du ratio de nombre d'utilisateurs qui adopte le collecticiel par rapport au nombre de ceux qui ne l'adopte pas. Ils montrent qu'une *masse critique* d'utilisateurs du système doit être atteinte afin que le collecticiel soit profitable. Pour illustrer cette notion de masse critique prenons l'exemple d'une simple application d'agenda coopératif. La figure A.1 montre les variations du profit d'un utilisateur (la courbe **U**) et de celui d'un non-utilisateur (la courbe **N**) en fonction du nombre total des utilisateurs de l'application. Les deux courbes sont des droits parallèles où la différence $P_2 - P_1$ représente le coût que paie un utilisateur pour utiliser le système (par exemple le temps qu'il passe à maintenir son entrée dans l'agenda). Le profit collectif de l'ensemble de l'organisation est représenté en courbe pointillé. D'après cette figure le nombre total d'utilisateurs doit être supérieur à N_c, la masse critique par définition, pour que l'utilisation du collecticiel soit profitable.

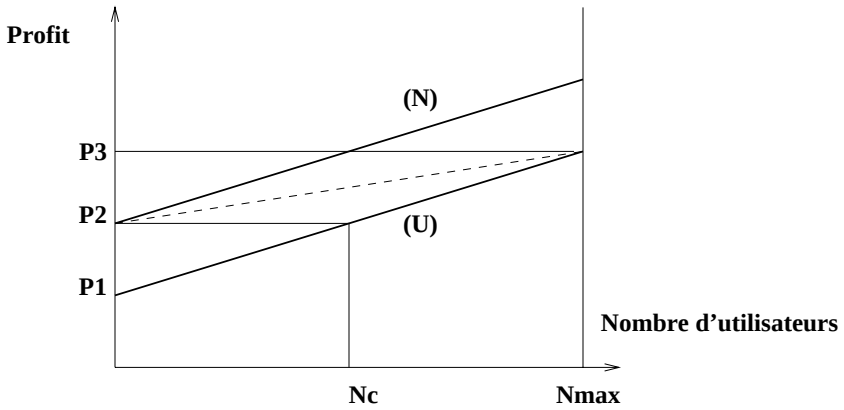


FIG. A.1 - : Le concept de la masse critique : le nombre des utilisateurs du collecticiel doit être supérieur à N_c pour que son rendement soit positif.

Pour contourner le problème de la création d'une masse critique une organisation peut forcer l'utilisation du collecticiel à tous ses membres. Cette solution est loin d'être la bonne puisque toute sorte l'obligation crée une *résistance* qui ne serait pas toujours au profit du collectif.

Les études citées ci-dessus et d'autres montrent que la conception et l'introduction d'un collecticiel au sein d'une organisation nécessite une analyse sociale et organisationnelle du contexte du travail. Dans le même temps l'emploi d'un collecticiel influencerait sans doute la *culture* de l'organisation cible. Une relation dialectique existe entre la technologie du *TCAO* et la sociologie des organisations cibles. Nous exposons les différentes facettes de cette relation dans les paragraphes qui suivent.

A.1 Le rôle de l'ethnographie

La négligence des données sociales est désignée d'être le facteur principale de l'échec constaté des collecticiels [Markus et al.90], [Shapiro94]. Afin de combler la lacune, les informaticiens sont amenés à coopérer avec des sociologues et des ethnographes pour identifier les besoins de travail en groupe.

L'ethnographie, selon *Larous*, est la branche des sciences humaines qui a pour objet l'étude descriptive des ethnies. Un ethnie est un groupement de familles dans une aire géographiquement variable dont l'unité repose sur une structure familiale, économique ou sociale commune et sur une culture commune. L'ethnographie est en conséquence la science qui s'occupe d'étudier le nature de travail et de relation au sein des entreprises et des organisations sociales.

Une expérience de construction d'un collecticiel de contrôle aérien fondée sur une étude ethnographique est décrite dans [Hughes et al.92], [Bentely et al.92a]. Dans cette étude les auteurs concluent que malgré la difficulté de la coopération entre des ethnographes et des informaticiens (une difficulté que nous explicitons dans la section A.5) l'étude ethnographique du contexte de l'utilisation du systèmes a largement influencé les choix de conception du collecticiel. Les résultats de cette étude ont révélé que certaines *règles* recommandées généralement pour la conception des systèmes informatiques ne sont plus valables dans le domaine du collecticiel. À titre d'exemple, nous citons la règle suivante *contredite* par l'étude ethnographique :

> *Le système informatique doit automatiser toutes les tâches manuelles qui consiste à comparer des données semblables et à trier des listes de données.*

Les observations ethnographiques montrent que certaines interventions manuelles doivent le rester. D'autre côté, l'ethnographe après avoir étudié le domaine de l'utilisation d'un système, devient la personne la plus apte à remplacer l'utilisateur final pour l'évaluation du système avant son delivrance. Ainsi, il peut

contribuer à découvrir des erreurs et des défauts de conception et d'implantation dans une phase préliminaire de la construction du produit. Pour résumer, l'ethnographie contribue à la fois à guider l'étude de conception du collecticiel et à participer à l'évaluation du système pendent et après son développement.

A.2 Le rôle de la psychologie

L'autre facette de la dialectique socio-technique du *TCAO* se manifeste dans les effets psycho-sociales de l'emploi des collecticiels. L'étude de l'impact des collecticiels sur le comportement des utilisateurs entre par excellence dans le domaine de la *psychologie sociale*.

Les effets psychologiques de l'utilisation de collecticiels ont été d'abord observés dans le cas des collecticiels de communication homme-homme médiatisée (*CHHM*) tels le courrier électronique (*e-mail*) et le *News*. Les études de l'utilisation des outils de *CHHM* ont révélé que les utilisateurs montrent une *agressivité* nettement supérieur par rapport aux autres méthodes de communication (par exemple dans le cas des réunions conventionnelles). Les insultes et l'utilisation des mots et des expressions inappropriés ne sont pas rares [Dyre et al.95]. L'agressivité verbale peut favoriser la violence et affaiblir la cohésion interne de l'organisation. L'analyse faite dans [Dyre et al.95] suggère que l'emploi d'application de *CHHM* conduit les utilisateurs à un état psychologique d'*individuation* où chacun est moins conscient des ses propres actes et des effets de ses actes sur les autres.

Dans [Sproull et al.91] les auteurs comparent le déroulement et la *qualité* des décisions prises par des groupes médiatisés (qui se servent des outils de *CHHM*) par rapport à des groupes conventionnels (où les réunions sont faites face à face). À l'issu de cette étude les auteurs identifient les caractéristiques suivantes des groupes médiatisés :

c1. La distribution de périodes de prise de parole est plus équitable.

c2. La participation des individus placés vers l'haut du pyramide organisationnel du groupe est beaucoup plus réduit. Ceci a le conséquence directe de réduire le poids d'avis de ces personnes contrairement aux réunions conventionnelles où ces même gens dominent le groupe.

c3. Obtenir un consensus est plus difficile pour les groupes médiatisés. L'emploi d'une langage violente est l'un des facteurs sources de cette difficulté.

c4. En conséquence directe de c1 et de c2, la qualité des décisions prises par les groupes médiatisés est meilleur puisque l'expertise est devenu prioritaire à la position sociale.

c5. Les décisions prises par les groupes médiatisés comprennent plus des risques que celles prises par les groupes conventionnelles.

Les résultats de cette étude montrent que l'emploi de collecticiels n'est pas toujours recommandé. La décision d'entreprendre une réunion médiatisée ou une réunion conventionnelle dépende du contexte du travail et des enjeux à discuter.

A.3 La psychologie cognitive

La psychologie cognitive vise à comprendre et à modéliser les mécanismes cognitifs mis en jeu dans les activités humaines (planification, exécution d'un plan, apprentissage, mémorisation et perception de l'environnement). La contribution de la psychologie cognitive est une extension directe de sa contribution dans le domaine de la conception et de l'évaluation des interfaces homme- machine. Un aspect important de l'application de la psychologie cognitive dans le domaine du travail coopératif consiste à évaluer la capacité de l'interface d'un collecticiel à favoriser la communication et la coopération. Des systèmes d'évaluation des interfaces de collecticiels fondés sur des théories cognitives sont déjà disponibles. Nous citons pour l'exemple le système *MESC* (méthode d'évaluation du support à la coopération) [ZV et al.95a], et le système *ICS* (Interacting Cognitive Subsystemes) [Salber95]. La méthode *MESC* est une méthode d'évaluation prédictive. Elle utilise les modèles de spécifications conceptuelles et de l'interaction homme-machine pour évaluer plusieurs alternatives de conception [ZV et al.95b]. Le système *ICS* présente l'intérêt de prendre en considération les aspects cognitifs de l'interaction humaine avec des systèmes informatiques multimédias.

A.4 Sciences de la communication

Il y a communication chaque fois un ensemble d'individus se met à travailler en commun. La communication fait depuis long temps l'objet des études scientifiques. Les sciences de la communication se sont intéressé à toutes les facettes d'un processus de communication : la transmission de données, les langages verbales et non verbales (gestuelle), l'organisation des individus communicants et même la formation à la communication [Amado et al.93]. Il est évident que les résultats apportés par les sciences de communication sont d'extrême importance pour l'étude de conception des collecticiels. Une présentation complète des différentes théories de communication et de leurs influence dans le domaine du *TCAO* est faite dans [Mccarthy et al.94]. D'autres branches des sciences sociales, comme l'éthique informatique et le droit sont aussi demandé à contribuer au domaine du collecticiel. L'étude de contributions de ces sciences est encore dans des phase primaires. Nous renvoyons le lecteur au [Salber95] pour une essaie sur la contri-

bution attendue de ces deux branches de sciences sociales dans le domaine de *TCAO*.

A.5 Pluridisciplinarité : mise en œuvre

Le large éventail des domaines et de disciplines liés à l'étude et à la réalisation des collecticiels montrent qu'une équipe de recherche qui s'occupe du travail coopératif assisté par ordinateur doit être une équipe pluridisciplinaire. La projection des contributions de différentes disciplines sur les trois principales phases de développement d'un logiciel, à savoir : analyse de besoins, conception et prototypage, et évaluation est donnée à la figure (A.2). En pratique, la mise en œuvre

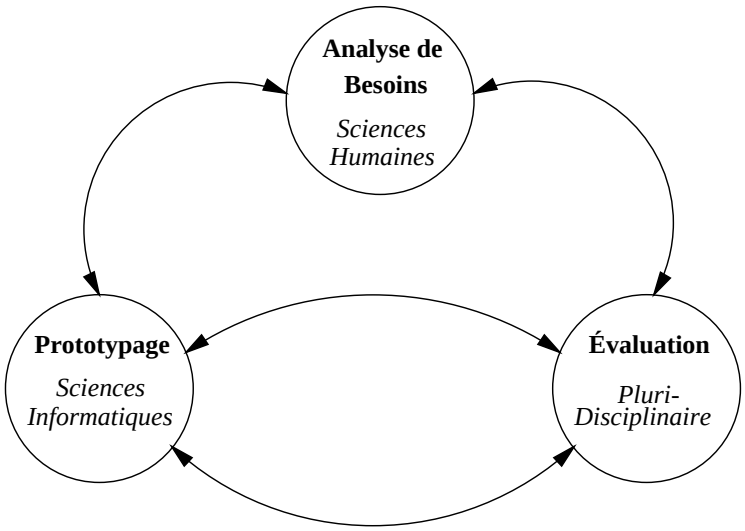


FIG. A.2 - : Pluridisciplinarité du développement du collecticiel

d'une telle synergie rencontre de nombreuse difficultés. Parmi ces difficultés nous citons les suivantes :

1. *La taille du groupe :* les présentes méthodes d'études sociologiques et ethnographiques s'appliquent à des populations quasi- fermées et à effectif réduit. Il est difficile d'effectuer des études sociologiques poussées sur des populations complexes (organisations).

2. *Le temps d'étude :* les études sociologiques exigent des durées d'étude typiquement longues. Le temps requis pour faire une étude ethnographique dépassent facilement toutes contraintes raisonnables pour la réalisation des systèmes informatiques.

3. *La nature analytique des sciences humaines :* les sciences sociales sont typiquement des sciences analytiques. Elles décrivent les problèmes sans proposer des solutions. Les informaticiens auront la tâche difficile d'interpréter

les résultats des études sociologiques et de dégager des solutions techniques aux problèmes identifiés.

Afin d'éviter les problèmes cités ci-dessus, de nouveaux schémas de coopération entre des sociologues et des informaticiens sont proposés dans [Hughes et al.94] :

1. *L'ethnographie concurrente* : où la conception d'un collecticiel est faite en parallèle avec une étude ethnographique de son domaine d'utilisation.

2. *L'ethnographie rapide et sale* : où une brève étude ethnographique précède la conception du système

3. *L'ethnographie évaluative* : où une étude ethnographique est faite pour valider des choix de conception déjà faits.

La figure A.3 illustre les trois schémas proposés ci-dessus.

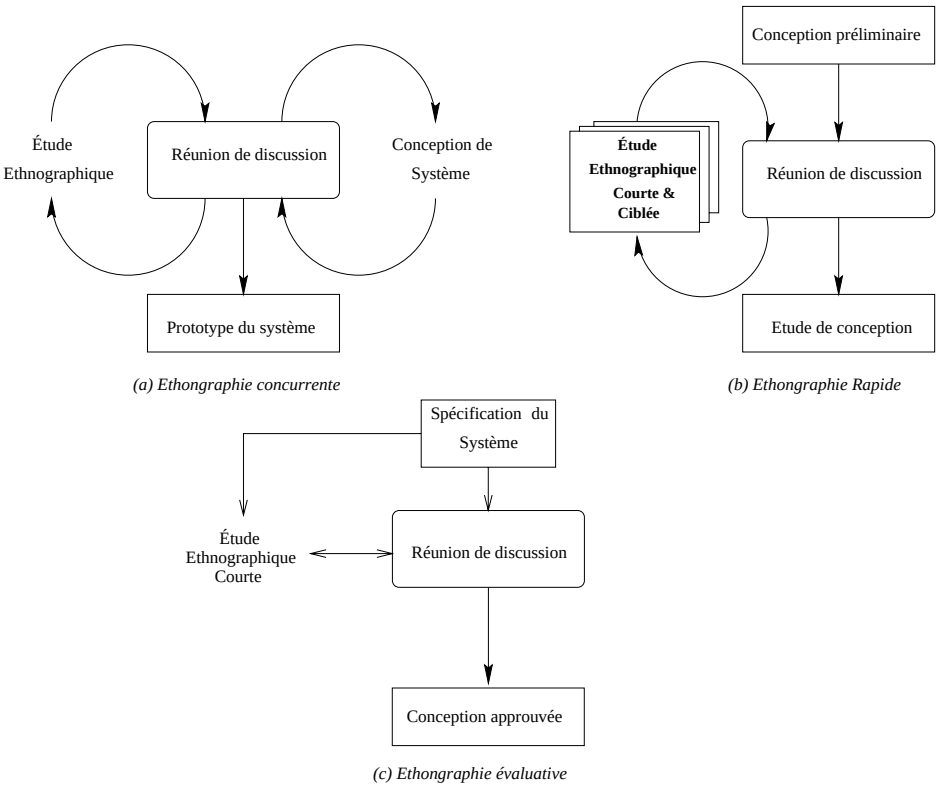


FIG. A.3 - : Alternatives de coopération socio-technique pour la conception de collecticiels [Hughes et al.94]

Annexe B

Services pour la coopération

L'objectif de cette section est d'identifier les services requis d'un support système informatique pour la construction et l'exécution des collecticiels *contractuels*. Plusieurs approches sont empruntées dans la littérature du *TCAO* afin d'atteindre cet objectif. Une première approche, dite *formelle*, repose sur l'introduction d'un formalisme qui *modélise* la coopération. Plusieurs modèles formels pour le travail coopératif sont proposés. Des exemples sont le modèle fondé sur la logique modale [Diaz92] et le modèle d'objets coopératifs fondé sur l'intégration des techniques de réseaux de *Petri* et l'approche de modélisation par objets [Palanque et al.95]. C'est aussi le cas du modèle d'objets reflectifs proposé dans [Dourish95a]. Le domaine des applications coopératives étant vaste, il est difficile de trouver un modèle formel qui modélise les différents types de collecticiels. À notre avis, un formalisme sert avant tout à clarifier un aspect particulier du travail coopératif. Il sertà vérifier la correction des solutions proposées qui mettent en œuvre les aspects qu'il modélise.

Une deuxième approche, que nous qualifions d'approche *sectorielle*, consiste à identifier les besoins d'une classe restreinte de collecticiels. Des exemples sont les études centrées sur les besoins des collecticiels synchrones [Roseman et al.93, Edwards95, Balter et al.96a] ou les travaux centrés sur le problème de l'édition coopérative [Nastos92, Knister et al.93]. Par nature, les études sectorielles présentent une vue partielle de l'ensemble des besoins du travail coopératif. Elle peuvent néanmoins alimenter des études dont les perspectives sont plus larges.

Une autre approche, que nous qualifions de *pragmatique* repose sur l'analyse des scénarios variés de coopération pour en déduire les besoins communs. Deux travaux intéressants dans ce domaine sont l'étude menée par *Schmidt* et *Rodden* dans [Schmidt et al.93] et l'étude reportée dans [Almasi et al.94]. Notre étude reprend et complète les résultats de ces deux travaux.

Par souci de clarté, nous avons choisi de structurer notre démarche en apportant des éléments de réponse à la suite *logique* des questions suivantes : Pourquoi travaille t-on en groupe? Avec qui? Quand choisir de travailler ensemble? Comment? Et sous quelles contraintes? Nous développons nos réponses à chacune des ques-

tions précédentes dans la suite.

B.1 Pourquoi?

Pour résumer l'intérêt et les enjeux du *TCAO*, *Derycke* et *Hoogstoel* font dans [Derycke et al.95] l'analogie du travail coopératif avec les systèmes multi-processeurs :

> *"... le passage d'un individu/processeur à N individus/processeurs n'entraîne pas un gain d'un facteur N. Pour certains processus, il peut même y avoir détérioration de la performance (le cas de la vitesse dans la prise de décision), dans d'autres cas, le gain est infini : un individu seul ne pouvant accomplir la tâche".*

L'analogie *TCAO* / systèmes multi-processeurs montre que deux motivations principales peuvent être à l'origine du lancement d'une activité coopérative :

1. La nature de la tâche à faire nécessite l'intervention de plusieurs individus. C'est le cas d'une activité d'enseignement où une *coopération* enseignant-élève est requis.

2. L'intervention de plusieurs individus *peut* améliorer la qualité du travail ou réduire le temps de l'exécution de la tâche. C'est le cas d'un processus de prise de décision (améliorer la qualité) ou de la rédaction commune d'un document (réduire le temps).

La décision d'effectuer en groupe une tâche qui peut être faite individuellement est généralement fondée sur une conviction que le profit apporté par le groupe est plus important que le coût de la coordination entre les membres du groupe.

Parfois le besoin de coopérer avec d'autres peut être ressentie pendent l'exécution individuelle d'une tâche ; souvent la progression d'une tâche nécessite d'alterner des phases de travail individuel et des phases de travail en groupe. Pour illustrer ceci prenons l'exemple d' un utilisateur qui rédige une lettre ou un courrier électronique. Il estime qu'il a besoin de l'avis d'une autre personne pour corriger et réviser sa lettre. Demander l'aide d'un autre utilisateur ne doit obliger l'utilisateur à changer l'application de traitement de texte qu'il utilise pour rédiger la lettre. Par suit, un premier besoin requis d'une plate-forme pour collecticiels est le suivant :

```
B1. Permettre le passage facile entre les deux modes de travail :
individuel et en groupe.
```

B.2 Qui?

La désignation des individus qui peuvent participer à une activité coopérative dépend de la nature et du but de l'activité en question. Nous distinguons deux sortes de désignation :

1. *Désignation directe* où les participants potentiels sont désignés chacun par son propre nom. C'est le cas par exemple d'une réunion entre amis.

2. *Désignation indirecte* où la participation d'un individu à une activité lui demande de remplir un critère ou d'avoir un rôle particulier. C'est le cas par exemple d'une réunion d'équipe de travail dont seuls les membres de l'équipe peuvent y faire part.

La mise en œuvre des politiques de désignation citées ci-dessus demande de fournir à chaque utilisateur (qui démarre une activité) des informations sur les autres utilisateurs potentiels du système. Le support système doit donc fournir une base d'informations qui décrit les membres de l'organisation cible du collecticiel. Cette base contient des informations sur les utilisateurs, leurs noms, leurs coordonnées, leurs rôles et leurs rangs dans l'organisation, leurs compétences et les tâches qui leurs sont confiées. Un utilisateur doit pouvoir interroger cette base pour trouver les personnes les plus aptes à coopérer avec lui sur sa tâche courante. Par conséquent, un support système pour le travail coopératif doit remplir le besoin suivant :

`B2. Fournir un répertoire organisationnel.`

Être désigné pour participer à une activité est une condition nécessaire mais non suffisante pour participer effectivement à l'activité en question. D'autres conditions peuvent être exigées. Par exemple, le nombre de participants peut être limité pour certaines tâches. C'est le cas dans la plupart des jeux multi-utilisateurs. D'autres tâches exigent que la participation d'un individu soit parriné par un ou plusieurs participants actuels. Il n'existe pas une seule politique de contrôle d'appartenance qui soit compatible avec tout sorte de collecticiels. De là, une plate- forme pour le travail coopératif doit satisfaire le besoin suivant :

`B3. Fournir une variété de politiques de gestion d'appartenance aux groupes.`

Dans certains cas, surtout lors de l'initiation des activités coopératives temps réel, il est souhaitable, voire nécessaire, de coupler le service de désignation par un service qui montre l'état actuel de l'*occupation* des utilisateurs désignés (comme c'est le cas lors d'un appel téléphonique). Il en découle qu'un support système pour le *TCAO* doit fournir le besoin suivant :

`B4. Rendre perceptible à chaque utilisateur l'existence et la disponibilité des autres utilisateurs dans l'espace partagé.`

Indépendamment du mode de désignation des utilisateurs, il se peut qu'un utilisateur *qualifié* pour participer à une tâche ne soit pas disponible lors du lancement de l'activité. De même, il se peut qu'un participant quitte une tâche avant sa terminaison. L'arrivée tardive ou le départ anticipé d'un utilisateur ne doivent pas perturber le déroulement d'une activité. Par exemple, dans une séance de télé-enseignement un étudiant qui arrive en retard doit avoir la possibilité de rejoindre la session. De même, le départ d'un étudiant ne doit pas entraîner l'interruption de la session. D'autre part, les besoins d'une activité coopérative peuvent changer en cours du déroulement de l'activité. Par exemple, dans une activité de prise de décision, l'examen du problème à résoudre peut montrer que la participation de nouveaux experts est recommandée, ou l'inverse. Par conséquent, un support système doit remplir le besoin suivant :

`B5. Permettre la composition dynamique du groupe des coopérants.`

B.3 Quand ?

Selon leurs modes de lancement les activités coopératives peuvent être classées en deux classes :

1. *Les activités planifiées* pour lesquelles le moment du démarrage est connu à l'avance. La date de démarrage peut être *absolue* ou *relative*. Un exemple d'une date absolue est l'inscription d'une réunion sur un agenda. Un exemple de date relative est de lier le démarrage à l'occurence d'un ou plusieurs événements, par exemple la terminaison d'une autre activité.

2. *Les activités opportunistes* où le lancement de l'activité est fait d'une manière fortuite. C'est le cas du rencontre de deux ou plusieurs utilisateurs sur un même document (ex. accéder en même temps à la même page *Web*).

Afin de supporter l'exécution des activités du premier type le système doit fournir aux utilisateurs la possibilité de connaître les activités planifiées et les activités en cours. Les caractéristiques de chaque activité annoncée doivent être précisées (par exemple les conditions et le mode de participation). De plus, les utilisateurs doivent avoir la possibilité d'annoncer la création d'une nouvelle activité (par publication ou par envoi des invitations aux autres utilisateurs). Ceci nous laisse déduire le besoin suivant :

`B6. Fournir un répertoire des activités coopératives.`

Favoriser le lancement fortuit des activités renforce le besoin `B4` qui incite à fournir une service de *conscience de groupe* qui rend chaque utilisateur *conscient* de l'existence et des positions des autres dans l'espace partagé.

B.4 Comment ?

Généralement, la réalisation d'un œuvre commun par un groupe demande aux acteurs :

- de communiquer entre eux pour discuter, argumenter et commenter leurs contributions,
- de partager les objets constituant l'œuvre et de pouvoir les échanger entre eux.

Par exemple lors de la rédaction d'un document les coauteurs doivent avoir la possibilité de communiquer entre eux pour discuter l'élaboration du document, de corriger et d'annoter leurs contributions et en même temps ils doivent avoir accès au même document ou pouvoir échanger entre eux les fragments du document qu'ils rédigent. Par conséquent un support pour la coopération doit fournir le besoin suivant :

> `B7. Fournir des mécanismes de communications directes entre les utilisateurs et des mécanismes de partage et/ou échange d'informations.`

La mise en œuvre des mécanismes de communication et de partage d'informations nécessite de fournir des politiques de *coordination* entre les différents utilisateurs (par exemple il faut éviter que tous les utilisateurs éditent en même temps la même section d'un document partagé ou que tous les utilisateurs parlent en même temps sur un canal audio). Une variété de politiques de coordination peuvent être mises en place. Dans [Greenberg et al.94] les auteurs montrent qu'il n'existe pas une politique de gestion de la concurrence qui peut être appliquée dans tout sorte de collecticiel. Le choix de la politique à appliquer dépend du contexte et de la nature de l'activité coopérative. De là nous déduisons le besoin suivant :

> `B8. Fournir une variété de politiques de coordination pour la gestion de la communication et du partage d'informations.`

B.5 Contraintes

L'introduction des collecticiels aux organisations humaines rencontre beaucoup de difficultés. Plusieurs travaux attribuent l'échec des collecticiels à la *négligence* des aspects sociaux des organisations cibles et à l'insuffisance des études des effets sociaux et psychologiques de l'emploi des collecticiels [Grundin88, Grundin94, Dyre et al.95]. La prise en compte des aspects sociaux et psychologiques demande d'établir une synergie qui réunit des informaticiens, des psychologues et des ethnologues. Nous développons l'aspect pluridisciplinaire du *TCAO* et les problèmes

qui en résultent dans l'annexe A. Dans ce même annexe nous présentons une étude décrite initialement dans [Markus et al.90] dont la conclusion est la suivante :

> *Pour qu'un collecticiel soit utilisable il faut qu'il satisfait une masse critique de ses utilisateurs potentiels.*

L'ordre de grandeur de la masse critique est bien sûre une fonction du nombre total des utilisateurs potentiels, mais il est aussi fonction du rapport $\frac{Satisfaction}{Effort}$ pour chacun des utilisateurs. Il est clair que pour satisfaire le plus d'utilisateurs il ne faut pas que la technologie de la coopération compromette les avantages acquises de la technologie *traditionnelle* de l'information (surtout par rapport à l'emploi des ordinateurs personnels en conjugaison avec des simples outils de communication comme le courrier électronique ou les systèmes de transfert de fichiers).

Un premier aspect important dans ce contexte consiste à garantir la disponibilité des informations ; une disponibilité assez élevée lors de l'utilisation des applications mono-utilisateurs sur des stations de travail personnelles. Par conséquent un support système pour la coopération doit répondre au besoin suivant :

> `B9. Garantir la disponibilité des informations partagées.`

Le besoin précédent implique la nécessité de fournir des mécanismes de tolérance aux pannes et aux défaillances de sites et de réseaux d'interconnexion.

D'autre part, les utilisateurs n'ont pas forcément les mêmes environnements informatiques du travail (matériels et logiciels). Les différents utilisateurs possèdent des stations de travail différentes (Sun, PC ou Macintosh), des systèmes d'exploitation différents (Unix, Dos, Windows ou MacOs) et des applications différentes pour mener la même tâche (par exemple deux utilisateurs peuvent employer deux éditeurs de texte différents pour rédiger le même document). Il est souhaitable donc qu'un support système pour le *TCAO* vérifie le besoin suivant :

> `B10. Permettre l'hétérogènité des environnements de travail en termes de plates-formes matérielles et des équipements logiciels (systèmes et applications).`

Le fonctionnement et l'utilisation d'un collecticiel dépendent fortement de son contexte d'utilisation. Par exemple ils dépendent du but de l'activité coopérative, du nombre de participants, des relations qui regroupent les participants et même de l'état d'avancement du travail. Pour illustrer la dépendance entre la configuration du collecticiel et son contexte d'emploi prenons l'exemple concret du choix de la politique de gestion de droit de parole dans un collecticiel synchrone. Lorsque le nombre de participants est relativement petit (deux ou trois) une *politique sociale*, à l'instar du cas d'une communication téléphonique, peut

être efficacement employée. Mais lorsque le nombre de participants devient plus important (de l'ordre de dix personnes) il faut mieux remplacer la politique sociale par une autre politique (ex. passage du jeton) afin de garantir une meilleure compréhension du déroulement de l'activité [Greenberg et al.94]. De même, dans une réunion informelle nous pouvons appliquer une politique de prise de parole fondée sur le principe du premier demandeur premier servi, tandis que dans une réunion formelle on emploi souvent une politique d'animation où un utilisateur privilégié se charge de d'attribution de droit de parole. Le contexte de l'emploi d'un collecticiel pouvant changer dynamiquement, il est important qu'un support système pour le *TCAO* vérifie le besoin suivant :

`B11. Permettre la configuration dynamique des activités coopératives.`

Un dernier mais pas le moins important besoin concerne l'ouverture du support système. Le domaine des applications coopératives étant très vaste, il est *naïve* d'accepter qu'une réalisation d'un support système fournit tout les services dont tous les scénarios coopératifs imaginables ont besoin. D'où le besoin suivant :

`B12. Un support système pour le TCAO doit être ouvert à toute évolution future.`

B.6 Synthèse

Nous regroupons les besoins identifiés dans les sections précédentes en cinq principaux services qui sont les suivants :

La conscience de groupe donne à chaque utilisateur une description des coopérants potentiels (B2), leurs positions dans l'espace partagé et leurs états actuels d'occupation (B3). Le service de conscience de groupe sert aussi à prévenir chaque utilisateur des activités en cours d'exécution et des activités planifiées (B6) ainsi de lui montrer les zones de l'espace du travail auquel il a le droit d'accès (B7).

Le partage et l'échange d'informations constitue un service de base pour la réalisation en groupe des œuvres communs (B7). La mise en œuvre des mécanismes de partage et d'échange d'informations nécessite le développement de mécanismes de coordination entre les utilisateurs (B8) ainsi que des fonctions de tolérance aux pannes afin d'augmenter la disponibilité des informations partagées (B9).

La gestion de groupes couvre les aspects de la définition de groupes (B2), de la gestion de l'appartenance aux groupes (B3,B5) ainsi que les aspects de communication directe à l'intérieur d'un groupe (B7).

La reconfiguration dynamique du système porte sur le changement dynamique de la configuration des activités coopératives. Ceci couvre la possibilité de supporter la connexion et la déconnexion dynamiques des utilisateurs (B6), de changer les politiques de coordination appliquées au cours de la coopération (B3,B11) ou de changer le mode de travail (B1).

L'ouverture porte sur la possibilité d'enrichir les applications et les services offerts par de nouveaux services et de comportements non initialement prévus par le système (B12). Une autre facette de l'ouverture consiste à supporter l'hétrogénité (matérielle et logicielle) des systèmes d'informations employées par les différents utilisateurs (B10).

Annexe C

Les systèmes répartis comme support pour le TCAO

Les systèmes répartis (*SR*) visent à faciliter la communication et le partage de ressources (mémoire, prépheriques et puissance de calcul) disponibles sur des stations de travail connectées entre elles par un réseau informatique. Le but de la présente section est d'évaluer à quel point les actuels *SR* supportent la construction des collecticiels.

La quasi majorité des *SR* sont fondés sur le principe de la *transparence de la distribution* selon lequel la répartition de ressources est *cachée* aux applications et aux utilisateurs. Le but est de donner l'*illusion* aux utilisateurs qu'ils disposent d'une seule machine dont la capacité est égale à la somme des capacités des machines connectées sur le réseau. Nous analysons rapidement dans la suite la *compatibilité* du principe de la transparence de la distribution avec les besoins des collecticiels. L'étude est faite en analysant l'impact de ce principe sur la mise en œuvre de six principaux services fournis par les *SR* qui sont les suivants : 1) la communication, 2) la duplication et la migration, 3) la gestion de la concurrence, 4) la tolérance aux pannes, 5) la mémoire partagée répartie et 6) la protection des données.

C.1 Modèles de communication

Tout *SR* repose sur un modèle de communication selon lequel un objet peut provoquer l'exécution d'une opération sur un autre objet du système. La projection de la transparence de la distribution sur le modèle de communication se traduit par rendre identiques, du point de vue des applications, l'accès aux objets distants et locaux. Plusieurs modèles de communication sont proposés et utilisés par les divers systèmes. Nous examinons dans la suite les trois modèles les plus

employés.

1. Le modèle client-serveur selon lequel la structuration d'un *SR* met en jeu deux entités : un *client* qui demande l'exécution d'un service et un *serveur* qui réalise ce service. Ce modèle de communication est adopté par plusieurs systèmes standards tels que ODP, OMG et DCE. Souvent, la mise en œuvre du modèle *client-serveur* repose sur l'emploi des appels de procédures à distance (*RPC*[1]). Le principe est de permettre à un processus p, s'exécutant sur un site C (client) d'exécuter une procédure P sur un site S (serveur) avec un effet global identique à celui qui serait obtenu par l'exécution locale de P [Balter et al.91]. À l'image d'un appel de procédure classique, le processus qui fait un *RPC* se bloque en attente du résultat. On dit alors que le *RPC* implante un modèle de communication *synchrone*. Bien que le *RPC* puisse servir comme un mécanisme de base pour le partage et l'échange d'informations ainsi que pour implanter un service de conscience de groupe, il présente des propriétés qui limitent son utilité pour la construction des collecticiels dont les principales sont les suivantes :

   (a) La synchronisation entre le client et le serveur rend la coopération plus difficile en raison de suspensions des exécutions à chaque envoi de message. Par exemple dans le cas d'édition synchrone d'un document l'utilisateur doit patienter après chaque modification (ex. insertion d'un caractère) le temps qu'il faut pour notifier les autres utilisateurs de la modification du texte. Ceci peut rendre l'éditeur non-utilisable à cause de son long temps de réponse.

   (b) Les clients doivent connaître à l'avance les serveurs dont ils auront besoin pendent leurs exécution. Ceci est peu compatible avec les collecticiels qui sont des applications dont le comportement est dynamique et peu prévisible.

   (c) Le modèle client-serveur est un modèle de communication bi-points qui offre peu de services pour la communication de groupe.

2. Le modèle à diffusion selon lequel le destinataire d'un message est un groupe de processus. L'exemple phare de l'emploi de ce modèle est le système q*ISIS* [Birman et al.94]. L'emploi d'un système à diffusion pose le problème majeur de la gestion de l'ordonnancement de la réception des messages par le groupe destinataire. *ISIS* propose des mécanismes intéressants qui permettent une grande soupeless dans le choix de l'ordre de la réception des messages (Ordre FIFO, causal, total) [Reness et al.96]. Ce modèle de communication est particulièrement attractif pour les applications coopératives où la notion de groupe est une notion centrale.

[1] Remote Procedure Call.

3. Le bus de messages dont le principe repose sur le dépôt asynchrone et la circulation des messages sur un bus logiciel auquel sont connectés les processus communicants (dits clients). Des exemples des bus de messages sont *The Information Bus* [Oki et al.93], *Koala* [Beust93] et *Field* [Reiss90]. La réception de messages par les clients est subordonnée à une acte d'abonnement. L'abonnement correspond à la mise en place d'un *filtre* qui permet de capter des messages circulant sur le bus et dont le client *veut* recevoir. La connexion au bus peut se faire d'une manière dynamique. De plus les clients peuvent modifier dynamiquement leurs abonnements. Le concept de bus de messages présente l'avantage d'être un moyen de communication asynchrone et anonyme. Il permet aussi une diffusion asynchrone à de nombreux destinataires. Cependant il n'offre pas un support adéquat pour la gestion de groupes puisque la réception des messages est contrôlée par les récepteurs et non pas par l'émetteur. Ce dernier se contente de confier le message au bus sans avoir aucune information sur l'existence d'un destinataire ou sans pouvoir restreindre l'émission à un ensemble des destinataires potentiels.

D'autres modèles de communication, dont l'emploi est moins répandu que les modèles présentés ci-dessus sont : le modèle de communication par espace de tuple, le *RPC* asynchrone et le modèle communication événementielle. Certains de ces modèles sont d'ailleurs proposés pour répondre en partie aux besoins de coordination non transparente entre processus communicants et le besoin de coopération, comme c'est notamment le cas de la communication événementielle. Une étude plus détaillée d'évaluation des mécanismes de communication pour les applications coopératives est faite dans [Lenormand96].

C.2 La migration et la duplication transparentes

La répartition des objets d'un système réparti pose un problème de performance en raison des envois de messages à travers un réseau. Pour remédier à ce problème deux mécanismes sont proposés : la duplication et la migration des objets. L'objectif est de *raccourcir* les distances entre les objets communicants et en conséquence diminuer le coût de la communication. Les deux mécanismes favorisent donc le partage de données en réduisant le coût d'accès aux informations partagées. Cependant ces deux mécanismes ajoutent un surcoût de fonctionnement. Par conséquence, l'emploi de ces mécanismes est une question de compromis entre les coûts qu'ils ajoutent et les bénéfices qu'ils réalisent. Un problème majeur dans la mise en œuvre de la duplication transparente est la gestion de la cohérence des copies des données dupliquées. Le système se charge de garantir la cohérence des différentes copies. Certaines applications de *TCAO*, notamment les applica-

tions fondées sur le paradigme *WYSIWIS* [2] peuvent bénéficier d'un support de duplication transparente. Or dans le cas général, l'exécution d'un collecticiel repose sur un modèle de succession de divergences et de fusions des versions de données partagées [Dourish95b]. Remonter la divergence des données au niveau de l'interface-utilisateur est parfois une fonction nécessaire [Condon92]. Par suite la gestion de la duplication doit être faite dans le contexte de l'application et non pas au niveau du support système.

La migration transparente fournit un moyen intéressant pour supporter la dynamicité et la flexibilité requises pour les collecticiels. Par exemple il est tout à fait envisageable de faire recours à la migration pour informer un nouvel arrivant de l'état actuel de la session coopérative qu'il vient de rejoindre. Le même mécanisme peut servir à optimiser le temps de réponse des applications à travers un meilleur équilibrage de la charge du système. Peu de travaux dans la littérature du *TCAO* ont étudié l'utilité de la migration pour la construction des collecticiels. Ceci constitue un axe de recherche qui nous semble intéressant à explorer.

C.3 La gestion de la concurrence

Conformément au principe de la transparence de la distribution la plupart des protocoles de gestion de la concurrence sont fondés sur l'idée de donner à chaque utilisateur l'illusion qu'il est le seul à utiliser le système. Les activités de différents utilisateurs sont isolées les unes des autres. La mise en œuvre de l'isolation repose sur l'emploi des mécanismes de sérialisation globale des actions faites par les utilisateurs. L'emploi des transactions est l'une des approches les plus classiques pour réaliser ce type de transparence [Bernstein et al.87]. Or le principe de l'isolation est à l'opposé du principe de la coopération où chaque utilisateur doit avoir une vision des activités des autres utilisateurs (la fonction de conscience de groupe). De nouvelles *politiques* de gestion de la concurrence sont à fournir pour supporter le travail coopératif. Ces nouvelles politiques reposeront sur les mêmes mécanismes de base (verrous et ordonnancement de messages). Nous étudions ce problème en détail dans le chapitre 5.

C.4 La transparence des pannes

Le but est de cacher l'occurence d'une panne (du réseau ou d'un site) aux sites non défectueux. Or pour les collecticiels, le traitement des pannes ne doit pas être systématiquemt caché aux participants. La réaction du système à l'occurence d'une panne dépend de son contexte d'utilisation. Pour illustrer ce point, prenons l'exemple d'un collecticiel d'apprentissage coopératif. Une session d'enseignement comprend, dans sa forme la plus simple, un enseignant et plusieurs étudiants.

[2] What You See Is What I See.

L'arrêt ou l'isolation d'un site étudiant ne doit normalement pas entraîner l'arrêt total de la session (transparence de la panne d'un site étudiant). Par contre la déconnexion du site de l'enseignant met le déroulement de la session en cause. Normalement, la session doit être avortée. D'une manière générale, le déroulement d'une session coopérative est gardé par des conditions de travail. Ces conditions expriment la validité de l'exécution de la session [Paoli et al.94, Villemur95]. La violation des conditions de garde entraîne l'arrêt ou la suspension de la session.

C.5 La mémoire partagée répartie

Un service de mémoire partagée répartie (*MPR*) permet à des processus s'exécutant sur des sites interconnectés par un réseau de partager un même espace d'adressage [Nitzberg et al.91]. Ainsi, chaque modification de la mémoire partagée par l'un des processus va être perçue par les autres processus puisque ils partagent la même mémoire. Le concept de *MPR* permet une partage souple et efficace d'informations. De plus une *MPR* facilite la mise en œuvre de certaines fonctions de traitement des évolutions dynamiques de la configuration d'un collecticiel, comme par exemple l'intégration d'un nouvel arrivant ou la mise à jour des privilèges d'un groupe d'utilisateurs, en raison de la propagation automatique à travers le système de tout mis à jour de la mémoire.Un problème cependant, posé par la mise en œuvre d'une *MPR*, est celui de la gestion de la cohérence de la mémoire. La plupart des *MPR* emploient des protocoles de gestion de la cohérence qui ne sont pas compatibles avec les besoins des applications coopératives (voir C.3). Une solution intéressante à ce problème est présentée par la *MPR ARIAS* qui fournit un service générique pour la gestion de la cohérence qui permet aux applications de spécifier et d'implanter les protocoles de cohérence qui conviennent le mieux à leurs besoins [Cortez96]. D'un autre côté, le concept de *MPR* ne fournit pas un support explicite pour la gestion des groupes d'utilisateurs. Finalement, bien que une *MPR* se prête bien pour implanter un service de conscience de groupe peu des *MPR* actuelles fournissent un tel service.

C.6 Les modèles de protection de données

Un aspect important dans tout système multi-utilisateurs est celui de la protection des données. Le but d'un service de protection est d'*inhiber* toute manipulation des objets maintenus dans le système par des utilisateurs non autorisés. Le principe d'un service de protection est de déterminer si un acteur P (application exécutée au profit d'un utilisateur) a le droit d'appliquer une opération (lire, écrire, supprimer, ...,etc) sur un objet O (fichier, objet dans un dessin, case dans un tableau de calcul). La mise en œuvre d'un service de protection doit satisfaire

plusieurs contraintes parmi lesquelles nous citons les suivantes :

- Le coût de l'exécution de la fonction d'évaluation de droits d'accès doit être le plus faible possible. Ce besoin est motivé par l'exécution très fréquente de cette fonction (à chaque demande d'accès à un objet). Un coût élevé peut affecter gravement les performances générales du système.

- Les utilisateurs doivent avoir une vision claire des règles d'accès mises en vigueur. En d'autres termes, le système doit permettre d'identifier facilement les droits des différents acteurs sur un objet donné ainsi que les droits attribués à un acteur précis.

- Les privilèges attribués aux acteurs peuvent changés dynamiquement. Ceci est particulièrement vrai dans le contexte du *TCAO* [Shen et al.92, Kanawati95]. Il est donc nécessaire que le système de protection permette de changer les règles d'accès d'une manière dynamique. Pour des raisons évidents de sécurité, il est souhaitable que le changement d'une règle de protection prenne effet immédiatement.

- Généralement les utilisateurs finals du système seront amenés à définir eux mêmes les règles d'accès des autres utilisateurs sur les objets qu'ils créent. Il est donc souhaitable que le système fournisse des primitives de manipulation et de définition de règles d'accès qui soient simples à employer.

Le grand nombre d'entités (acteurs, objets et droits) qui intervient dans un système réparti (et dans un système coopératif) complique davantage la mise en œuvre des systèmes de protection. Plusieurs modèles de gestion et d'expression de droit d'accès sont proposés dans la littérature. Nous présentons brièvement dans la suite les deux approches les plus employées : *les systèmes à base de matrice de protection* et *les systèmes à base de rôles.*

1. Les systèmes fondés sur l'emploi de la matrice de protection : le concept de la matrice de protection est initialement proposé par *Lampson* [Lampson74]. Les colonnes de la matrice représentent les acteurs définis dans le système tandis que les lignes représentent les objets disponibles. À l'intersection d'une ligne et d'une colonne se trouve le droit attribué à l'acteur associé à la colonne sur l'objet indexé par la ligne courante. Afin d'éviter de manipuler des matrices de grande taille et qui sont souvent creuses, deux stratégies d'implantation de matrice de protection sont employées : implantation sous forme de *listes d'accès* (*ACL*) et implantation sous forme de listes de *capacités.*

   Selon la première approche chaque objet est associé une liste qui définit les acteurs qu'ont le droit de l'accéder (ça revient à couper la matrice en lignes). La deuxième approche consiste à associer à chaque acteur une liste

qui donne les privilèges qui lui sont attribués (voir figure C.1). Le concept de matrice de protection est assez intuitif. Il est simple à mettre en œuvre. Les deux schémas d'implantation (ACL et capacités) permettent de réduire le coût de l'évaluation de droit d'accès. Cependant, ce type de systèmes offre peu de support pour la gestion de groupes. De plus les privilèges attribués aux utilisateurs sont en somme très peu lisibles. Pour illustrer notre point de vue, prenons l'exemple concret du système *Unix*. Le système de protection dans *Unix* est implanté sous forme d'ACL. Des groupes d'utilisateurs peuvent être définis afin de faciliter l'expression des règles de protection. La composition des groupes est définie dans les deux fichiers /etc/passwd et /etc/group. Il est donc très facile de déterminer les groupes auxquels un utilisateur appartient. Par contre, il est moins évident de *recenser* les privilèges attribués à un groupe, à défaut de parcourir l'ensemble du système de fichiers. La décentralisation totale d'attribution des droits aux groupes (n'importe que utilisateur peut donner à un groupe le droit sur les fichiers qu'il administre) rend la notion de groupe peu utile en terme d'unité d'organisation.

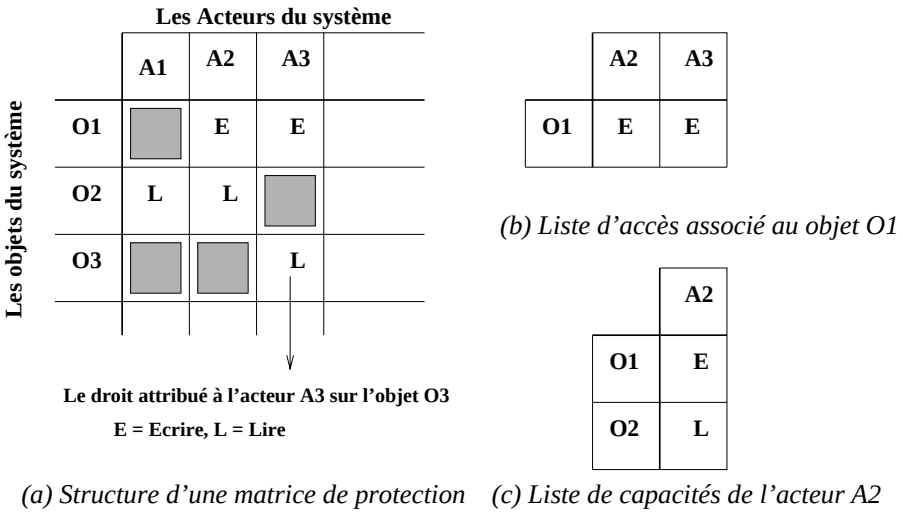
2. Les systèmes à base de rôles. Un rôle est une entité sémantique qui définit un ensemble de privilèges dans le système. Tandis que la notion de groupe, évoqué ci-dessus, désigne un ensemble d'utilisateurs la notion de rôle désigne un ensemble de privilèges. Un rôle peut modéliser une compétence particulière (ex. informaticien) ou une autorité (ex. administrateur système). Plusieurs types de systèmes de protection à base de rôles sont définis. Une synthèse des différentes approches de définition des rôles est donnée dans [Sandhu96]. Les droits attribués à un utilisateur sont déterminés par la combinaison des relations suivantes :

   (a) La relation rôle-privilège qui décrit les droits associés à un rôle.

   (b) La relation rôle-rôle où certains rôles peuvent hériter certaines de leurs attributions d'autres rôles [Shen et al.92, ?].

   (c) La relation rôle-utilisateur qui définit l'ensemble des rôles attribués à chaque utilisateur.

   L'articulation de ces trois relations permet de *personnaliser* facilement les privilèges des utilisateurs. De plus, ce schéma de protection semble être plus naturel, de point de vue des utilisateurs vu la ressemblance avec les mécanismes de sécurité (non informatique) généralement employés dans les organisations.



(a) Structure d'une matrice de protection *(b) Liste d'accès associé au objet O1* *(c) Liste de capacités de l'acteur A2*

FIG. C.1 - : Concepts et schéma de mise en œuvre d'une matrice de protection.

C.7 Synthèse

Dans les sections précédentes, nous avons examiné le support offert par les actuels systèmes répartis pour la construction et l'exécution des collecticiels. L'étude d'analyse est récapitulée dans le tableau C.1. Les notations suivantes sont employées : • = support convenable , ⊗ = Support partiel, ○ = Peu de support et - = non appliquable.

Services/Besoin	Conscience de groupe	Partage	Gestion de groupes	Reconfiguration dynamique	Ouverture
RPC	○	⊗	○	-	○
Diffusion	○	⊗	•	-	○
Bus de communication	⊗	⊗	○	•	⊗
Duplication	○	⊗	⊗	⊗	-
Concurrence transparente	○	○	○	-	-
Transparence de pannes	○	○	○	⊗	-
MPR	○	•	⊗	⊗	-
Matrice de protection	○	⊗	○	○	-
Rôles de protection	○	⊗	⊗	○	-

TAB. C.1 - : Évaluation de la satisfaction des besoins des collecticiels par les systèmes répartis.

En résumé, les systèmes répartis ne fournissent qu'un support partiel pour les collecticiels. Certains besoins du travail coopératif sont plus au moins bien

supportés, c'est notamment le cas du besoin de partage d'informations et de communication. D'autres besoins sont bien moins supportés comme par exemple la conscience de groupe et l'ouverture du support système. Une application coopérative peut bénéficier des services de base fournis par les supports systèmes disponibles pour construire les services dont elle a besoin. Notre conclusion est en fait prévisible. Elle est dictée par l'objectif même des systèmes répartis qui visent à supporter une large éventail de types d'applications (transactions bancaires, contrôle aérien, messagerie,..., etc). Il est difficile de concevoir un système qui soit à la fois générique et qui offre un support complet pour des classes variées d'applications.

Annexe D

Le protocole dOPT - preuve d'incorrection

Le protocole *dOPT* est implanté dans le cadre de l'éditeur de texte coopératif *Groove* [Ellis et al.89]. Le principe de *dOPT* est de forcer l'exécution des opérations échangées entre les sites selon l'ordre causal. Des horloges vectorielles sont employées afin de détecter l'ordre causal des événements. Un processus de site P_i maintient un *vecteur d'états* S dont la dimension est le nombre des sites participants. Deux vecteurs d'états sont comparés au moyen des relations suivantes :

- $S_i > S_j - \exists n : S_i[n] > S_j[n]$.
- $S_i < S_j - \forall n : S_i[n] \leq S_j[n]$.
- $S_i = S_j - \forall n : S_i[n] = S_j[n]$.

Lorsque P_i génère une opération Op l'opération est immédiatement exécutée sur le site local δ_i, l'entrée dans le vecteur d'état $S[i]$ est incémentée et un événement de notification est envoyé aux autres sites. Un événement est un quaruplets $E = (i, S_i, Op, p)$ o—ù i est l'identificateur du site, S_i son vecteur d'état, Op est l'opération générée et p est une valeur de priorité de l'opération.

Le protocole fonctionne comme suit. À la réception d'une opération Op sur un site δ_j, le processus du site compare le vecteur d'état local avec le vecteur associé à l'opération reçue. Trois cas de figure peuvent avoir lieu :

- $S_i > S_j$ l'opération O ne peut pas être exécutée sur le site δ_j car le site δ_i a exécuté des opérations qui ne sont pas encore reçues sur δ_j. Op est alors mise dans une file d'attente.
- $S_i = S_j$ l'opération Op est exécutée immédiatement.
- $S_i < S_j$ l'opération Op doit être transformée car le site δ_j a exécuté d'autres opérations qui sont concurrentes à Op. *dOPT* cherche la plus récente opération dans la liste d'historique locale dont l'horloge soit inférieur ou égale

à l'horloge de Op. Soit Op_A l'opération ainsi déterminée. L'opération reçue est transformée en fonction du segment de l'historique local qui commence par l'opération Op_A.

Allison et *Livesely* montrent dans [Allison et al.94] que le protocole $dOPT$ ne vérifie pas la propriété de la correction dans des situations dites de *concurrence partielle*. Un ensemble d'opérations sont en concurrence partielle si une opération au moins est en concurrence avec un série d'opérations toutes causalement liées entre elles. La figure D.1 illustre la situation la plus élémentaire de la concurrence partielle.

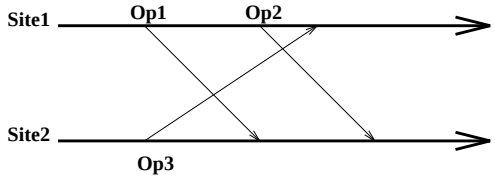


FIG. D.1 - : Définition de la concurrence partielle.

Pour illustrer la défaillance de $dOPT$ nous consiéderos l'exemple (4.6) donné dans le chapitre 4 (4.5.4.2). Selon $dOPT$ les séquences d'exécution suivantes auront lieu sur les sites impliqués :

Exécution sur le site δ_1 les deux opérations Op_1 et Op_2 sont exécutées séquentiellement. Après leurs exécution l'horloge locale a la valeur (0,2). À la réception de Op_3 dont l'horloge est (0,0), l'algorithme $dOPT$ transforme cette opération d'abord en fonction de Op_1 puis en fonction de Op_2. Dans les deux cas l'opération Op_1 reste telle qu'elle est car le site δ_1 est plus prioritaire que δ_2. La couleur de l'objet o_1 passe donc au rouge.

Exécution sur le δ_2 conformément à l'algorithme $dOPT$, l'opération Op_3 est d'abord exécutée et ajoutée à la liste de l'historique locale H_2. L'opération est envoyée au site δ_1 avec l'horloge (0,0). Après l'exécution de Op_3 l'horloge locale du site passe à (1,0). Le site va recevoir Op_1 avec l'horloge (0,0). L'horloge de Op_1 étant inférieur à l'horloge locale, l'opération doit être transformée en fonction des opérations exécutées sur le site δ_1. $dOPT$ cherche la plus récente opération dans H_2 dont l'horloge soit inférieur ou égale à l'horloge de l'opération reçue. Dans le cas actuel, l'opération Op_1 sera transformée en fonction de Op_3. Le site δ_1 étant plus prioritaire que le site δ_2, l'opération Op_1 est transformée en NOP et est ajoutée à H_2. L'objet o_1 est toujours en rouge. L'horloge locale passe à la valeur (1,1). Maintenant à la réception de Op_2 dont l'horloge est (0,1), une nouvelle transformation doit être effectuée. Or la plus récente opération dans H_2 dont l'horloge est est inférieur à celle de Op_2 est l'opération Op_1. Les deux opérations Op_1 et Op_2 ont la même priorité, l'opération Op_2 est exécutée telle qu'elle est, la couleur de l'objet o_1 passe au bleu. Ainsi à la fin de l'échange des opérations le site δ_1 a l'objet o_1 en bleu tandis que le même objet a la couleur rouge sur le

site δ_2. L'incohérence résulte du fait que pour transformer une opération, *dOPT* prends en compte seules les opérations dont l'horloge est plus grande que l'horloge de l'opération qui arrive en retard. Or certaines opérations plus anciennes que l'opération à transformer peuvent avoir elles aussi subi des transformations (c'est le cas de Op_1) et par conséquent la nouvelle opération doit être transformée aussi en fonction des transformations des opérations qui la précèdent.

Annexe E

Liste de Publications

Cette thèse a donné lieu aux publications suivantes :

Revues internationales

- R. KANAWATI, *LICRA : Lock-free Interactive Concurrency Resolution Algorithm for Real Time Groupware Applications* , Parallel Computing Journal, 22(1997) pp. 1733-1746.

Conférences avec comité de lecture

- S. BENATALLAH, R. KANAWATI, R. BALTER et M. RIVEILL, *CoopScan : Une plate-forme générique pour le développement de collecticiels*, IHM'95, Toulouse, Octobre 1995.

- R. KANAWATI et M. RIVEILL, *Access Control Model for Groupware Applications*, Adjunct Proceedings of the British Conference on Human Computer Interaction (HCI'95), Huddersfield, UK, 1995.

- R. BALTER, S. BENATALLAH et R. KANAWATI, *CoopScan : A Platform for Building Synchronous Groupware*, International Conference on Human Computer Interaction - Tokyo, 1995.

Conférences nationales et ateliers de recherche

- R. KANAWATI, *Replicated Data Management Algorithm for Distributed Synchronous Groupware Applications*, Proceedings of the 1st Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS'96), KEFI-1996-09 TR - Miskolc, Hungary, October 1996.

- R. KANAWATI, *Un modèle de protection de données pour les applications coopératives*, Actes de journées jeunes chercheurs en système répartis - IRISA, Rennes, Octobre 1995.

- R. KANAWATI, *Groupware : Architecture and control issues*, Doctoral Consortium, HCI'95 - Huddersfield, UK, August 1995.

- S. BENATALLAH et R. KANAWATI, *Architecture logicielle pour les collecticiels synchrones*, Actes de journées françaises sur l'ingénerie de l'interface homme mahine (IHM'94) , Lille, Novembre, 1994.

Rapports techniques

- R. BALTER, S. BENATALLAH, R. KANAWATI et M. RIVEILL, *Collecticiels synchrones : Analyse des besoins et étude des architectures*, (SIRAC 10-96), IMAG-INRIA, 1996.

- R BALTER, S. BENATALLAH, R. KANAWATI et M RIVEILL, *CoopScan : Une plate-forme générique pour la construction de collecticiels synchrones*, (SIRAC-11-96), IMAG-INRIA, Juin 1996.

Bibliographie

[Abowd et al.92] Abowd (G.), Coutaz (J.) et Nigay (L.). – Structuring the space of interactive system properties. *In : Proceedings of the IFIP TC2/WG2.7 on Engineering for Human Computer Interaction.* – Ellivuroi, Finland, 1992.

[Allison et al.94] Allison (C.) et Livesey (M.). – Coping with concurrency in real time groupware. *In : IEEE Conference on Distributed and Multiprocessor Systems.*

[Almasi et al.94] Almasi (G.), Babadi (A.), Brandt (W.), Butcher (A.), Callahan (J.), Cleetus (K.), Fotta (M.), Gollapudy (C.), Gradestky (N.), Iyer (S.), Jagannathan (V.), Karinthi (R.), Lawson (R.), Nichols (D.), Raman (R.), Shank (R.), Sobolewski (M.), Srinvas (K.) et Zhang (X.). – Functional specification for collaboration services. *In : third workshop on enabling technologies infrastructure for collaborative entreprise.* IEEE. – Morgantown, West Virginia, April 1994.

[Amado et al.93] Amado (G.) et Guittet (A.). – *Dynamique des communications dans les groupes.* – Paris, Armand Colin, 1993.

[AW96] Abdel-Wahab (H.). – The design and implementation of an internet conference information system. *Journal of internetworking research and experience*, vol. 91 (1-2), May 1996.

[Baecker et al.93] Baecker (R.), Nastos (D.), Posner (I.) et Mawby (K.). – The user centred iterative design of collaborative writing software. *In : Proceedings of the ACM INTERCHI conference on human factors in Computing Systems.* – Amestrdam, 1993.

[Balter et al.91] Balter (R.), Banâtre (J.-P.) et Krakowiak (S.). – *Construction des Systèmes d'Exploitation Répartis.* – INRIA - Collection Didactique, Avril 1991.

[Balter et al.95] Balter (R.) et Krakowiak (S.). – Objectifs et plan du travail du projet sirac. – Rapport technique SIRAC RT-1-SIRAC, IMAG-INRIA, juin 1995.

[Balter et al.96a] Balter (R.), Benatallah (S.), Kanawati (R.) et Riveill (M.). – Collecticiels synchrones: analyse des besoins et études des architectures. – Rapport technique, RT-10-96, IMAG-INRIA, 1996.

[Balter et al.96b] Balter (R.), Benatallah (S.), Kanawati (R.) et Riveill (M.). – Coopscan : une plate-forme générique pour la construction de collecticiels. – Rapport technique, RT-9-Sirac, IMAG-INRIA, 1996.

[Bannon et al.91] Bannon (L.) et Schmidt (K.). – *Studies in Computer Supported Coopeartive Work, Theory, Practice and Design*, chap. CSCW : four charecters in search of a context. – North-Holland, 1991.

[Benatallah et al.94] Benatallah (S.) et Kanawati (R.). – Architectures systèmes pour les collecticiels synchrones. *In : Actes des journnées française sur l'ingenerie de l'IHM (IHM'94).* – Lille France, December 1994.

[Benford et al.95] Benford (S.), Greenhalgh (C.) et Snowden (D.). – Collaborative virtual environments. – HCI'95 Tutorial Notes, Huddersfield, UK, August 1995.

[Bentely et al.92a] Bentely (R.), Hughes (J.), Randall (D.), Rodden (T.), Sawyer (P.), Shapiro (D.) et Sommerville (I.). – Ethnographically-informed system design for air traffic control, October 1992.

[Bentely et al.92b] Bentely (R.), Rodden (T.), Sawyer (P.) et Sommerville (I.). – An architecture for tailoring cooperative multi-user displays. *In : CSCW'92.* – ACM.

[Bentley et al.97] Bentley (R.), Appelt (W.), Busbach (U.), Hinrichs (E.), Kerr (D.), Sikkel (S.), Trevor (J.) et Woetzel (G.). – Basic support for cooperation work on the world wide web. *International Journal of Human Computer studies*, April 1997.

[Bernstein et al.87] Bernstein (P. A.), Hadzilacos (V.) et Goodman (N.). – *Concurrency Control and Recovery in Database Systems.* – Addison-Wesley, 1987.

[Beust93] Beust (C.). – Conception d'outils destinés à assister au développement d'applications distribuées. – Thèse de doctorat, Université de Nice, 1993.

[Birman et al.94] Birman (K.) et Renesse (R. Van). – Reliable distributed computing with the ISIS toolkit. *IEEE Computer Society Press*, 1994.

[BL91] Beaudouin-Lafon (M.). – The graph widget - user's manuel. – Rapport technique, Université paris-sud, LRI, June 1991.

[Bonfiglio et al.91] Bonfiglio (A.), Malatesta (G.) et Tisato (F.). – *Studies in CSCW*, chap. Conference toolkit: A framework for real time conferencing. – Elsevier-Sciences (North-Holland), 1991.

[Chung et al.93] Chung (G.), Jeffay (K.) et Abdel-Wahab (H.). – Accommodating latecomers in shared window systems. *IEEE Computer*, vol. 26 (1), January 1993.

[Chung et al.94] Chung (G.), Jeffay (K.) et Abdel-Wahab (H.). – Dynamic participation in a computer-based conferencing system. *Journal of computer communications*, vol. 17 (1), January 1994.

[Comer92] Comer (D.). – *TCP-IP : Architecture, protocoles et applications.* – Inter-Editions, 1992.

[Condon92] Condon (C.). – *CSCW: Cooperation or Conflict?*, chap. The Computer Won't Let Me: Cooperation, Conflict and the Ownership of Information. – Springer-Verlag, January 1992.

[Cortes et al.95] Cortes (M.) et Mishra (P.). – Replicated servers for on-line groupware. *In : Proceedings of Concurrent engineering: research and applications.* – Washington, August 1995.

[Cortez96] Cortez (E.). – La cohérence sur mesure dans un service de mémoire virtuelle partagée. – Thèse de doctorat, INPG, 1996.

[Cosquer et al.94] Cosquer (F.) et Veríssimo (P.). – *Survey of Selected Groupware Applications and Supporting platforms.* – 1994.

[Coulouris et al.94] Coulouris (G.) et Dollimore (J.). – A security model for cooperative work. *In : ACM European SIGOPS Workshop.* – Dagstul, September 1994.

[Coutaz90] Coutaz (J.). – *Interface homme ordinateur, conception et réalisation.* – Dunod Informatique, 1990.

[Croisy94] Croisy (P.). – Modèle multi-agent et conception d'applications coopératives interactives. *In : IHM'94.* – Lille, 1994.

[Croisy95] Croisy (P.). – Collecticiel temps réel et apprentissage coopératif : des aspects sociaux et pédagogiques jusqu'au modèle multi-agent de l'interface de groupe. – Thèse de doctorat, Université de Lille, Janvier 1995.

[Crowley et al.90] Crowley (T.), Milazzo (P.), Baker (E.), Forsdick (H.) et Tomlinson (R). – Mmconf: An infrastructure for building shared multimedia applications. *In : CSCW'90*, p. 329. – Los Angeles, October 1990.

[Decouchant et al.96] Decouchant (D.), Quint (V.) et Romero (M.). – *Groupware and authoring*, chap. Structured and distributed cooperative editing in a large scale network. – Academic press, 1996.

[Derycke et al.95] Derycke (A.) et Hoogstoel (F.). – Le travail coopératif assisté par ordinateur : quels enjeux pour les concepteurs. *In : Actes du XIII congres INFORSID.* – Grenoble, France, June 1995.

[Diaz92] Diaz (M.). – A logical model of cooperation. *In : Proceedings of the IEEE third workshop of future trends of distributed computing systems.*

[Dourish et al.92] Dourish (P.) et Belloti (V.). – Awareness and coordination in a shared workspace. *In : CSCW'92.* – Toronto, 1992.

[Dourish95a] Dourish (P.). – Developping a reflective model of collaborative systems. *ACM transactions on computer-human Interaction*, March 1995.

[Dourish95b] Dourish (P.). – The parting of the ways: Divergence, data management and collaborative work. *In : CSCW'95.* – Sweeden, September 1995.

[Dyre et al.95] Dyre (R.), Green (R.), Pitts (M.) et Millward (G.). – What's the flamming problem ? or computer mediated communication - deindivduating or disinhibing? *In : People and Computers X*, éd. par M. Kirby (A. Dix J. Finlay). HCI'95. – Huddersfield - UK, August 195.

[Edwards94] Edwards (W. K.). – Session management for collaborative applications. *In : Proceedings of the ACM conference on Computer Supported Collaborative Work, CSCW'94.* – Chapel-Hill, October 1994.

[Edwards95] Edwards (W.). – *Coordination Infrastructures in Collaborative Systems.* – PhD thesis, Georgia Institute of Technology, 1995.

[Ellis et al.89] Ellis (C.) et Gibbs (S. J.). – Concurrency control in groupware systems. *In : ACM SIGMOD.* p. 399. – Portland, Oregon, June 1989.

[Ellis et al.94a] Ellis (C.) et Wainer (J.). – A conceptual model of groupware. *In : CSCW'94 Conference on Computer Supported Cooperative Work.* – Chapel-Hill, North Carolina, USA, 1994.

[Ellis et al.94b] Ellis (C.) et Wainer (J.). – Goal-based models of collaboration. *Collaborative Computing*, vol. 1 (1), March 1994.

[Fidge88] Fidge (C.). – Timestamps in message-passing systems that preserve the partial ordering. *In : the 11th Australian Conference.* – Australia, october 1988.

[Goldberg et al.92] Goldberg (Y.), Safran (M.) et Shapiro (E.). – Active mail - a framework for implementing groupware. *In : Sharing perspectives - Proceedings of the ACM conference on CSCW.* – Canada, November 1992.

[Greenberg et al.94] Greenberg (S.) et Marwood (D.). – Real time groupware as a distributed system: Concurrency control and its effect on the intertface. *In : CSCW'94.* – Chapel Hill, NC, October 1994.

[Greenberg91] Greenberg (S.). – Personalizable groupware: Accomodating individule roles and group differences. *In : Proceeding of the ECSCW European Conference of CSCW.* – Amsterdam, September 1991.

[Grundin88] Grundin (J.). – Why cscw applications fail: Problems in the design and evaluation of orgnisational interfaces. *In : Proceedings of the first conference on CSCW*. ACM. – Portland OR, 1988.

[Grundin94] Grundin (J.). – Groupware & social dynamics: Eight challenges for developeres. *Communication of the ACM*, vol. 37 (1), January 1994, pp. 93–105.

[Gust88] Gust (P.). – Sharedx : X in a distributed group environment. *In : the second annual X technical conference*, éd. par MIT.

[Gutwin et al.96] Gutwin (C.), Roseman (M.) et Greenberg (S.). – *A Usability Study of Awareness Widgets in a Shared Workspace Groupware System.* – Technical report, University of Calgary, Canada, 1996.

[Guyennet et al.97] Guyennet (H.), Lapayre (J-C.) et Tréhel (M.). – CAliF : une plate-forme de développement de collecticiels à base de mémoire partagée répartie. *Calculateurs parallèls*, vol. 9 (2), July 1997.

[Hill92] Hill (R.). – The abstraction-link-view paradigme. *In : Proccedings of the ACM Conference on Computer Human Interaction, CHI'92.*

[Hughes et al.92] Hughes (J.), Randall (D.) et Shapiro (D.). – Faltering from ethnography to design. *In : Proceedings of CSCW'92.* ACM. – Canada, October 1992.

[Hughes et al.94] Hughes (J.), King (V.), Rodden (T.) et Andersen (H.). – Moving out from the control room: Enthography in system design. *In : Proceedings of CSCW'94*, éd. par ACM. – Chapell-Hill, CA, October 1994.

[Ishii et al.91] Ishii (H.) et Ohkubo (M.). – Message driven groupware design based on an office procedural model. OM-1. *Journal of Information Processing*, vol. 14 (2), 1991.

[Kanawati95] Kanawati (R.). – Un modèle de protection de données pour les applications coopératives. *In : Journées Jeunes chercheurs en systèmes répartis.* – Rennes, Octobre 1995.

[Karsenty et al.93a] Karsenty (A.) et Beaudouin-Lafon (M.). – An algorithm for distributed groupware applications. *In : Proceedings of*

the 13 th Intl. Conf. on Distributed Computing Systems ICDCS'93. – Pittsburgh, May 1993.

[Karsenty et al.93b] Karsenty (A.), Tronche (C.) et Beaudoin-Lafon (M.). – Groupdesign : Shared editing in a heterogeneous environment. *Computing Systems (Usenix)*, vol. 6 (2), Spring 1993.

[Karsenty94a] Karsenty (A.). – Groupdesign : Un collecticiel synchrone pour l'edition de documents. – Thèse de Doctorat, Université de Paris sud, Février 1994.

[Karsenty94b] Karsenty (A.). – Le collecticiel : de l'interaction homme-homme à la communication homme-machine-homme. *Technique et Science Informatiques*, vol. 13 (1), 1994.

[Knister et al.93] Knister (M.) et Parakash (A.). – Issues in the design of a toolkit for supporting multiple group editors. *Usenix Computing Systems*, vol. 6 (2), 1993.

[Lamport78] Lamport (L.). – Time, clocks and the ordering of events in a distributed system. *Communication of the ACM*, vol. 21 (7), 1978.

[Lampson74] Lampson (B. W.). – Protection. *ACM Operating System Review*, vol. 8 (1), 1974, p. 18.

[Lenormand96] Lenormand (E.). – Coordination d'activités dans un système réparti. – Thèse de doctorat, Université Joseph Fourier, November 1996.

[Lochovsky et al.88] Lochovsky (F.), Hogg (J.), Weiser (S.) et Mendelzon (A.). – OTM: Specifying office tasks. *In : ACM Proceedings on the Conference on Office Information Systems*. – Palo Alto, CA, 1988.

[Markus et al.90] Markus (M.) et Connolly (T.). – Why cscw applications fail: Problems in the adoption of interdependent work tools. *In : Proceedings of CSCW'90*. ACM.

[Mccarthy et al.94] McCarthy (J. C.) et Monk (A. F.). – Channels, conversation, cooperation and relevance: All you wanted to know about communication but were afraid to ask. *Collaborative Computing*, vol. 1 (1), March 1994.

[Moran et al.93] Moran (T.), McCall (K.), Melle (B. Van), Pedersen (E.) et Haasz (F.). – *Designing Principles for Sharing Data*

in Tivoli, a Whiteboard Meeting-Support Tool. – McGraw-Hill, 1993.

[Nastos92] Nastos (D.). – *A Structured Environment for Collaborative Writing.* – PhD thesis, University of Toronto, 1992.

[Navarro et al.93] Navarro (L.), Prinz (W.) et Rodden (T.). – Cscw requires open systems. *Computer Communication*, vol. 16 (5), May 1993.

[Nitzberg et al.91] Nitzberg (B.) et Lo (V.). – Distributed shared memory: A survey of issues and algorithms. *Computer*, vol. 24 (8), August 1991.

[NW et al.92] Newman-Wolfe (R. E.), Webb (M. L.) et Montes (M.). – Implicit locking in the ensemble concurrent object oriented graphics editor. *In : Proceedings of CSCW'92.* – Canada, November 1992.

[Oki et al.93] Oki (B.), Pfluegl (M.), Siegel (A.) et Skeen (D.). – The information bus: an architecture for extensible distributed systems. *Operating System review*, vol. 27 (5), 1993.

[Palanque et al.95] Palanque (Ph.) et Bastide (R.). – Formal specification and verification of cscw using interactive cooperative object formalism. *In : Proceedings of the HCI'95 Conference people and Computer X*, éd. par Kirby (M.), Dix (A.) et Finlay (J.). – Huddersfield, UK, August 1995.

[Paoli et al.94] Paoli (F. De) et Tisato (F.). – Csdl: A language for cooperative systems design. *IEEE Transactions on Software Engineering*, vol. 20 (8), August 1994.

[Patel et al.93] Patel (D.) et Kalter (D.). – A unix toolkit for distributed synchronous collaborative applications. *Computing Systems (Usenix)*, vol. 6 (2), 1993.

[Patterson et al.90] Patterson (J.), Hill (R. D.), Rohall (S. L.) et Meeks (W. S.). – Rendezvous: An architecture for synchronous multi-user application. *In : CSCW'90.* – L. A., October 1990.

[Pons et al.96] Pons (M-C.) et Merialdo (B.). – Un langage de description d'applications coopératives. *In : CRAC'96- Le contrôle réparti dans les applicatons Coopératives.* – Paris, May 1996.

[Quint et al.94] Quint (V.) et Vatton (I.). – Making structured documents active. *Electronic Publishing*, vol. 7 (2), June 1994, p. 53.

[Raynal et al.95] Raynal (M.) et Schiper (A.). – A suite of formal definitions for consistency criteria in distributed shared memories. – Rapport technique, IRISA No. 968, 1995.

[Raynal et al.96] Raynal (M.) et Singhal (M.). – Logical time: Capturing causality in distributed systems. *IEEE Computer*, February 1996.

[Raynal90] Raynal (M.). – Order notions and atomic multicast in distributed systems : a short survey. – Rapport technique, IRISA, Mars 1990.

[Reiss90] Reiss (S.). – Connecting tools using message passing in the field environment. *IEEE software*, vol. 7 (4), 1990.

[Reness et al.96] Reness (R.), Birman (K.) et Maffeis (S.). – Horus: a flexible group communication system. *Communications of the ACM*, vol. 39 (4), April 1996.

[Roseman et al.92] Roseman (M.) et Greenberg (S.). – Groupkit: A groupware toolkit for building real-time conferencing applications. *In : CSCW'92.*

[Roseman et al.93] Roseman (M.) et Greenberg (S.). – Building flexible groupware through open protocols. *In : Proceedings of the ACM Conference on Orgnizational Computing Systems.* – California, 1993.

[Roseman et al.96a] Roseman (M.) et Greenberg (S.). – Building real time groupware with groupkit, agroupware toolkit. *ACM transactions on CHI*, March 1996.

[Roseman et al.96b] Roseman (M.) et Greenberg (S.). – Teamrooms: Network places for collaboration. *In : Proceedings of the International Conference on Computer Supported Cooperative Work Col.*

[Salber et al.95] Salber (D.), Coutaz (J.), Decouchant (D.) et Riveill (M.). – De l'observabilité et de l'honnêteté : le cas du contrôle d'accès dans la communication homme-homme médiatisée. *In : IHM'95.* – Toulouse, Octobre 1995.

[Salber95] Salber (D.). – De l'intercation homme-machine individuelle aux systèmes multi-utilisateurs, l'exemple de la communication homme-homme médiatisée. – Thèse de doctorat, Université Joseph Fourier, September 1995.

[Sandhu96] Sandhu (R.). – Role-based access control models. *IEEE Computer*, february 1996.

[Sasse et al.96] Sasse (M. A.) et Handley (M. J.). – *Groupware and authoring*, chap. Collaborative writing with synchronous and asynchronous support environments. – Academic press, 1996.

[Schmidt et al.93] Schmidt (K.) et Rodden (T.). – Putting it all together: requirements for a cscw platform, 1993.

[Shapiro94] Shapiro (D.). – The limits of ethnography: Combining social sciences for cscw. *In : proceeding of CSCW'94*, éd. par ACM. – Chapel-Hill, CA, October 1994.

[Shen et al.92] Shen (H.) et Dewan (P.). – Access control for collaborative environments. *In : CSCW'92*. p. 51. – Toronto Canada, November 1992.

[Shneiderman83] Shneiderman (B.). – Direct manipulation : A step beyond programming languages. *IEEE Computer*, no16, february 1983.

[Shu et al.94] Shu (L. I.) et Flowers (W.). – Teledesign: Groupware user experiment in three-dimensional computer aided design. *Collaborative Computing*, vol. 1 (1), March 1994, pp. 1–14.

[Smith et al.94] Smith (G.) et Rodden (T.). – Access as a means of configuring cooperative interfaces. *Collaborative Computing*, vol. 1 (2), August 1994.

[Sproull et al.91] Sproull (L.) et Kiesler (S.). – *Connections, New ways of working in the Networked Orgnisation.* – MIT press, 1991.

[Tani et al.94] Tani (M.), Horita (M.), Yamashi (K.), Tanikoshi (K.) et Futakawa (M.). – Courtyard: integrating shared overview on a large screen and per-user detail on individual screens. *In : Proceedings of ACM conference on human factors in Computing Systems.* – Boston, April 1994.

[Tatar et al.91] Tatar (J.), Foster (G.) et Bobrow (D.). – Design for conversation : Lessons from cognator. *Int. Journal of Multi Media Systems*, vol. 2 (34), 1991.

[Trevor et al.93] Trevor (J.), Rodden (T.) et Balir (G.). – Cola: A lightweight platform for cscw. *In : Proceedings of the Third*

European Conference on Computer-Supported Cooperative Work, éd. par Michelis (G. De), Simone (C.) et Schmidt (K.).

[Villemur95] Villemur (T.). – Conception de services et de protocoles pour la gestion de groupes coopératifs. – Thèse de doctorat, Université Paul Sabatier, January 1995.

[Welch95] Welch (B.). – *Practical Programming in Tcl and Tk.* – Prentice-Hall, 1995.

[ZV et al.95a] Zorola-Villarreal (R.) et Bastide (R.). – Mesc : Une méthode de l'évaluation de l'utilisabilité des interfaces multi-utilisateurs. *In : IHM'95*, éd. par Cépaduès-Éditions. – Toulouse, France, October 1995.

[ZV et al.95b] Zorola-Villarreal (R.), Pavard (B.) et Bastide (R.). – Simcoop : A tool to analyse and predict cooperation in complexe environments. *In : Fifth International Conference on Human-Machine Interaction and Artificial Intelligence in Aerospace.* – Toulouse - France, September 1995.