

THESE

présentée par

Slim BEN ATALLAH

pour obtenir le titre de

Docteur de l'Université de Savoie

(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)

Spécialité

INFORMATIQUE

Architectures systèmes pour la construction et l'exécution de collecticiels

Soutenue le 27 juin 1997 devant le jury composé de :

M Roland Balter	Président
M Michel Beaudoin-Lafon	Rapporteur
M Bertrand David	Rapporteur
Mlle Laurence Nigay	Examineur
M Michel Riveill	Directeur de thèse

Thèse préparée au sein du projet INRIA SIRAC

Introduction

D'outil de calcul et de stockage de données qu'il était il y a quelques dizaines d'années, l'ordinateur a vu son emploi s'étendre à des domaines d'utilisation variés, notamment aux applications de télécommunications. Cette évolution a été largement influencée par l'essor de la technologie numérique, aussi bien dans le domaine du traitement de l'information que de la communication. Actuellement, son et image sont aisément véhiculés à travers des réseaux informatiques.

Dans le cadre de notre travail, nous nous intéressons à l'étude et au développement de systèmes informatiques permettant à des utilisateurs répartis d'utiliser leurs stations de travail pour communiquer, travailler en commun et organiser des activités de groupe.

1 Travail coopératif assisté par ordinateur et collecticiels

Les collecticiels sont des applications qui permettent à des utilisateurs de partager un environnement commun et de participer à une tâche commune (e.g. la rédaction d'un document) à partir de leurs stations de travail sans avoir à se déplacer pour coordonner leurs actions. De ce fait, les collecticiels allient le traitement local des données sur des stations de travail et la communication de ces données entre des utilisateurs. Ils répondent ainsi aux besoins de plus en plus pressants des entreprises souhaitant réunir "virtuellement" divers intervenants dans différents domaines d'expertise. Ces applications doivent remplir deux fonctions : réaliser la tâche spécifiée, et respecter les règles usuelles d'interaction des utilisateurs pour la réalisation d'une tâche collective. Cela revient à coordonner les activités d'un groupe d'utilisateurs au sein d'un groupe (e.g. déplacer un objet virtuel à plusieurs, éditer un journal). Ces applications permettent d'organiser ces opérations en mettant à la disposition des utilisateurs des outils coopératifs (applications partagées) et des services de coordination (politiques de contrôle pour utiliser convenablement les applications partagées) afin d'atteindre les objectifs fixés.

Terminologie et définitions

Le terme anglo-saxon *CSCW*, désignant "*Computer-Supported Cooperative Work*", est employé dans la littérature pour définir l'ensemble des systèmes informatiques qui facilitent la coopération d'individus autour d'une tâche commune. La traduction française de *CSCW* est **Travail Coopératif Assisté par Ordinateur (TCAO)**. L'autre terme anglo-saxon souvent utilisé en substitution de *CSCW* est *groupware*. Ce terme, initialement employé dans le langage courant des scientifiques américains, a vite connu une utilisation plus universelle dans la littérature informatique. En 1978, Peter and Trudy Johnson-Lenz, pionniers dans le travail coopératif, introduisirent le terme *groupware* pour définir une activité de groupe intentionnelle augmentée d'un support logiciel permettant sa réalisation. "**Collecticiel**" est le terme employé dans la communauté française pour désigner *groupware*.

Depuis l'émergence du *CSCW*, plusieurs applications et produits industriels ont vu le jour. Néanmoins, l'outil le plus diffusé reste le courrier électronique (*electronic mail*) [Sproull and Kiesler 91]. L'utilisation du courrier électronique accompagnée d'une organisation des dialogues et une structuration des échanges de messages a donné naissance à la classe d'applications de téléconférences ([Hiltz and Turoff 78], [Hiltz 84]). Depuis, le *CSCW* est fondé sur la convergence des télécommunications et du traitement informatique des données. Il évoque la possibilité de participer à des conférences à distance en utilisant les technologies audio-visuelles. Le résultat est souvent appelé *Desktop videoconferencing*. L'ordinateur est donc utilisé aussi bien pour réaliser des tâches qui nécessitent de l'assemblage et de la coordination (e.g. la rédaction d'un document), que pour réaliser des tâches fondées sur la communication comme, par exemple, une prise de décision par un groupe (*group decision support system*).

Une définition plus générale de la coopération peut être la suivante : "un ensemble d'interactions médiatisées entre des individus qui partagent un ensemble de ressources communes". Les interactions peuvent être répertoriées en deux classes : des interactions implicites causées par le partage de ressources informatiques communes (données communes), des interactions explicites réalisées par des échanges de messages (messages textuels, données audio ou vidéo) entre les individus.

Aujourd'hui encore, plusieurs controverses sur la nature et la définition des termes *CSCW* et *Groupware* persistent. Par exemple, Malone (cité dans [Coleman 92]) donne la définition suivante du groupware : "*Une technologie informatique utilisée pour assister des utilisateurs dans l'accomplissement d'un travail commun.*" Peter et Trudy Johnson-Lenz qui ont utilisé ce terme dans [Johnson-Lenz 80] et [Johnson-Lenz 82] donne finalement dans [Johnson-Lenz 92] la définition : "*computer-mediated culture*".

Le collecticiel est tributaire de plusieurs disciplines technologiques, sociales et économiques. L'essor que connaît actuellement le domaine du TCAO est essentiellement lié à l'émergence des réseaux de communication et à la large diffusion des moyens informatiques dans les environnements professionnels, éducatifs et domestiques.

Dans une récente étude des collecticiels [Coleman 95], Bob Flanagan (ancien analyste au Workgroup Technologies Inc. et actuel analyste au Yankee Group) estime le marché du collecticiel à 1,7 billion de dollars en 1993. Il prévoit une croissance des ventes atteignant en 1998 5,5 billions de dollars. La plus grande croissance concerne essentiellement le marché du courrier électronique (*email*) et les applications flot de travail (*en anglais Workflow*). Le courrier électronique représentant à lui seul 19% du marché mondial.

L'utilisation des collecticiels au sein des entreprises est essentiellement motivée par un meilleur contrôle des coûts de production, une augmentation de la productivité, une diminution du nombre de réunions, une automatisation des processus de routine (e.g. production de la documentation technique de produits), une amélioration de la coopération entre des équipes géographiquement réparties, une meilleure coordination au sein des équipes, ...

Actuellement, les collecticiels peuvent être regroupés en 5 catégories d'applications : les applications de messagerie/courrier électronique, les applications d'édition de documents partagés, les outils de téléconférence, les outils de Workflow, et les environnements de développements partagés.

Cette thèse traite la problématique de la construction de différent groupes de collecticiels en prenant comme contrainte essentielle la récupération d'applications existantes (e.g. éditeurs de texte, palettes de dessin, navigateurs World Wide Web, ...). Nous allons, dans ce qui suit présenter les objectifs de ce travail et l'approche suivis.

2 Objectifs du travail

L'objectif de ce travail est d'étudier les architectures des collecticiels et les protocoles de contrôle qui régissent le travail coopératif afin de faciliter la conception, la construction, et l'exécution de ces applications dans divers environnements (plates-formes, réseaux, applications). À la différence des travaux qui se sont intéressés aux applications coopératives, notre étude est fondée sur l'analyse des besoins des collecticiels à différents niveaux : de l'"utilisateur", du support d'exécution et du réseau. Le but recherché à travers cette étude est de trouver des solutions génériques pour construire et utiliser des collecticiels. Le terme de "généricité" est employé pour caractériser des solutions pouvant être appliquées pour réaliser des tâches variées dans des environnements d'exécution variés. L'originalité de ce travail se situe à deux niveaux.

- **La conception d'une architecture générique.** L'un de nos objectifs est de proposer aux développeurs d'applications des méthodes, des techniques et des outils qui facilitent et accélèrent la mise en œuvre des applications coopératives.

L'architecture que nous proposons est appelée **CoopScan**. Elle est décrite dans un modèle d'agents communicants qui représentent les différents modules intervenant dans un collecticiel. Conceptuellement, les agents de *CoopScan* implantent les fonctions de coopération qui permettent à plusieurs utilisateurs de partager un espace commun en temps-réel. Cette architecture est fondée sur une stratégie de construction par réutilisation de modules logiciels existants, elle peut être instanciée pour la réalisation de différentes applications sans apporter de modifications majeures à l'implantation des agents. La validation de *CoopScan* est faite à travers le développement de deux applications coopératives : une application de téléconférence et un navigateur World Wide Web coopératif.

- **La conception et la mise en œuvre des protocoles de contrôle.** Le second aspect auquel nous nous intéressons est celui de la conception et la mise en œuvre des fonctions de coopération. Plus particulièrement, nous nous intéressons à la gestion des contextes partagés par les utilisateurs d'un collecticiel et les protocoles de connexion-déconnexion dynamique de participants. Dans ce contexte, nous proposons une classification de ces protocoles en fonction de la répartition de leurs

exécutions. La validation expérimentale de ces protocoles est faite dans l'implantation des deux applications pilotes introduites plus haut.

3 Approche suivie

Dans [Ellis and al 91], les auteurs définissent le collectif (groupware) comme un environnement commun permettant la réalisation de tâches communes. Dyson [Dyson 92] émet la remarque suivante : "plus qu'une définition pour coder ou construire des applications, le groupware est plutôt une manière de définir, structurer et lier des applications, des données et les personnes qui les utilisent." Cette définition nous semble intéressante du fait qu'elle met en évidence les différentes entités qui constitue le collectif. En adoptant une démarche plus fonctionnelle, Ellis et Wainer [Ellis and Wainer 94] proposent un modèle fondé sur trois aspects fonctionnels caractérisant les collectifs :

- l'aspect *ontologique* qui définit les données utilisées lors de la coopération ainsi que les opérations pouvant être accomplies sur ces données. Cet aspect recouvre bien les entités "données" et "applications" citées dans la définition de Dyson,
- l'aspect *coordination* qui définit l'organisation et les relations entre les activités des participants,
- l'aspect *interface* qui définit les interactions entre les participants et le collectif.

Les trois aspects donnés par cette décomposition définissent assez clairement les fondements d'une tâche coopérative. D'ailleurs, certains travaux de recherche s'en sont inspirés pour définir une tâche coopérative. Le modèle du *Trèfle* proposé par le groupe de travail SCOOP du pôle de recherche concerté sur la communication homme-machine, cité dans [Salber 95], est inspiré de la décomposition tripartite d'Ellis. Le modèle du *Trèfle* est fondé sur la définition des services offerts par un système multi-utilisateur, en l'occurrence, de trois espaces : l'espace de production, l'espace de coordination et l'espace de communication. L'espace de production correspond au modèle ontologique d'Ellis augmenté des espaces privés des participants. L'espace de communication offre aux utilisateurs du système la possibilité d'échanger de l'information. Il joue le rôle de support pour *la communication homme-homme médiatisée*.

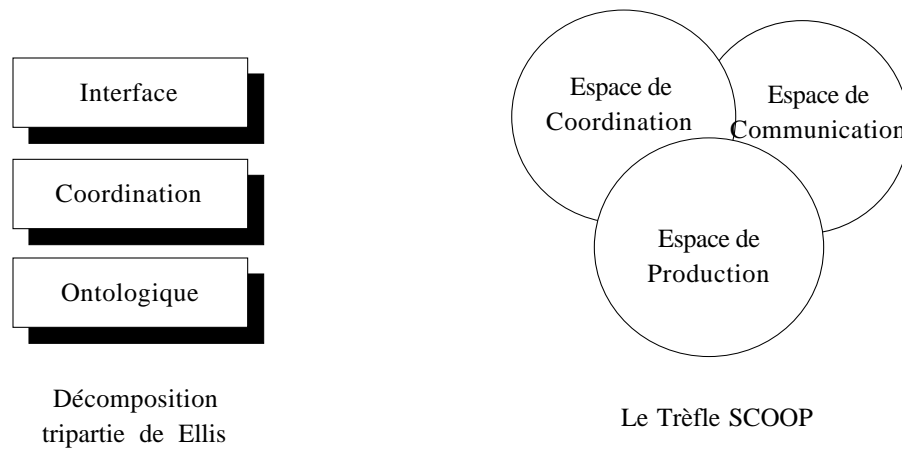


Fig. Introduction . 1 : Modèle du Trèfle inspiré de la décomposition tripartie d'Ellis.

Dans le modèle du Trèfle, les auteurs ne tiennent pas compte de l'aspect *interface*. Cependant, ils introduisent le concept *espace de communication*. Ils considèrent l'interaction homme machine comme une conséquence des services de fond dont il est la vitrine. Cette représentation du collectif est intéressante. Toutefois, ce modèle nous paraît fortement influencé par une classification antérieure du travail coopératif en trois types d'applications : l'édition coopérative (production), le Workflow (coordination), et la vidéoconférence (communication médiatisée).

Notre approche consiste à donner une représentation fonctionnelle commune pour modéliser l'ensemble du travail coopératif, ou du moins, le plus grand nombre de tâches coopératives. Nous ne nous intéressons par conséquent pas à associer un espace de représentation à un type d'application donné. Le modèle est essentiellement la représentation fonctionnelle de l'architecture générique que nous souhaitons développer. Pour cela, nous nous appuyons sur une décomposition fonctionnelle des collecticiels inspirée de la première définition de Peter and Trudy Johnson–Lenz [Johnson–Lenz 80]. Notre modèle coopératif est fondé sur les trois entités suivantes :

- **l'espace de coopération** contient l'ensemble des ressources logicielles partagées lors de la coopération. Il s'agit par exemple, des documents partagés lors d'une édition coopérative, des données échangées lors d'une communication audio entre plusieurs utilisateurs.
- **l'espace de coordination** contient les données de contrôle qui permettent de réaliser la coopération. Généralement, ces données

représentent les règles qui régissent la tâche coopérative. Il s'agit particulièrement des droits d'accès attribués aux utilisateurs,

- l'**espace des acteurs** permet la représentation des utilisateurs dans le modèle coopératif. Cette entité peut être assimilée au concept "Interface" utilisé dans le modèle ontologique d'Ellis.

Nous répartissons les données entre les différentes entités du modèle en fonction de l'usage qui en est fait tout au long de la coopération. Nous distinguons entre deux grandes classes de données : les données qui sont utilisées pour contrôler la réalisation du travail, et les données produites par le travail lui-même.

Les données de contrôle sont contenues dans l'espace de coordination et permettent de contrôler les agissements des acteurs sur l'espace de production. En général, ces données permettent :

- de définir les tâches coopératives pouvant être effectuées,
- de définir les attributions des utilisateurs dans ces tâches,
- de définir les protocoles d'accès aux tâches coopératives, etc.

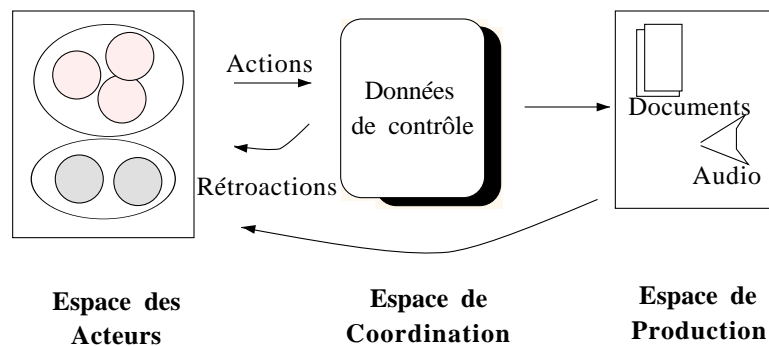


Fig. Introduction . 2 : Modélisation fonctionnelle de la coopération

La modélisation d'un travail coopératif donné consiste à définir les relations qui existent entre les trois espaces définis plus haut. L'intérêt de cette approche est double :

- elle fournit un outil de représentation pour un large spectre de collecticiels, ceux déjà répertoriés en tant que tels, et les applications à venir,
- elle permet de passer facilement à une modélisation formelle de la coopération.

4 Contexte du travail

Nos travaux de recherche sont menés dans le cadre d'un projet de recherche appelé SIRAC (Systèmes Informatiques Répartis et Applications Coopératives) commun à l'INRIA, à l'IMAG et à l'Université de Savoie. L'objectif de SIRAC est de concevoir et de réaliser un environnement pour le développement et l'exécution d'applications réparties. Les recherches sont menées dans les deux thèmes suivants :

- **Construction d'applications réparties.** L'objectif est de fournir des outils répondant à deux besoins : a) construire des applications réparties en combinant des techniques de programmation à base d'objets et des techniques d'intégration de composants ; b) faciliter l'administration, la configuration et l'évolution de ces applications.
- **Services 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. Sa conception tient compte des nouvelles infrastructures matérielles (grandes mémoires virtuelles et réseaux rapides) et utilise des outils génériques pour le maintien de la cohérence.

Des applications, menées en coopération avec des partenaires extérieurs, doivent servir de banc d'essai aux outils, méthodes et services développés dans le projet :

- Applications coopératives, notamment collecticiels synchrones (télé Réunion avec partage de documents, agenda réparti) ; applications nécessitant une gestion coopérative de ressources.
- Support pour la réalisation efficace de serveurs sur des grappes de machines homogènes.

Notre action de recherche s'insère dans le cadre du premier objectif du projet SIRAC, la construction d'applications coopératives. Elle se situe dans le domaine du travail coopératif supporté par ordinateur. Nos objectifs premiers sont d'une part l'adaptation d'applications existantes à la coopération et d'autre part la configuration à la carte de ces applications dans des environnements variés. Dans ce cadre, nous nous intéressons à l'étude des architectures de mise en œuvre des applications coopératives, et la réalisation des fonctions spécifiques de ces applications.

5 Organisation du mémoire

Ce mémoire est composé de deux parties : la première est centrée sur les aspects d'architecture logicielle pour la mise en œuvre de ces applications. La seconde étudie le fonctionnement et les protocoles de gestion de groupes en particulier.

L'idée directrice dans la présentation de ce mémoire consiste à dresser un bilan des fonctions devant être fournies par un collecticiel, à étudier les architectures logicielles pour mettre en œuvre ces fonctions, et, enfin, à étudier une fonction spécifique des collecticiels, la participation dynamique de participants.

Le mémoire est organisé comme suit :

- le chapitre I présente une étude des besoins des collecticiels. Nous y identifions les fonctions requises d'un collecticiel,
- le chapitre II présente une étude des architectures logicielles pour la construction de collecticiel. Plus particulièrement, nous nous intéressons aux infrastructures systèmes pouvant servir au développement de collecticiel. Les aspects abordés sont : la décomposition modulaire du collecticiel, la séparation entre les fonctions de contrôle et les fonctions des applications partagés et les schémas de répartition possibles des modules d'un collecticiel. A l'issue de cette étude, nous présentons l'architecture conceptuelle de *CoopScan* dans un modèle à base d'agents coopérant. Nous donnons également la spécificité de ces agents ainsi que leur instanciation pour la construction de prototypes d'applications coopératives.
- le chapitre III présente une étude des environnements de mise en œuvre de collecticiels, notamment les environnements à base d'objet répartis et l'infrastructure World Wide Web,
- le chapitre IV aborde les services de gestion dynamique de groupes dans les collecticiels, en particulier les protocoles de connexion/déconnexion dynamique de participants,
- le chapitre V décrit l'implantation de la plate-forme *CoopScan*.

Pour conclure, le chapitre VI dresse un bilan de l'étude des architectures logicielles pour les collecticiels synchrones et des protocoles de connexion dynamique de participants dans ces applications. Le bilan consiste en une critique des choix de conception pour la mise en œuvre de la plate-forme *CoopScan* à travers le fonctionnement des prototypes développés.

Introduction

1 Travail coopératif assisté par ordinateur et collecticiels . . .	1
2 Objectifs du travail.	4
3 Approche suivie	5
4 Contexte du travail	8
5 Organisation du mémoire	9

Chapitre I

Collecticiels : définition, classification et étude des besoins

1.1 Introduction

Les collecticiels constituent une classe d'applications qui sont interactives et pouvant être distribuées. En plus des fonctions habituellement fournies par les applications interactives (par exemple, les fonctions permettant la manipulation de données privées, ou les fonctions permettant la visualisation des résultats d'une commande), les collecticiels offrent des fonctions spécifiques destinées à supporter un travail dans lequel plusieurs intervenants agissent en même temps sur un même ensemble d'informations.

Nous donnons dans ce chapitre la définition des principaux concepts utilisés dans le travail coopératif. Nous utilisons ensuite ces concepts pour définir les caractéristiques et les particularités des collecticiels *synchrones* et les besoins fonctionnels auxquels ils doivent répondre. Ce chapitre illustre notre état de réflexion sur les collecticiels à travers les différents travaux de recherche menés dans le domaine des systèmes coopératifs.

Par ailleurs, l'étude présentée dans ce chapitre fait le lien entre les différentes taxonomies proposées pour les collecticiels et les aspects d'architectures systèmes développés dans le chapitre 2 qui constituent l'un des objectifs de cette thèse.

A l'issue de cette étude, notre objectif est de concevoir une architecture générique offrant un ensemble de fonctions qui répondent aux besoins identifiés. L'utilisation de ces fonctions devrait nous permettre de :

- mettre en œuvre des tâches coopératives,
- faciliter la construction de collecticiels synchrones,

- réutiliser les mêmes applications pour réaliser des scénarii coopératifs variés.

1.2 Terminologie

Le terme collecticiel est défini dans le chapitre introduction. Dans cette section, nous définissons la terminologie utilisée dans le domaine du travail coopératif assisté par ordinateur.

Coopération, travail coopératif et tâches communes

La définition littéraire du terme coopérer est *Agir conjointement avec quelqu'un*. La *coopération* en informatique est une instanciation de cette définition en utilisant des moyens logiciels et matériels. Elle désigne une activité informatique menée par un groupe d'utilisateurs dans un espace informatique commun.

Dans ce contexte, le *travail coopératif* désigne un travail faisant intervenir plusieurs utilisateurs sur des ressources informatiques communes.

Le terme *tâche* peut être employé pour définir une partie d'un travail coopératif. Elle peut résulter d'un découpage du travail en plusieurs parties, en fonction de l'un des critères suivants : la spécificité (e.g. édition, dessin, ...), les attributions des participants, le partage d'un même sous-ensemble de ressource, etc.

Souvent, les termes *tâche* et *travail* sont employés pour désigner la même chose.

Acteur

Les utilisateurs d'un collecticiel sont souvent appelés les *acteurs* de la coopération. Le terme *participant* est également employé pour distinguer entre un utilisateur dans un environnement non coopératif et un intervenant dans un collecticiel.

Groupe

Le terme groupe est employé dans plusieurs domaines tels que les sciences sociales, la médecine ou les mathématiques. Il désigne un ensemble d'individus ou d'entités partageant des caractéristiques communes. Ces caractéristiques peuvent être statiques ou dynamiques, et définissent un comportement commun.

Dans le contexte du collecticiel, le groupe est employé pour désigner un ensemble d'utilisateurs impliqués dans un travail commun.

Contexte partagé

Un contexte partagé désigne un ensemble de ressources accessibles par un groupe d'utilisateurs. Ces ressources peuvent être des documents partagés, des informations sur l'état d'exécution d'un processus, etc.

Rôle et vue

Le terme *rôle* est utilisé pour définir les attributions d'un acteur dans un collecticiel (par exemple, lire et écrire dans le document 1, lire dans le document 2).

Le terme *vue* est employé pour définir une *visuelle*⁽²⁾ d'un ensemble d'informations du contexte partagé. La *vue* peut être attachée à un *rôle*. La différence essentielle entre *vue* et *rôle* peut être définie de la manière suivante :

- le rôle permet de définir les actions que peut mener l'acteur sur le contexte partagé,
- la vue permet de définir comment l'acteur perçoit les actions réalisées par d'autres acteurs sur le contexte partagé,
- enfin, différentes vues peuvent représenter la même information, alors que, deux rôles distincts confèrent nécessairement des attributions différentes.

Par exemple, un participant peut avoir un rôle de *président*⁽³⁾ dans une réunion de travail lui conférant le droit d'accepter ou de refuser les requêtes d'adhésion de nouveaux participants. Un utilisateur peut également avoir un rôle de rédacteur ou de simple lecteur sur un document partagé.

1.3 Paramètres des collecticiels

Modes d'interaction : collecticiels synchrones/asynchrones

Les collecticiels offrent plusieurs possibilités pour échanger des informations entre les participants. Un collecticiel est dit *synchrone* si les informations sont

(2) La perception de quelque chose à travers la vue.

(3) "Chairman" en anglais. Ce rôle est généralement attribué à une personne en fonction d'un certain nombre de critères (compétences, à titre honorifique). Nous considérons dans notre cas précis que ce rôle est attribué au premier participant du collecticiel.

échangées selon un mode temps réel. Dans ce cas, les acteurs sont avertis en temps réel des actions menées sur les ressources partagées.

Le mode d'interaction synchrone nécessite que les membres du groupe soient présents en même temps pour effectuer le travail coopératif. Les systèmes de téléconférence sont des exemples représentatifs de ce mode de fonctionnement [Crowley 90]. Ce mode permet un partage immédiat des informations.

Les collecticiels asynchrones, au contraire, correspondent aux applications coopératives dont le support de communication est asynchrone, de type courrier électronique. Le mode d'interaction asynchrone ne nécessite pas la co-présence des différents participants. Ce mode est fréquemment employé dans l'édition coopérative de documents [Decouchant 95]. Le mode asynchrone permet un partage séquentiel des informations.

La classification des collecticiels en fonction du mode d'interaction ne recouvre pas complètement l'ensemble des tâches coopératives. L'édition coopérative représente un exemple de collecticiels qui peuvent être à la fois synchrone et asynchrone.

Le mode de fonctionnement WYSIWIS⁽⁴⁾

Le terme *WYSIWIS*, introduit par Stefik dans [Stefik 87], signifie en français "*ce que tu vois est ce que je vois*". Ce terme est utilisé pour définir le fonctionnement d'un collecticiel fournissant une vue cohérente du contexte partagé, à tout moment de la coopération, pour tous les acteurs.

La différence essentielle entre les termes : synchrone et *WYSIWIS* réside dans le fait que le mode synchrone définit la communication temps réel des actions, alors que le mode *WYSIWIS* caractérise la perception visuelle des actions communiquées (en mode synchrone).

Les relations logiques qui existent entre ces deux concepts sont les suivantes :

- un mode de fonctionnement *WYSIWIS* implique nécessairement un mode d'interaction synchrone,
- alors qu'un mode d'interaction synchrone ne suffit pas pour définir un mode de fonctionnement *WYSIWIS*.

(4) What You See Is What I See.

(5) La notification est un moyen qui permet d'avertir un acteur de l'exécution d'une opération sur une donnée partagée.

Le *WYSIWIS strict* représente l'application extrême du mode de fonctionnement WYSIWIS. Il consiste à avoir une vue identique des informations partagées sur tous les terminaux des participants. La mise en œuvre de ce fonctionnement nécessite un support matériel identique pour tous les acteurs, le cas échéant, des mécanismes de conversion capables d'adapter la taille des fenêtres à celles des écrans.

Différentes approches ont été présentées pour relaxer le *WYSIWIS strict* dans [Stefik 87]. Elles sont fondées sur les aspects *visuel*, *temporel*, *taille* (taille du groupe d'acteurs).

Il est à noter qu'une relaxation *temporelle* poussée peut conduire vers un mode de fonctionnement asynchrone.

Différentes approches sont proposées dans [Gutwin 95] et [Greenberg 95] pour prendre en compte la notification dans le cas d'un fonctionnement du collectif selon un mode WYSIWIS relaxé. Une méthode assez répandue, utilisée dans [Smith 89] et [Baecker 94], consiste à fournir à chaque acteur deux fenêtres : une contenant une vue détaillée, une autre contenant une vue "radar".

La Fig. 1.1 illustre une vue "radar" fournie par l'application *GroupKit* [Roseman 96]. La fenêtre de gauche est une vue complète d'une partie de l'espace de travail, dans ce cas un document texte. La fenêtre de droite représente la vue "radar". L'acteur y voit toutes les parties du document dans une échelle réduite. Cette vue permet ainsi d'identifier les actions de tel participant sur telle partie du document, grâce à une coloration différente des rectangles encadrant certaines zones de la fenêtre.

Fig. 1.1 : La vue "radar" dans GroupKit [Roseman 96]. La fenêtre de droite représente la vue "radar" sur le document alors que la fenêtre de gauche donne une vue plus détaillée d'une partie du document, celle sur laquelle travaille l'acteur local.

Cohérence

Dans les collecticiels synchrones, le mode WYSIWIS strict définit une *cohérence forte* du contexte partagé. Selon ce mode, tous les acteurs ont une vue identique des données partagées.

La *cohérence faible* peut être engendrée par le mode WYSIWIS relaxé selon l'aspect temps. Une telle relaxation donne dans le cas extrême, où aucune hypothèse n'est faite sur le mode d'interaction, un fonctionnement asynchrone.

Cependant, le mode WYSIWIS relaxé selon l'aspect visuel n'affecte pas la cohérence du contexte partagé. Même si les acteurs n'ont pas la même vue des informations, l'état de ces informations est identique pour tous. Dans GroupKit [Roseman 96], bien que les vues "radar" soient différentes d'un participant à l'autre, elles sont générées à partir d'un état cohérent du contexte partagé.

En conclusion, dans les collecticiels, la cohérence est plutôt liée au mode d'interaction qu'aux différentes variantes du mode WYSIWIS.

Granularité

La *granularité* est une caractéristique spatiale qui, à l’opposé, du mode d’interaction, est fréquemment associée aux droits d’accès aux données attribués aux utilisateurs du collectif. La granularité définit l’unité d’information (le mot, le paragraphe, le document, ...) qui peut être accédée simultanément par plusieurs utilisateurs.

Une granularité fine associée à un mode d’interaction synchrone définit un travail *fortement couplé*. Dans le cas contraire, une granularité *forte* associée à un mode asynchrone définit un travail *faiblement couplé*.

Par exemple, un éditeur de texte permettant l’accès simultané à un même mot permet une granularité plus fine qu’un éditeur de texte ne permettant l’accès au document d’un utilisateur à la fois.

Dans cas des collectifs qui utilisent des politiques à *jeton*, si celui-ci est attribué pour accéder tout un document, nous sommes dans le cas d’une granularité est *forte*.

Droits d’accès, droit de parole, politiques à jetons⁽⁶⁾

Les *droits d’accès* associés à un utilisateur sont en partie constitués des attributions qui lui sont conférées par son rôle, et en partie par les droits qui peuvent lui être confiés pendant le déroulement du travail coopératif. Ces droits, dynamiquement attribués, sont communément appelés *droits de parole*.

Dans les collectifs, le droit de parole est échangé entre les participants à l’aide de *politiques à jeton*. Il s’agit de différents protocoles qui permettent aux participants de se communiquer une information de contrôle désignant un *jeton* logiciel.

En général, les politiques utilisent un seul jeton. Ce procédé permet de garantir un accès exclusif aux informations partagées. Toutefois, dans certaines applications, plusieurs jetons peuvent coexister, c’est souvent le cas des applications de jeux virtuels où m utilisateurs parmi n ($m \leq n$) peuvent modifier le contexte partagé.

Les politiques de passage du jeton sont regroupées en deux classes. Celles-ci sont respectivement fondées sur :

(6) "Floor-control"

- la désignation explicite. Un acteur se charge de passer explicitement le jeton à un autre membre du groupe. Celui-ci peut être l'ancien détenteur du jeton ou un acteur auquel revient la responsabilité d'organiser le travail et de coordonner les interventions des participants,
- le passage implicite. Dans ce cas les acteurs n'interviennent pas explicitement dans l'affectation du jeton. L'application se charge de désigner le détenteur du jeton en fonction de l'ordre d'arrivée des demandes pour l'acquisition du jeton (e.g. ordre FIFO), ou en fonction d'une quelconque priorité instaurée entre les participants.

Accès libres, conflits d'accès et synchronisation

L'adoption du mode d'accès libre peut engendrer des conflits et des incohérences de l'espace partagé. Un exemple simple d'une telle situation est observé lorsque deux opérations conflictuelles (opérations d'écriture): *op1*, *op2*, sont effectuées par deux acteurs sur un même objet *shared_obj* (*shared_obj* est dupliqué sur tous les sites participants). Si l'exécution de ces deux opérations n'est pas faite selon le même ordre sur tous les sites, l'état de *shared_obj* sera incohérent.

Afin de garantir la cohérence dans les collecticiels synchrones, le système doit *synchroniser* les actions des différents participants sur tous les sites afin de maintenir un état cohérent pour toutes les données.

GroupDesign [Karsenty 93] est un collecticiel synchrone de dessin partagé qui fournit un mode d'*accès libre* aux données partagées. Deux modes de fonctionnement sont fournis par le système : un mode WYSIWIS strict, et un mode WYSIWIS relaxé. Le mode relaxé est fondé sur une opération de "validation" qui a pour effet de répercuter les actions accomplies par le participant sur les autres sites du collecticiel. Ce mode d'interaction nous paraît plutôt asynchrone.

Pour le fonctionnement en WYSIWIS strict, GroupDesign permet un accès libre aux données qui est similaire à celui utilisé dans Grove [Ellis 91]. GroupDesign fournit un protocole de résolution des conflits basé sur la notion d'*horloges logiques* [Lamport 78].

Chaque action menée sur le contexte partagé provoque la création et l'émission d'un événement estampillé de la valeur de l'horloge locale du site. Les événements renferment chacun les informations : code de l'opération effectuée, identificateur de l'objet touché par l'opération, date de création.

Le protocole maintient un état cohérent de l'ensemble des données partagées. Il favorise l'exécution des actions locales. Le protocole peut, dans certaines configurations, défaire une exécution locale s'il s'avère que des actions conflictuelles faites sur d'autres sites précèdent l'exécution des actions locales.

Gestion dynamique de groupes : protocoles de connexion/déconnexion de participants dans un collecticiel

Un service de gestion de groupe dans collecticiel fournit principalement des protocoles permettant : la connexion de nouveaux membres et la déconnexion d'anciens membres.

La participation dynamique de participants figure parmi les fonctions spécifiques fournies par un collecticiel synchrone. Cette fonction marque le passage d'un utilisateur d'un environnement de travail privé à un environnement de travail coopératif.

Afin de garantir la flexibilité d'utilisation, le collecticiel doit permettre à des retardataires de se connecter pendant le déroulement du travail. La complexité de cette opération consiste à fournir au nouvel arrivant une vue "cohérente" du contexte partagé et à informer les membres du collecticiel de l'arrivée d'un nouveau membre.

De même, le collecticiel doit gérer le départ de certains membres qu'il soit volontaire ou résultant d'une panne quelconque du réseau ou des sites.

En résumé, deux contraintes régissent l'exécution des fonctions de participation et de déconnexion. Elles sont respectivement spatiales et temporelles :

- cohérence du contexte partagé : pour les collecticiels adoptant un mode de fonctionnement WYSIWIS, à l'issue de la phase de connexion, le nouvel arrivant doit avoir une vue identique du **contexte partagé**. Dans ce cas, le protocole garantit une cohérence forte,
- temps de réponse : le traitement des demandes de connexion ou de déconnexion doit être asynchrone. le protocole de connexion doit garantir des **temps** de réponse faibles aussi bien pour le demandeur de la connexion que pour les acteurs du collecticiel.

Remarques

La cohérence du contexte n'est pas absolue, elle est liée au rôle, voire à la vue, qui sont conférés à l'utilisateur au moment de son adhésion au groupe. Par exemple, si un utilisateur n'est autorisé qu'à partager une application particulière

du collecticiel en mode WYSIWIS, le service de gestion de groupe ne lui fournit que le contexte de l'application en question.

La fonction de déconnexion marque la sortie d'un acteur du groupe de participants. Le protocole de déconnexion doit garantir un fonctionnement normal du collecticiel pendant cette phase. En général, le protocole doit s'assurer que le participant n'a pas de jeton, ou qu'il n'occupe pas de rôle pouvant entraver le déroulement du travail (par exemple, le rôle de président doit être cédé avant de quitter la session).

Les opérations de connexion/déconnexion dynamiques sont accomplies par des protocoles qui sont souvent choisis en fonction du type du travail coopératif. Dans certains collecticiels, la connexion d'un utilisateur peut être soumise à l'accord d'un *président*. Dans d'autres applications telles que les téléconférences simulant par exemple un séminaire grand public, la connexion d'un nouvel arrivant est complètement transparente aux acteurs du collecticiel.

Session

Une session permet d'associer : un groupe d'acteurs, un contexte partagé, des rôles, des protocoles d'accès, et des protocoles de connexion/déconnexion, afin de définir selon la dimension *espace* un travail coopératif. La définition complète du travail nécessite la donnée du mode d'interaction adopté. Souvent, le terme *session* est employé pour définir une durée d'exécution d'un collecticiel synchrone.

Les concepts IHM⁽⁷⁾

Plusieurs concepts hérités du domaine interface homme-machine sont également utilisés dans le domaine du collecticiel. Des termes comme l'*observabilité* et l'*honnêteté* décrivant les caractéristiques des interfaces utilisateur dans les collecticiels sont définies dans [Salber 95] :

– l'*observabilité* est la capacité du système à rendre perceptibles à l'utilisateur les variations des données faisant l'objet de la coopération,

– l'*honnêteté* : le système est honnête si le rendu de l'état observable des données est conforme à l'état interne, et si la forme de ce rendu conduit l'utilisateur à interpréter correctement cet état (c'est à dire, le résultat de cette interprétation),

(7) Interface Homme-Machine

(8) Environnement de Développement Partagé

– la *pro-activité* des retours d'information caractérise un comportement qui empêche l'utilisateur d'accomplir une action interdite. Par exemple, un élément de menu en grisé indique à l'utilisateur qu'il est inopérant (caractère observable) et, en même temps, n'active pas de fonction système qui conduirait à une erreur (il est pro-actif).

– la *conformité du temps de réponse* mesure la capacité du système à réagir dans un laps de temps en accord avec l'attente de l'utilisateur.

1.4 Classifications des collecticiels

Plusieurs taxonomies existent pour les collecticiels. Nous donnons dans cette section les plus représentatives d'entre elles.

Taxonomie espace-temps

La taxonomie la plus répandue dans la littérature est celle fondée sur les critères *espace-temps*. Cette classification a été citée dans plusieurs travaux, parmi lesquels [Ellis 91]. Il s'agit de classer les collecticiels dans un repère à deux dimensions espace-temps (cf. Fig. 1.2). L'axe *espace* désigne la distance géographique séparant les acteurs, alors que l'axe *temps* définit le mode d'interaction adopté dans le collecticiel (i.e. synchrone/asynchrone).

Fig. 1.2 : La taxonomie espace-temps. L'édition coopérative permettant des interactions synchrones/asynchrones n'est pas distinctement classée dans cette taxonomie.

Cependant, la taxonomie *espace-temps* présente quelques insuffisances pour représenter l'ensemble des collecticiels. Par exemple, les collecticiels d'édition coopérative présentant un mode d'interaction "variable" synchrone/asynchrone tels que l'éditeur présenté dans [Minör 93], peuvent figurer dans la classe des collecticiels synchrones et la classe des collecticiels asynchrones.

Pour résumer, les limitations de la taxonomie *espace-temps* nous paraissent multiples :

- la classification ne permet pas de placer distinctement tous les collecticiels dans une classe parmi les quatre classes identifiées,
- le critère espace ne paraît pas fondamental dans la classification des collecticiels,
- plus d'informations sur la nature et le fonctionnement sont généralement requises pour une classification.

Une autre classification très intéressante est fondée sur la nature du collecticiel. Plus précisément, il s'agit de grouper les collecticiels en tenant compte des fonctions qu'ils offrent. Selon cette approche, cinq classes de collecticiels sont identifiées.

Taxonomie fonctionnelle

Messageries électroniques

La messagerie électronique est un outil inspirée du courrier postal. La large diffusion de ce type d'application a fait du courrier électronique l'application de collecticiel la plus répandue.

La facilité d'utilisation apportée par le courrier électronique, engendre des problèmes de surcharge généralement provoqués par la fréquence élevée des réceptions des courriers. Plusieurs travaux se sont intéressés aux aspects de présentation et de structuration des courriers visant à alléger la tâche de l'utilisateur. Par exemple, dans [Mallone 87], le système permet de classer le courrier en plusieurs types définis par l'utilisateur. Il est ainsi possible de spécifier des traitements appropriés à chaque classe de courriers.

Caractéristiques :

mode d'interaction : asynchrone,

résolution de conflits : pas de conflits,

granularité : message,

participation : implicite,
communication : données.

Éditeurs de documents partagés

Ces applications permettent à des utilisateurs de partager un ensemble de documents, de les éditer, d'en modifier le contenu et de visualiser les opérations des autres intervenants.

Plusieurs collecticiels d'édition coopérative existent. Ils se distinguent essentiellement par le mode d'interaction qu'il fournissent. Certains collecticiels privilégient un travail découplé dans lequel chaque participant a la responsabilité d'un fragment de document. L'application Alliance (cf. [Decouchant 95]) figure parmi ces collecticiels.

Fig. 1.3 : Alliance

Grove est également un collecticiel pour l'édition coopérative développé par Ellis [Ellis 91]. Le mode d'interaction fournit par l'application est synchrone. Les

ensembles d'*items* manipulés dans un texte peuvent être privés, partagés ou publics. L'application n'offre aucun verrouillage sur les données et une granularité d'accès très fine. La résolution des conflits pouvant survenir pendant la coopération est à la charge des utilisateurs. D'après une étude citée dans [Ellis 91], les utilisateurs eux-mêmes évitent de créer des conflits en se plaçant sur des parties différentes du texte.

Caractéristiques :

mode d'interaction : synchrone, asynchrone, synchrone/asynchrone.

résolution des conflits : politiques à jetons, libre accès.

granularité : correspondant à des fragments (asynchrone), fine (synchrone).

participation : nécessaire pour le cas synchrone.

communication : données et contrôle.

Téléconférences assistées par ordinateurs

Les réunions de travail et les conférences assistées par ordinateur sont des collecticiels axés sur la communication synchrone. Ces applications ont la lourde tâche de remplacer les réunions autour d'une table, devant un tableau ou un rétro-projecteur.

La particularité des applications de téléconférence est qu'elles mettent à la disposition des utilisateurs des canaux de communication audio/vidéo (cf. [Johansen 84]). Cependant, elle se distingue par le support technologique utilisé pour l'acheminement du son et de la vidéo.

Les premières applications de téléconférence utilisaient des canaux de communication analogiques annexes au support informatique. Les systèmes tels que CAVECAT [Mantei 91] mettent en œuvre un équipement vidéo annexe au matériel informatique.

La seconde génération d'applications de téléconférence présente un environnement de travail complètement intégré. Souvent équipées de moyen de communication audio/vidéo numériques, des applications telles que MMConf [Crowley 90], MERMAID (*Multimedia Environment for Remote Multiple Attendee Interactive Decision-making*) [Ohmori 92], ShowMe (Sun), IVS [Turetti 94], fournissent des outils pour supporter le travail coopératif. Ce type d'application fournit un environnement coopératif (en anglais *Desktop Conferencing*). En général, il s'agit de fenêtres partagées servant de tableaux de présentation, d'éditeurs ou d'applications de dessin.

Caractéristiques :

mode d'interaction : synchrone, WYSIWIS strict.

résolution des conflits d'accès : politiques à jetons.

granularité : grande (application partagée).

participation : dépend de la nature des applications (peu importante pour les canaux de communication).

communication : données et contrôle.

Systèmes supports d'aide à la décision de groupe

Ces systèmes permettent d'assister un groupe d'utilisateurs pour des discussions et facilitent l'obtention d'un consensus, soit sur des sujets généraux, soit dans un domaine spécifique (création de document, génie logiciel [Johnson 93]).

CoNex (*Coordinator and Negotiation support for eXpert in design applications*) [Hahn 91] est un environnement expérimental qui offre un support coopératif utilisable dans le cadre de l'ingénierie du logiciel. L'application fournit un éditeur d'arguments pour assister les négociations entre les concepteurs et les analystes (système de conférence pour l'échange de messages informels). Elle fournit également un éditeur de contrats qui permet aux analystes d'attribuer les tâches à programmer et de responsabiliser les programmeurs.

D'autres applications sont issues du domaine de l'intelligence artificielle distribuée et des systèmes multi-agents. L'étude présentée dans [Durfee 89] donne un état de l'art en matière de solveurs de problèmes distribués coopératifs. Les grandes directions de recherche dégagées sont :

- *L'interprétation distribuée* : dans cette catégorie sont classés les réseaux de capteurs, les diagnostics de fautes pour des réseaux de communication, etc.
- *Le planning et le contrôle distribué* : notamment les applications de contrôle aérien distribué, les robots coopérants, les véhicules pilotés à distance, ...
- *Les systèmes experts coopératifs*.
- *La coopération humaine supportée par ordinateur*.
- *Les plates-formes pour appliquer et tester les modèles cognitifs*.

Les applications flot de travail, appelée *Workflow* en anglais, assistent et organisent un travail de groupe pour la création de documents au sein d'une entreprise. Le système permet d'organiser les interventions des acteurs et se

charge de véhiculer le document entre les différents points d'un circuit d'élaboration de documents.

Caractéristiques :

mode d'interaction : asynchrone.

résolution des conflits : mode découplé.

granularité : grande (le document).

participation : peu importante.

communication : données.

Environnements de développement partagé (EDP)

Ils s'agit d'environnement logiciels qui permettent à une équipe de développeurs de partager des ressources communes (fichiers sources, fichiers exécutables) organisées en versions en fonction de leur date de soumission. Ces environnements sont avec les applications *Workflow* les seuls exemples caractéristiques de collecticiels asynchrones.

Caractéristiques :

mode d'interaction : asynchrone.

résolution des conflits : mode découplé.

granularité : grande.

participation : implicite.

communication : données.

Bilan

La table de Fig. 1.4 illustre l'utilisation de la taxonomie fonctionnelle pour définir quelques propriétés affichées par chacune des classes de collecticiels répertoriées.

Cette étude est issue de notre analyse pragmatique des enjeux et des articles et documents qui décrivent les travaux existant et des travaux en cours dans le domaine du collecticiel.

	Messagerie électronique	Éditeurs partagés	Téléconférences	<i>flot de travail</i>	EDP
mode d'interaction	asynchrone	synchrone/asynchrone	synchrone	asynchrone	asynchrone
résolution des conflits	pas de conflits	Politiques à jetons/libre accès	Politiques à jetons/libre accès	tour de rôle	tour de rôle
granularité	grande	grande/faible	grande/faible	grande	grande
participation	implicite	implicite/complexe	complexe	implicite	implicite
communication	données	données/contrôle	données/contrôle	données	données

Fig. 1.4 : Table récapitulative des caractéristiques des différentes classes de collecticiels

1.5 Besoins des collecticiels

1.5.1 Partage, communication, coordination et interaction

Une activité coopérative fait intervenir un ensemble d'individus qui se sont fixés comme objectif la réalisation d'une tâche commune. Les résultats de cette tâche peuvent être persistants (e.g. édition d'un document) ou non (e.g. conversation). Dans le cas général, la réalisation d'une œuvre commune doit donner aux intervenants la possibilité :

- d'agir ensemble sur l'œuvre en question (e.g. éditer un document),
- de communiquer entre eux afin de commenter les actions faites sur l'œuvre,
- et finalement, de coordonner leurs actions.

Cette décomposition n'est qu'un raffinement du modèle dit "ontologique tripartite" proposé par Ellis [Ellis 94]. En fait, les deux espaces de production et de

coordination correspondent respectivement aux modèles ontologique et de coordination d'Ellis. Par contre, nous mettons en relief, dans notre décomposition, le besoin de fournir des moyens de communication Homme–Homme médiatisée, un aspect qui n'est pas explicite en tant que tel dans le modèle proposé par Ellis.

La décomposition fonctionnelle proposée ci-dessus nous permet d'identifier les besoins que doivent satisfaire une plate-forme pour la construction de collecticiels :

B1. Offrir un espace d'informations partagées entre les utilisateurs.

B2. Permettre des communications directes entre les utilisateurs.

A la différence des applications multi-utilisateurs classiques (e.g. bases de données) qui donnent à chaque utilisateur l'illusion qu'il est le seul à utiliser le système, les collecticiels doivent montrer pour chaque utilisateur l'existence des autres utilisateurs dans l'espace partagé. Ainsi, l'interface utilisateur d'un collecticiel doit visualiser des informations qui décrivent, par exemple, la liste des participants et leurs rôles effectifs et potentiels, leur place dans l'espace partagé et leurs actions en cours dans cet espace. Il faut donc :

B3. Rendre perceptible à chaque utilisateur les actions des autres utilisateurs dans l'espace partagé.

La perception des actions faites par les autres utilisateurs peut avoir lieu en temps réel ou en temps différé. Dans le premier cas, chaque fois qu'un utilisateur apporte un changement pertinent pour les autres, l'action est immédiatement propagée vers l'ensemble des utilisateurs ; on parle ici d'un mode de *coopération synchrone*. Dans le deuxième cas, les changements apportés par un utilisateur ne sont pas notifiés aux autres en temps réel [Gutwin 95]. La propagation des changements se fait ou bien à la demande explicite de l'utilisateur, ou bien, elle peut être déclenchée par l'application, d'une façon invisible à l'utilisateur, par exemple en fonction d'une horloge périodique ou en fonction du nombre d'actions accomplies. On dit dans ce cas que la coopération est *asynchrone*.

Une application de tableau blanc (e.g. tableau blanc de XTV [Abdel–Wahab 91]) est typiquement un exemple du mode de coopération synchrone. En revanche, la majorité des applications d'édition coopérative (e.g. Alliance

Decouchant 94]) fonctionnent selon le mode asynchrone. Dans le cas général, les deux modes de coopération doivent être permis par un collecticiel. Pour justifier ce besoin, prenons l'exemple de la rédaction coopérative d'un document. Une telle activité passe par au moins trois phases principales :

- discussion et élaboration du plan du document à rédiger, et attribution des rôles de chacun des participants (qui rédige quoi),
- rédaction proprement dite des différentes parties du document (chaque utilisateur rédige la partie qui lui a été affectée à l'issue de la première phase),
- et finalement, correction du document rédigé.

Si les trois phases sont à faire collectivement, il est plus opportun de fonctionner en mode synchrone pour la première et la troisième phase, tandis que le mode asynchrone convient mieux pour la phase de rédaction (la deuxième phase) [Minör 93], d'où le besoin de :

B4. Permettre les deux modes de coopération synchrone et asynchrone ainsi que la transition dynamique entre les deux modes.

D'un autre côté, il est évident que le bon déroulement d'une activité coopérative nécessite la synchronisation entre les différentes actions faites dans les différents espaces. Par exemple, dans une application d'édition coopérative, qui offre aussi un espace de communication, un auteur peut utiliser un canal audio/vidéo pour commenter ses contributions. Il est souhaitable, voire nécessaire, que les autres participants reçoivent les commentaires et les mises à jour du document d'une façon synchronisée. D'où le besoin de :

B5. Fournir des mécanismes pour synchroniser l'émission et la réception de différents flots de données susceptibles d'être engendrés, en parallèle, par différentes applications partagées.

1.5.2 Problèmes de mise en œuvre

Le travail individuel et le travail coopératif constituent deux modes de travail complémentaires. Afin de faciliter la transition entre ces deux modes, un collecticiel doit permettre l'utilisation des mêmes outils pour le travail individuel et pour les tâches coopératives. Cela se traduit par le besoin de :

B6. Rendre partagées des applications mono-utilisateurs existantes.

Deux avantages immédiats découlent de cette conception du collecticiel :

- une facilité d'utilisation qui évite à l'utilisateur une période d'apprentissage supplémentaire dans le cas de l'introduction de nouveaux outils,
- un faible coût de mise en œuvre dû à la réutilisation de modules logiciels existants.

Or l'ensemble des utilisateurs qui coopèrent ne dispose pas forcément du même environnement de travail. Les différents utilisateurs utilisent des stations de travail différentes (e.g. Sun, PC ou Macintosh) et souvent des systèmes d'exploitation différents (e.g. Unix, DOS, MacOS). En conséquence, il est souhaitable que le collecticiel satisfait le besoin suivant :

B7. Permettre l'hétérogénéité des environnements de travail en termes de systèmes d'exploitation et de plate-formes matérielles.

Répondre à ce besoin exige que l'implantation des services pour la coopération soit indépendante des caractéristiques de la plate-forme d'exécution sous-jacente. Ce besoin peut être résolu par l'utilisation des gestionnaires d'objets tels que CORBA ou d'outils logiciels, tels que ILU (*Interconnection Language Unificator*) [Jansen 95], qui permettent à des modules écrits dans des langages de programmation différents de communiquer d'une manière complètement transparente.

D'un autre côté, les différents utilisateurs peuvent utiliser des applications différentes pour mener la même tâche. Par exemple, pour éditer un document, un utilisateur peut se servir de MS-Word, de FrameMaker ou de WordPerfect. Des protocoles inter-applications sont nécessaires afin de permettre l'emploi d'applications hétérogènes dans la même session de travail. Il faut donc :

B8. Permettre l'emploi d'applications différentes pour accomplir la même tâche coopérative.

1.5.3 Configuration à la carte

Le processus de travail en groupe est un processus dynamique par nature. Les besoins des utilisateurs, en termes de politiques de contrôle, changent en fonction de plusieurs facteurs (e.g. le nombre de participants, le but de la session, l'état d'avancement du travail). Par exemple, dans une session de téléconférence faisant intervenir deux ou trois personnes, il est commode d'appliquer une politique de prise de parole basée sur un protocole dit social (par analogie avec une conversation téléphonique). Dans un tel protocole, les échanges sont fondés sur un consensus admis par les utilisateurs. Si le nombre de participants devient plus important, il est plus judicieux de passer à une politique système telles que les politiques de tour de rôle pour éviter que plusieurs personnes ne parlent en même temps.

Le collecticiel doit permettre la réalisation de scénarii coopératifs variés (e.g. une séance de télé-enseignement avec un enseignant et plusieurs élèves, une réunion de travail ordinaire, un vote collectif). Il faut donc :

B9. Offrir une bibliothèque de politiques de contrôle et donner aux utilisateurs la possibilité de choisir la politique à appliquer en fonction de la tâche qui leur incombe.

Au niveau système, le changement dynamique nécessite de fournir des mécanismes qui assurent des transitions correctes entre les différentes politiques : pouvoir terminer l'exécution d'un protocole de contrôle tout en garantissant la validité de toutes les conditions initiales exigées par le nouveau protocole utilisé. Il en découle immédiatement le besoin de :

B10. Permettre le changement dynamique des politiques de contrôle.

1.5.4 Configuration dynamique du collecticiel

Les utilisateurs impliqués dans un travail coopératif ne sont pas forcément tous disponibles pour participer à la session en même temps. Il est contraignant d'exiger la présence de tous les acteurs de la coopération au moment de l'initialisation de la coopération. Il est nécessaire de tolérer l'entrée tardive ainsi que la déconnexion dynamique des participants, (e.g. 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 causer l'interruption, voire la fin de la séance). Il faut donc :

B11. Permettre la connexion et la déconnexion dynamiques de participants.

La connexion dynamique d'un participant doit restituer sur le site du nouvel arrivant le contexte courant de l'espace partagé sans perturber le déroulement de la session.

La déconnexion d'un participant peut se faire suite à l'occurrence d'une panne de site ou du réseau de communication, ou suite à une initiative délibérée. Dans tous les cas de figure, la déconnexion ne doit pas entraîner la terminaison de la session ni la perturbation de son évolution. Il faut donc :

B12. Fournir des mécanismes qui permettent de continuer la coopération même en cas de panne.

1.6 Conclusion

Dans ce chapitre nous avons présenté les caractéristiques des collecticiels ainsi que les fonctions qu'ils doivent fournir. Cette étude contribue à l'élaboration du cahier des charges de la plate-forme *CoopScan*.

Les collecticiels que nous souhaitons développer dans *CoopScan* sont ceux construits à partir de modules logiciels existants (approche fondée sur la réutilisation de code et sur la réutilisation de programmes sources). Ces collecticiels permettent aux utilisateurs de coopérer en utilisant des applications initialement mono-usager (B6) auxquelles nous rajoutons un module de contrôle commun. Ce module de contrôle répond aux besoins définies dans (B4, B5, B9, B11).

D'autre part, l'adoption de cette approche permet de répondre aux besoins (B1) et (B2) en utilisant les applications appropriées (e.g. un éditeur coopératif utilisé conjointement avec un outil de communication audio).

Le problème d'hétérogénéité des environnements de travail (B7) n'est pas approfondi dans ce rapport. Toutefois, le coût de mise en œuvre des fonctions qui permettent de répondre à ce besoin est pris en compte dans l'évaluation des architectures des collecticiels.

L'utilisation d'applications différentes (B8) pour réaliser une même tâche coopérative (e.g. édition coopérative où chaque participant utilise une application distincte des autres) ne figure pas parmi nos priorités. Quelques solutions

pponctuelles à ce problème existent dans la littérature. Elles apportent des solutions spécifiques à des applications particulières, GroupDesign [Karsenty 93] en est un exemple. En fin, la synchronisation entre plusieurs flots de données (B5) constitue un sujet d'étude à part entière; il n'est pas approfondi dans ce rapport.

Chapitre I

Collecticiels : définition, classification et étude des besoins

I.1 Introduction	11
I.2 Terminologie	12
I.3 Paramètres des collecticiels	13
I.4 Classifications des collecticiels	21
I.5 Besoins des collecticiels	27
I.5.1 Partage, communication, coordination et interaction	27
I.5.2 Problèmes de mise en œuvre	29
I.5.3 Configuration à la carte	31
I.5.4 Configuration dynamique du collecticiel	31
I.6 Conclusion	32

Chapitre II

Architectures logicielles pour collecticiels

II.1 Introduction

Nous nous intéressons dans ce chapitre à l'étude des architectures logicielles pour la construction de collecticiels synchrones. L'objectif de cette étude est double, elle vise d'une part la définition d'un modèle d'architecture qui facilite la construction d'applications coopératives et, d'autre part, l'exploitation de ce modèle pour configurer ces applications en fonction de leur environnement d'exécution.

Des études menées dans les différentes disciplines telles que la science de la communication [MacCarthy 94], de l'éthique informatique [Moor 85], ou de l'ethnographie, nous ont permis d'identifier et d'analyser quelques besoins des collecticiels (cf. chapitre I) en termes de fonctions devant être fournies par l'outil informatique à l'utilisateur humain. Il s'agit maintenant de donner les spécifications du système à mettre en œuvre. Le choix de l'architecture y apparaît comme un élément de base tant sa définition et sa mise en œuvre influent sur le coût de développement et sur les performances. Notre principal objectif est de faciliter la construction de collecticiels en fonction d'une part des contraintes d'utilisation (être capable de spécialiser le collecticiel en fonction des besoins des utilisateurs) et, d'autre part, de l'environnement sous-jacent (i.e. plate-forme d'exécution système, réseau de communication, modalités d'utilisation de l'application, ...).

Par ailleurs, face à la complexité croissante des systèmes et à la diversité des techniques logicielles proposées, l'architecture demeure le critère de classification et d'évaluation le plus intéressant pour un concepteur de systèmes informatiques dédiés au travail coopératif. L'architecture d'un collecticiel permet de définir les modules logiciels qui constituent l'application, leur répartition physique (localisation sur des sites informatiques) et la façon dont ces modules

interagissent. La communication mise en œuvre par les modules de l'application regroupe aussi bien les interactions entre modules que les interactions du type modules logiciels–utilisateurs pour établir une communication homme–homme médiatisée.

Les approches communément adoptées pour la construction de collecticiels sont soit le développement complet de l'application coopérative, soit la réutilisation d'outils et d'applications existantes généralement mono–utilisateur, qui sont assemblés et intégrés au sein du collecticiel. La première approche est généralement adoptée pour construire des collecticiels qui demandent des fonctions spécifiques que les applications existantes ne fournissent pas nécessairement, tandis que la seconde approche permet d'étendre au travail coopératif des environnements mono–usager. Nous nous intéressons dans ce travail uniquement à l'approche fondée sur la réutilisation de modules logiciels existants, l'approche de construction complète d'une application spécifique ne faisant pas partie de nos objectifs.

II.1.1 Organisation du chapitre

Nous étudions dans ce chapitre différentes architectures logicielles utilisées pour mettre en œuvre un collecticiel. La section II.2 présente les deux principales approches adoptées pour la construction d'applications coopératives : la construction complète d'applications ou la ré–utilisation par l'intégration d'applications existantes, cette dernière approche constitue le sujet du travail que nous présentons. Nous détaillons ensuite les approches fondées respectivement sur la réutilisation d'applications dites *ouvertes*, et l'intégration d'applications au niveau du système de fenêtres.

Dans la section II.3, nous présentons le modèle d'architecture selon lequel est construite la plate–forme *CoopScan*. Nous y définissons l'ensemble des modules chargés de rendre coopératives des applications qui, *a priori*, ne le sont pas.

Dans cette section nous définissons également les concepts fondamentaux sur lesquels se fonde notre étude ; notamment, la généricité et la ré–utilisation d'applications. Nous y précisons également les objectifs poursuivis et les choix de conception que nous retenons pour *CoopScan*.

II.2 Les approches architecturales pour la construction de collecticiels

Le développement par intégration consiste à transformer une application initialement mono-utilisateur en une application coopérative qui puisse être simultanément partagée par plusieurs utilisateurs. Ceci ne doit s'accompagner d'aucune modification du code source de l'application. Ce code n'est d'ailleurs généralement pas disponible, ce qui nécessite donc l'encapsulation du programme objet afin de pouvoir le réutiliser. Ce type de développement ne peut toutefois être appliqué qu'aux applications qui offrent une ouverture via une interface (e.g. API), même minimale, vers leur environnement d'exécution.

Nous distinguons deux sortes d'ouverture d'une application : celle permise par le système de fenêtres, et celle prévue lors de la conception initiale de l'application. Les méthodes de construction de collecticiels par intégration dépendent de ce type d'ouverture et se scindent en deux classes : intégration d'applications *ouvertes* à travers des mécanismes appropriés et intégration au niveau du système de fenêtres.

II.2.1 L'intégration d'applications ouvertes

Une application ouverte est une application mono-utilisateur qui fournit une interface sous forme d'une API (*Application Programming Interface*) vers son environnement et d'un système de rétro-action ou service de notification (*Callback*). L'API est une interface de contrôle qui permet à des modules logiciels extérieurs à l'application d'accéder aux ressources de celle-ci (e.g. fenêtres, fichiers, texte, ...), et de reproduire les actions que l'on peut faire à partir de l'interface utilisateur. Le service de notification permet d'intercepter les tentatives d'actions accomplies à travers l'interface utilisateur afin de les prendre en compte pour réaliser des traitements ajoutés. Un tel service permet à des modules extérieurs à l'application de s'abonner à des événements de l'application afin de contrôler son exécution.

L'intégration d'une application ouverte consiste à définir un module logiciel qui intercepte les actions des utilisateurs au moyen du service de notification pour les reproduire à travers l'API chez d'autres utilisateurs. Pour que cette approche soit applicable, il est indispensable que l'API offre une expressivité équivalente à celle de l'interface utilisateur.

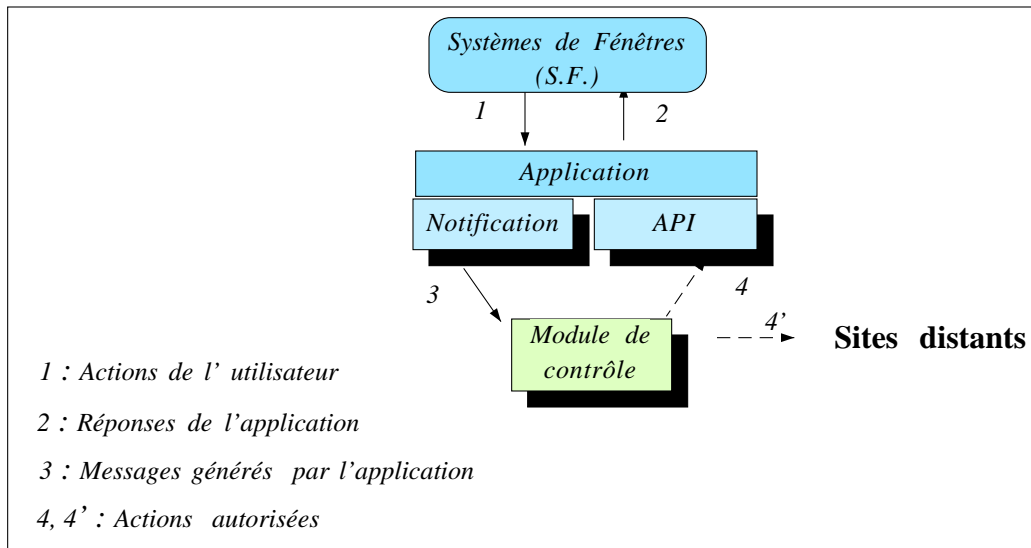


Fig. 2.1 : Schéma d'intégration pour une application ouverte dans une architecture client-serveur utilisant un serveur de fenêtres.

L'interprétation des événements dans le contexte de l'application facilite la mise en œuvre des opérations de filtrage des actions sur l'application. Ceci rend possible la mise en œuvre des protocoles de contrôle pour la protection des données de grain varié qui dépendent essentiellement du type de données manipulées par l'application et non plus des ressources du système de fenêtres (SF). De plus, il est plus facile de mettre en œuvre des mécanismes de journalisation sélective des événements, ce qui permet d'enregistrer des sessions de travail afin de pouvoir les rejouer. De même, les protocoles de connexion et déconnexion dynamique de participants à une session du collecticiel qui s'appuient sur de tels mécanismes [Chung 93] peuvent alors être inclus dans la plate-forme.

II.2.1.1 Etude de cas pour l'intégration d'une application ouverte : l'éditeur de documents Thot [Quint 94]

L'éditeur offre deux mécanismes de contrôle lui permettant d'interagir avec son environnement d'exécution : une API pour agir sur les documents en cours de traitement, et un mécanisme d'appels externes appelé ECF (*External Call Facility*).

Le mécanisme ECF

Le mécanisme ECF permet de développer sur la base de Thot des applications fondées sur le concept de document actif. Un document actif est un document qui

se transforme lui-même ou agit sur l'environnement dans lequel il est traité. Cet aspect constitue l'une des caractéristiques des applications ouvertes.

Le mécanisme ECF permet d'étendre les fonctions de l'éditeur par des procédures de traitement particulières, appelées *actions*, qui sont exécutées quand certaines commandes d'édition sont lancées par l'utilisateur. L'ECF est fondé sur la structure logique des documents traités par Thot, appelée schéma de structure. Vue par l'éditeur, une application est un ensemble de procédures (les traitements) qui doivent être exécutées dans certaines conditions explicitées dans un schéma d'interface écrit dans un langage spécifique.

Un message est envoyé par l'éditeur lorsqu'une commande d'édition est effectuée par l'utilisateur. Par exemple, un message est envoyé quand l'utilisateur sélectionne un élément, quand il modifie une chaîne de caractères, quand il crée ou détruit un élément logique (e.g. une section, un paragraphe), quand il enregistre le document, etc. Chaque message généré est associé à un objet concerné par la commande d'édition effectuée.

Pour préserver une terminologie cohérente dans tout le chapitre, nous appellerons *événements* les messages générés par l'éditeur.

Remarque

Par opposition aux commandes d'édition de l'utilisateur, les opérations effectuées sur les documents par les programmes d'application ne donnent pas lieu à des messages, bien que l'utilisateur et les programmes aient accès aux mêmes fonctions de l'éditeur, mais à travers des interfaces différentes : OSF/Motif pour l'utilisateur, API pour les programmes.

Pour les commandes d'édition qui génèrent des événements, le mécanisme ECF génère deux événements qui portent le même nom mais qui se distinguent par un suffixe : un événement précédant l'exécution de la commande et un événement qui suit son exécution.

- Suffixe `.Pre` : événement envoyé au moment même où la commande est appelée par l'utilisateur, avant que l'éditeur l'ait traité. Cet événement permet à l'application d'effectuer un traitement avant que l'éditeur fasse son traitement normal, ou à la place du traitement normal de l'éditeur. Un booléen est retourné par l'action en fin de traitement pour indiquer si l'action exécutée remplace ou complète le traitement de l'éditeur. S'il y a un remplacement (retour `True`), l'éditeur n'exécute pas le traitement prévu pour la commande.

- Suffixe `.Post` : événement envoyé après que l'éditeur a traité la commande.

Les événements produits pas l'ECF sont uniquement envoyés aux objets qui les ont demandés et qui sont concernés par la commande.

Les événements envoyés par l'ECF sont accompagnés d'un contexte qui permet d'identifier dans quelles conditions s'est produit le message. Ce contexte varie selon les événements, parce qu'il ne donne que les informations pertinentes pour l'événement considéré. Les informations contenues dans le contexte représentent:

- la commande qui a provoqué l'événement,
- le document concerné,
- l'élément concerné,
- le type de l'élément concerné,
- l'élément cible sur lequel porte l'opération d'édition, qui peut être différent de l'élément concerné,
- le type de la règle de présentation concernée,
- l'attribut concerné,
- le type de l'attribut concerné.

Le mécanisme API

L'éditeur Thot offre une boîte à outils (l'API) constituée d'un ensemble de fonctions d'édition conçues pour développer des applications qui manipulent les documents structurés avec un système de fenêtres X-*Windows* sous Unix. Elle contient un ensemble de bibliothèques qui peuvent être reliées à tout autre programme. L'API permet à d'autres modules d'effectuer le même type d'opérations sur les documents que ceux proposées à l'utilisateur de l'éditeur (e.g. création de documents). Vu par ces modules, l'éditeur se présente à la fois comme un ensemble de fonctions d'édition qui peuvent être appelées à travers l'API, et comme un émetteur de messages.

Mise en œuvre de la coopération

Pour doter l'éditeur Thot d'un comportement coopératif, notre démarche consiste à intercepter toutes les commandes d'édition effectuées sur des documents partagés et de juger de leur exécution. En général, deux cas de figure sont envisagés, interdire l'exécution de la commande, ou l'autoriser et appeler les

fonctions appropriées qui se chargent de la reproduire sur les autres sites du collectif.

L'intégration de Thot dans un environnement coopératif se fait en deux étapes :

- L'Abonnement : il s'agit de spécifier au niveau d'un fichier de description d'interface les événements de l'éditeur auxquels nous nous intéressons. Cette phase consiste à associer des procédures de traitement spécifiques à certains événements générés par l'éditeur. Ces procédures sont définies dans un module extérieur dit de *contrôle* qui implante des fonctions coopératives.
- Le Traitement : il s'agit de construire les procédures de traitement qui seront systématiquement exécutées à la réception des événements.

Le schéma d'interface qui permet d'associer événements et procédures de traitement est toujours relié à un schéma de structure du document définissant la structure logique du document.

Un schéma d'interface commence par le mot-clé APPLICATION suivi du nom du schéma auquel il est relié.

Après l'instruction APPLICATION, au moins une des trois sections introduite par les mots clés suivants doit être présente: DEFAULT, ELEMENTS, ATTRIBUTES. Le schéma doit se terminer par le mot-clé END.

```
SchemaI = 'APPLICATION' IdentElem ';'
          ['DEFAULT' DefActions]
          ['ELEMENTS' <ElemActions>]
          ['ATTRIBUTES' <AttrActions>]
          'END' .
```

Les éléments et les attributs constituent les deux entités de base d'un document Thot. Les commandes d'édition sont menées sur des éléments ou sur des attributs d'un document.

Les éléments définissent la structure logique d'un document (par exemple, des sections, des paragraphes, ...). Les attributs sont des informations attachées à des types pour en préciser la fonction dans le document. Un attribut ajoute une information d'ordre sémantique. Il existe plusieurs types d'attributs : énuméré, entier, texte, et référence. Les attributs références permettent de désigner des éléments (par exemple, pour signifier que tel objet graphique doit être aligné horizontalement avec tel autre).

La section `DEFAULT` de *schémaI* permet de définir les traitements communs aux attributs et aux éléments. Les sections `ELEMENTS` et `ATTRIBUTES` définissent les actions attachées uniquement à l'une ou l'autre de ses entités.

La Fig. 2.2 illustre le schéma d'interface utilisé pour intégrer Thot dans la plate-forme *CoopScan* dont nous donnerons la spécification et l'architecture plus loin. Toutefois, à ce niveau de l'étude, *CoopScan* est définie comme un ensemble de modules logiciels qui mettent en œuvre la coopération.

```
APPLICATION EDITOR ;

DEFAULT
BEGIN

    StdDocOpen.Pre           : CheckFloor;
    StdDocOpen.Post          : DocOpen;

    StdElemSelect.Pre        : CheckFloor;
    StdElemSelect.Post       : ProcessSelection;

    StdElemExtendSelect.Pre  : CheckFloor;
    StdElemExtendSelect.Post : ProcessESelection;

    StdElemTextModify.Pre    : PreModElem;
    StdElemTextModify.Post   : ModifyElem;

    StdElemDelete.Pre        : DeletElem;

    StdElemActivate.Pre      : ChekFloor;
    StdElemActivate.Post     : ProcessESelection;

    ...
END;
END.
```

Fig. 2.2 : Schéma d'interface définissant l'association d'événements Thot à des fonctions de coopération

Les fonctions dont le nom apparaît dans la partie de droite des règles d'association du schéma d'interface `EDITOR` sont définies dans un module extérieur à l'éditeur.

Par exemple, La fonction *CheckFloor* vérifie les droits d'accès associés à l'utilisateur ayant activé une commande d'édition sur un document. La fonction

retourne une valeur qui indique si l'utilisateur est autorisé ou non à mener des actions sur le document en question.

L'information permettant d'identifier le document ainsi que les autres informations définissant le contexte de la commande sont, en général, récupérées dans deux structures de données. Ces structures définissent deux entités :

- l'entité élément :

```
typedef struct {
    ECFevent event;
    Document document;
    Element element;
    ElementType elementType;
    int position;
} NotifyElement
```

- l'entité attribut :

```
typedef struct {
    ECFevent event;
    Document document;
    Element element;
    Attribute attribute;
    AttributeType attributeType;
} NotifyAttribute;
```

Principe de fonctionnement

L'objectif visé par l'intégration de Thot dans un environnement coopératif est la réalisation d'une tâche d'édition synchrone dans laquelle plusieurs participants interagissent en temps-réel en partageant des documents Thot.

Les utilisateurs de l'application coopérative doivent avoir la même vue des documents partagés ainsi que des modifications qui y sont apportées.

Pour aboutir à un tel fonctionnement, il est donc indispensable de pouvoir contrôler l'exécution de l'éditeur sur chaque site moyennant les mécanismes ECF et API.

Chaque commande d'édition activée par un utilisateur est attachée à une action de traitement. Pour la plupart des actions, l'événement suffixé par `.Pre` est attaché à la fonction de contrôle d'accès `CheckFloor`. Il s'agit d'une fonction booléenne qui, en consultant des informations relatives au déroulement de la coopération, vérifie si l'utilisateur a le droit d'exécuter telle commande sur tel document (voire tel élément ou attribut de tel document). Pour statuer sur les droits attribués à l'utilisateur en question, `CheckFloor` exploite les informations

concernant le contexte d'exécution de la commande, les données du contexte étant retournées dans la structure *NotifyElement* (cf. Fig. 2.3).

La valeur retournée par la fonction *CheckFloor*, lorsqu'elle autorise l'utilisateur d'accomplir des opérations, permet aussi à l'éditeur de traiter la commande.

A la fin du traitement de la commande, l'ECF retourne un second événement suffixé par *.Post*. En général, cet événement renferme le complément d'information sur la commande annoncée par l'événement *.Pre*.

Si la commande est autorisée, la procédure de traitement attachée au *.Post* se charge de la communiquer aux autres sites participants. Pour chacun de ces sites, un module de contrôle se charge de récupérer la commande, et de la transmettre à l'éditeur via le mécanisme API.

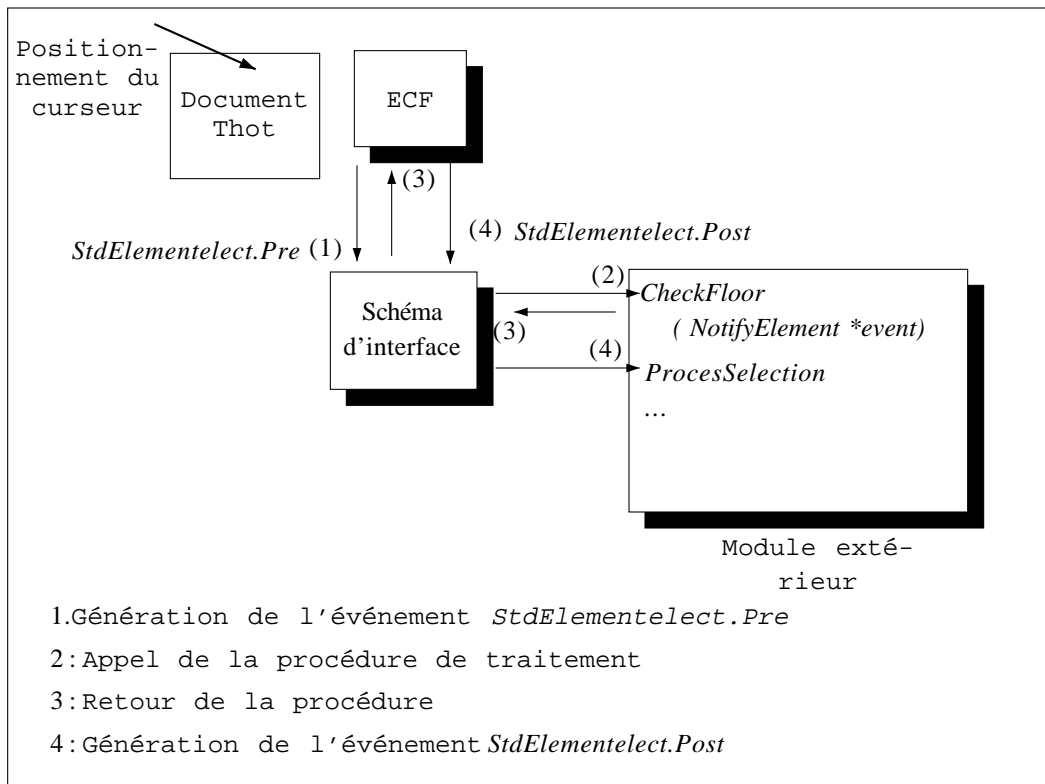


Fig. 2.3 : Les étapes de traitement d'une commande de positionnement du curseur dans un document Thot

La Fig. 2.3 illustre les étapes franchies par le flot d'exécution qui permet de traiter la commande de sélection d'un élément dans le document. A l'étape (1), l'ECF génère l'événement *StdElemSelect.Pre*. Conformément au schéma

d'interface, l'émission de cet événement déclenche l'exécution de *CheckFloor* (étape 2). Le retour `False` de *CheckFloor* permet à l'éditeur de traiter normalement la commande (étape 3). Suite à l'exécution de la commande, l'ECF génère l'événement *StdElementelect.Post*. Cet événement contient toutes les informations nécessaires pour reproduire la commande. La fonction *ProcessesSelection* se charge de récupérer les informations de la structure `NotifyElement` dans un message de contrôle. Le message est ensuite transmis vers tous les sites par l'intermédiaire du module de communication fourni par la plate-forme `CoopScan`.

Limitations affichées par la version actuelle de Thot.

Les mécanismes API et ECF s'avèrent très utiles dans la mise œuvre de la coopération. Cependant, nous avons pu dégager, quelques limitations de ce mécanisme qui concernent notamment l'ECF. Les plus importantes sont regroupées dans la liste suivante :

- l'ECF ne génère pas d'événements quand des opérations sur le document sont accomplies à partir de l'API,
- Certaines commandes d'édition ne sont pas à l'origine d'événements `.Pre`.

II.2.1.2 Conclusion

L'intégration d'une application ouverte présente l'avantage de manipuler des événements produits par l'application. Ce type d'événements est susceptible de délivrer une information synthétique généralement contenue dans un nombre bien supérieur d'événements produit par un système de fenêtres (SF) pour désigner la même action utilisateur. Les avantages de cette approche sont :

- un trafic réseau réduit dû à un faible nombre d'événements générés par l'application,
- une journalisation des événements aisée, pouvant être exploitée par d'autres services du contrôleur (e.g. rejouer une session de travail pour un retardataire),
- une bonne interopérabilité entre SF hétérogènes puisque les événements sont liés à l'application, et, surtout,
- l'interprétation des événements dans le contexte de l'application ; par exemple, il est possible, dans le cas d'un éditeur de documents ouvert, de savoir que le troisième paragraphe de la section 3.2 du document est sélectionné et non pas une zone z donnée de la fenêtre f .

II.2.2 Les approches d'intégration au niveau d'un système de fenêtres : La plate-forme X-Windows

Il s'agit d'une approche de construction d'applications coopératives par intégration au niveau *bas*. Elle est adoptée dans le cas des applications dont la gestion de l'interface utilisateur est prise en compte par un SF (Système de Fenêtres) (e.g. X-Windows, SunWindows). Par exemple, X-Windows alloue à chaque application un ensemble de ressources (e.g. fenêtres, couleurs) à travers lesquelles il gère ses entrées-sorties. La mise en œuvre de cette approche consiste essentiellement à intercaler entre le système de fenêtres et les applications intégrées un module logiciel communément appelé pseudo-serveur dont les fonctions principales sont le partage et la communication [Abdel-Wahab 91] [Crowley 90].

Plusieurs applications coopératives sont fondées sur des approches d'intégration exploitant des informations du niveau du système de fenêtres et plus précisément des informations récupérées au niveau du protocole X dans le cas d'un environnement X-Windows. La plupart de ces applications sont construites conformément à l'approche SharedX [Garfinkel 89] dans laquelle le client X qui représente le noyau fonctionnel de l'application envoie des requêtes d'affichage définies par le protocole X au serveur d'interaction. Le serveur collecte, analyse puis communique au client les actions de l'utilisateur. Des produits industriels tels que SharedX ou ShowMe [ShowMe] de Sun sont des exemples d'applications reposant sur des mécanismes de multiplexage du protocole X. Parmi les projets de recherche les plus conséquents, citons Mermaid [Ohmori 92] et MMConf [Crowley 90].

Ces applications sont augmentées de canaux de communication audio/vidéo annexe afin d'améliorer leur utilisation dans un environnement complètement distribué.

D'autres applications et prototypes développés sur la plate-forme Unix/X-Windows sont donnés dans la liste suivante :

- **Xshare** [Gabre 96] est l'outil permettant le partage d'applications s'exécutant dans un environnement X-Windows. L'outil permet des échanges de messages textuels entre les différents participants, le partage d'un "tableau blanc" et la mise en place de communication audio/vidéo entre les participants. L'application est mise en œuvre selon une architecture centralisée offrant un module de contrôle qui gère l'accès des utilisateurs

- **Share project** [Glicksman 93] fournit un environnement pour le travail coopératif qui permet de réaliser des tâches synchrones ou asynchrones. Il renferme un outil asynchrone basé sur l'application *email*, et des outils synchrones basés sur *Xshare* permettant d'utiliser les applications X-Windows en mode coopératif.

En conclusion, l'étude de l'environnement X-Windows nous permet de regrouper les approches de construction d'applications coopératives en deux classes fondées respectivement sur l'extension des clients et l'extension du serveur. Ces deux approches sont explicitées dans la suite.

II.2.2.1 L'extension d'applications clientes

L'approche "extension d'applications" est une instanciation particulière de l'approche de construction complète de l'application dans un environnement client-serveur. Il s'agit d'augmenter le noyau fonctionnel de l'application (i.e. le client) par des fonctions de coopération. Ceci passe inévitablement par une modification du code source de l'application initiale. Les partisans de cette approche sont souvent les constructeurs même de l'application initiale.

L'intérêt de cette approche réside dans le fait que les seules modifications sont faites sur le client. Cet intérêt se transforme en gros handicap si le code du client ne nous est pas accessible. Néanmoins, les avantages et les inconvénients de cette approche ne sont pas connus avec précision, aussi nous ne répertorions aucune application construite selon ce modèle.

II.2.2.2 L'approche pseudo-serveur

Dans un environnement classique client-serveur, un serveur peut être étendu afin de pouvoir communiquer avec d'autres serveurs. Dans ce cas, le serveur joue le rôle d'intermédiaire il est chargé de transmettre les informations échangées entre les clients et leurs serveurs locaux comme s'il s'agissait d'une fonction supplémentaire du serveur local.

Selon cette approche, il est possible d'intercepter les informations échangées entre l'application et le SF, et de les échanger entre différents sites utilisateurs. Ces informations se présentent en général sous la forme d'événements produits par le SF suite aux actions d'un utilisateur sur l'interface de l'application, de requêtes émises de l'application vers le SF pour afficher ses sorties, et de réponses émises par le SF vers l'application afin de l'informer de l'état des ressources qui lui sont allouées.

Plusieurs collecticiels sont construits selon cette approche. Par exemple, dans XTV, les auteurs réutilisent des applications initialement non coopératives qui s'exécutent avec le système de fenêtres X–Windows [Scheifler 86] (e.g l'éditeur *xedit*, la palette de dessin *xpaint*, ...) pour en faire des applications partagées. La mise en œuvre de cette approche dans XTV, dont l'architecture est illustrée dans Fig. 2.4, consiste à intercaler un module de contrôle, jouant le rôle de serveur virtuel (pseudo–serveur), entre l'application et le serveur X.

Au moment de son lancement, l'application s'abonne au module de contrôle au même titre qu'elle s'abonne au serveur X. Le module de contrôle, préalablement connecté au serveur X, se charge de faire l'intermédiaire entre l'application et le serveur. Il intercepte les événements générés par le serveur X, les traite dans un module de communication PTP (*Packet Translator Process*) qui identifie leur provenance (e.g. identification du serveur X, identification de la ressource X) et les communique à un module de contrôle PSP (*Packet Switch Process*) localisé sur le même site que l'application. Suivant la même logique de fonctionnement, les requêtes émises par l'application sont récupérées par le PSP qui se charge de les diffuser en direction des autres sites partageant l'application afin d'y reproduire les actions d'un utilisateur, de manière synchrone (cf. Fig. 2.4). Dans le cas général, les informations échangées entre les sites dépendent de l'architecture de mise en œuvre implantée. Le module de contrôle est détaillé dans la sous–section II.2.3.

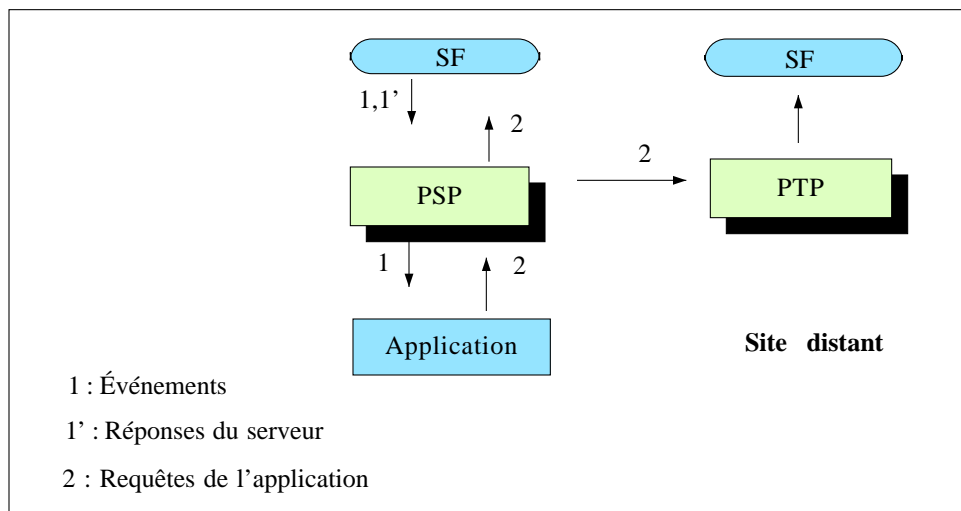


Fig. 2.4 : Schéma d'intégration au niveau du système de fenêtres X–Windows

L'intégration au niveau bas présente les avantages suivants.

- Un grand choix d'applications à intégrer : cette approche s'applique à toutes les applications s'exécutant sous le même SF.
- Un module de contrôle générique : les fonctions de base de ce module sont effectuées sur les données échangées entre le SF et les applications. Ces données sont codées d'une manière identique pour toutes les applications. Par exemple, dans le cas du système X-Windows, toutes les ressources allouées aux applications (e.g. fenêtres, fontes, contextes graphiques) sont gérées par le serveur X indépendamment de la nature de l'application ; les événements produits par le serveur X suite aux changements d'état des ressources ont un format identique. Par conséquent, les traitements réalisés par le module de contrôle sur les informations fournies par le SF (événements X) peuvent être communs à toutes les applications.
- Un faible coût de développement : dans le cas où le collecticiel reproduit chaque action d'un utilisateur sur tous les sites indépendamment de la nature de l'action et de la nature de l'application, le module de contrôle est réutilisable sans aucun changement pour toutes sortes d'applications. Dans le cas où des traitements particuliers sont souhaités, en fonction de la nature de l'application, ou de la nature des actions accomplies sur son interface (e.g. ne pas reproduire tous les mouvements du pointeur souris), le module de contrôle peut être modifié en conséquence de manière systématique et peu coûteuse en adoptant une démarche de construction modulaire.

À ces avantages s'opposent différents inconvénients.

- Ce type d'intégration demeure étroitement lié au système de fenêtres utilisé. L'utilisation de deux systèmes de fenêtres différents rendrait impossible la mise en œuvre de cette solution sans prévoir des mécanismes d'interopérabilité non standards.
- Un trafic réseau important qui peut s'avérer contraignant lors de l'utilisation d'un protocole de diffusion vérifiant certaines propriétés (e.g. fiabilité, atomicité).
- La mise en correspondance des ressources allouées par le système de fenêtres aux applications partagées sur chaque site.

Par ailleurs, l'intégration au niveau du SF n'offre pas la souplesse requise dans les collecticiels pour la protection des données. En effet, les informations véhiculées entre les différents modules qui constituent le collecticiel sont ici des

événements liés à la gestion de l'interface (e.g. clic souris, déplacement souris). Ces événements ne possèdent pas une sémantique suffisante pour permettre d'identifier facilement les données manipulées par l'application. La mise en œuvre de mécanismes de contrôle d'accès offrant un faible grain de partage est souvent prohibitive de par le nombre d'événements reçus (e.g. allocation de différents droits d'accès sur plusieurs zones d'une même fenêtre). C'est pourquoi les protocoles d'accès les plus utilisés sont des politiques à gros grain telles que les politiques dites à *jeton* dans lesquelles le droit d'agir sur les données de l'application est attribué par l'intermédiaire d'une information particulière échangée entre les utilisateurs (e.g. un jeton pour accéder un fichier). Dans XTV, un *jeton* est attribué pour chaque application partagée. L'utilisateur qui détient le jeton est le seul à pouvoir agir sur l'interface de l'application.

D'une manière générale, cette approche permet de rendre "coopératives" des applications pour lesquelles on ne souhaite pas mettre en place des politiques de contrôle d'accès permettant un faible grain de partage des données (e.g. paragraphes, sous paragraphes) [Ellis 89] [Karsenty 93], ou dans lesquelles le contrôle est assuré par les utilisateurs eux-mêmes [Ellis 91] selon des conventions sociales (par analogie avec une communication téléphonique où deux personnes ne parlent pas simultanément) [Trevor 94] [Stefik 87]. La communication directe entre les utilisateurs est alors nécessaire et elle est généralement assurée grâce à des outils annexes à l'application (canal audio ou télépointeur). Ces politiques sont particulièrement appropriées dans les applications où le risque de conflit d'accès est minime telles que les palettes de dessin partagées [Shu 94] où les données partagées sont des *pixels*.

II.2.2.3 Intégration d'applications dans l'environnement Macintosh

Intérêt de l'étude

Nous étudions dans cette sous-section les moyens fournis par l'environnement Macintosh pour rendre des applications, initialement mono-utilisateur, coopératives. Nous nous intéressons en particulier aux mécanismes fournis par MacOs pour faire communiquer des processus s'exécutant sur le même site ou sur des sites distants.

Nous plaçons cette étude dans le cadre de l'intégration d'applications au niveau du système de fenêtres. Ce choix est motivé par le fait que les solutions fournies sont des solutions communes à toutes les applications Macintosh. Les mécanismes fournis par MacOs ne soient pas tout à fait comparables à ceux fournis par une plate-forme X-Windows ; trois catégories d'événements pouvant être exploitées pour faire communiquer des applications.

Les dernières générations des systèmes Macintosh, MacOS 7.x et MacOS 8 offrent des mécanismes évolués, du niveau applicatif, qui permettent de faire communiquer entre elles des applications. Ces mécanismes constituent les outils de base pour construire un environnement coopératif intégrant des applications existantes.

Les applications Macintosh reçoivent généralement des informations au sujet de l'environnement matériel et logiciel de la machine. Les événements constituent le moyen à travers lequel le Gestionnaire d'Événements (EM: *the Event Manager*) communique des informations à propos des actions utilisateur, du statut d'exécution de l'application, et d'autres configurations de fonctionnement qui requièrent des réponses de l'application.

Les événements reçus par une application se classent en trois classes d'événements :

- des événements de *bas-niveau (low-level events)*. Généralement, le gestionnaire d'événements envoie à destination de l'application des événements de bas-niveau suite à des actions de l'utilisateur menées à partir des boutons de la souris ou à partir des touches du clavier (les événements sont dans ce cas générés par le module *Operating system event Manager*), au rafraîchissement des fenêtres allouées à l'application (le module *Window Manager*), à l'insertion d'une disquette dans le lecteur de la machine provoquent aussi la génération d'événements de bas-niveau,
- des événements *systèmes* générés lors du changement du statut d'exécution de l'application. Par exemple, dans le cas où l'application passe d'une exécution en *foreground* à une exécution en *background*, le gestionnaire d'événements génère un événement système approprié, *resume event*. Dans ce cas, la réactivation de l'application est faite conjointement par le gestionnaire de processus et le système de fenêtres (cf. Fig. 2.6),
- et des *événements de haut-niveau (high-level events)* : Dans les systèmes Macintosh ultérieurs à la version 7, le gestionnaire d'événements fournit aux applications des routines qui leur permettent de communiquer en échangeant des événements de *haut-niveau*. Un événement de haut niveau est utilisé par une application pour envoyer des informations à une autre application, pour recevoir des informations provenant d'une autre application, ou pour commanditer une action à une autre application.

Par exemple, une application peut envoyer un événement de haut-niveau à destination d'une autre application (qui peut bien être une autre instance de la même application) pour lui demander d'exécuter une action spécifique, telle que l'ajout d'une colonne dans un tableur, ou le changement de la police de caractères d'un paragraphe dans un éditeur de documents. Une application peut envoyer une requête d'information dans un événement de haut-niveau, par exemple, pour demander à une application dictionnaire de retourner la définition d'un mot donné.

Les AEs (*Apple Events*) sont des événements de haut-niveau dont la structure et l'interprétation sont prises en compte par le protocole AEIMP (*Apple Event Interprocess Messaging Protocol*).

La structure de données utilisées par le gestionnaire d'événements pour décrire un événement donné est un enregistrement : *EventRecord*. Cette structure contient des informations sur la nature de l'événement (e.g. activation d'un bouton de la souris, pression sur une touche du clavier) et contient des informations supplémentaires sur l'événement, par exemple, l'identification d'une touche activée. La description complète de la structure est donnée dans la figure Fig. 2.5:

```
Type EventRecord =
  RECORD
    what: Integer; {Code de l'événement :
    0=nullEvent, 1=mouseDown, 2=mouseUp, 3=keyDown, ...
    }
    message: LongInt; {Message de l'événement : }
    when: LongInt; {Date de génération}
    where: Point; {Position de la souris}
    modifiers: Integer; {Information sur les
    touches de modification et le bouton de la
    souris}
  END;
```

Fig. 2.5 : Représentation d'un événement

L'attribut *what* désigne le type d'un événement donné alors que l'attribut *message* représente l'information supplémentaire associée à l'événement. Dans le cas où *what = kHighLevelEvent*, l'attribut *message* désigne la classe à laquelle l'événement appartient.

Pour les événements de niveau-bas et les événements système, l'attribut *where* désigne la position du curseur, celle-ci étant donnée par des coordonnées

globales. Pour les événements de haut-niveau, *where* contient un deuxième identifiant *eventId* qui définit le type particulier de l'événement défini dans la classe d'événements donnée par l'attribut *message*. Dans ce cas, l'attribut *where* est de type *OsType*.

L'attribut *modifiers* contient des informations concernant l'état des touches de modification (e.g. les touches : *Caps Lock*, *Control*, *Shift*) et le bouton de la souris.

Remarque : Pour le système MacOS 6, le Multifinder offre quelques fonctions fournies par le gestionnaire de processus (process manager). Sur les machines équipées d'un système 6 sans Multifinder, un seul contexte d'application est supporté à la fois.

Fig. 2.6 : Les sources d'événements d'une application Macintosh

Schéma de fonctionnement défini par la Fig. 2.6 :

Conformément au schéma de la Fig. 2.6, les événements de bas-niveau sont créés par le module *Operating System Event Manager*, et ensuite transmis à une

file d'événements (*Operating System event queue*) gérée par le système d'exploitation. Le gestionnaire d'événements récupère les événements qui y sont stockés, puis les délivre, un à la fois, à l'application concernée.

Par ailleurs, le gestionnaire d'événements permet aux applications de s'échanger entre elles des événements de haut-niveau. Pour cela, il fait appel à des fonctions fournies par la boîte à outil *Program-to-Program Communication (PPC)*, qui se charge de l'émission et de la réception des événements entre les applications.

Toutefois, il est important de noter que chaque application peut utiliser ces mécanismes de communication pour s'envoyer à elle même des événements. Cette fonction est très intéressante pour contrôler l'exécution de l'application, notamment son interface utilisateur.

Le gestionnaire d'événements : EM (Event Manager)

Le gestionnaire d'événements peut être utilisé par l'application pour envoyer et recevoir des événements de haut-niveau. Pour l'émission, le gestionnaire d'événements utilise la boîte à outil PPC (*Program-to-Program Communication*) (cf. Fig. 2.6). Pour toute application ouverte, capable de recevoir des événements, il maintient à jour une queue d'événements. La taille de cette queue est déterminée par la taille de la mémoire disponible.

Par ailleurs, une application peut utiliser le mécanisme AEM (*Apple Event Manager*) pour communiquer. Il présente un ensemble de routines de communication pour les événements *Apple Event*.

Les routines offertes par le gestionnaire d'événements et le gestionnaire d'événements *Apple Event* pour la communication des événements de haut-niveau utilisent des services de la boîte à outil *Program-to-Program Communication*.

La communication entre applications : IAC (Interapplication Communication)

Pour mettre en œuvre une communication inter-application, l'IAC fait intervenir cinq entités du système : le gestionnaire d'édition (*the Edition Manager*), le support de script (*Open Scripting Architecture*), le gestionnaire d'événements *AppleEvent (Apple Event Manager)*, le gestionnaire d'événements (*Event Manager*) et la boîte à outil PPC (*Program-to-Program Communications (PPC) toolbox*).

l'IAC (Interapplication communication) est un mécanisme standard et extensible pour la communication entre applications Macintosh. Les attributions de IAC sont regroupées de la manière suivante :

- Le gestionnaire d'édition permet de réaliser des opérations copier/coller entre différentes applications.
- le support de script permet de contrôler des applications à partir de programmes scripts,
- le gestionnaire d'événements *Apple Event* (AEM) permet l'émission et la réception d'événements de haut-niveau *Apple Event* entre applications,
- le gestionnaire d'événements permet d'émettre et de recevoir des événements de haut-niveau autres que les *Apple Event*,
- le module (PPC) *Program-to-Program Communication* permet de lire et d'écrire des blocs de données entre des applications.

La figure Fig. 2.7 met en évidence les relations existantes entre les cinq mécanismes de communication offerts par l'IAC. La dépendance entre ces mécanismes est descendante de gauche à droite : le gestionnaire d'édition et le support de script OSA utilisent le gestionnaire d'événements *Apple Event*, le gestionnaire d'événements *Apple Event* (AEM) utilise à son tour le gestionnaire d'événements EM, etc.

Fig. 2.7 : Les mécanismes de communications fournis par l'IAC

Communications moyennant les événements *Apple Events*

L'exigence principale requise d'une communication de *haut-niveau* (communication utilisant des événements de haut-niveau) est l'utilisation d'un vocabulaire d'événements commun. Cette condition est rempli par le protocole AEIMP (*Apple Event Interprocess Messaging Protocol*). Les événements de haut-niveau conformes à ce protocole sont les événements *Apple Event*. Les quatre *Apple Event* communément utilisés par une application sont émis par le Finder : *Open Application*, *Open Documents*, *Print Documents* et *Quit Application*.

La figure Fig. 2.8 montre un exemple d'utilisation de l'événement *Open Documents Event*. Dans cet exemple, le *Finder Macintosh* joue le rôle du client, il demande à l'application *surfWriter* d'ouvrir les documents "Dec. Invoice" et Nov. Invoice". L'application répond au Finder en ouvrant une fenêtre pour chacun des documents spécifiés.

Mise en œuvre de la coopération

Les fonctions fondamentales devant être fournies par les applications sont des fonctions de communication et de contrôle de l'interface utilisateur. Chaque application doit pouvoir recevoir des événements, accomplir les traitements qui leur sont appropriés, rendre compte de son exécution et émettre des événements vers d'autres processus (applications). Ces attributions sont mises en œuvre par les fonctions suivantes :

1) *La réception des événements :*

Une application utilisée en mode coopératif reçoit des événements en provenance de deux types de source : son interface utilisateur et d'autres processus.

Par exemple, lors de l'accomplissement d'une tâche d'édition coopérative, à un moment donné, seul un groupe d'utilisateurs bénéficie du droit d'agir sur les documents partagés. Afin de bloquer les tentatives des autres utilisateurs, l'application doit intercepter tous les événements générés suite à leurs actions.

Sachant que les actions menées explicitement par l'utilisateur sur l'interface de l'application sont essentiellement accomplies au moyen de la souris et du clavier, l'application doit donc intercepter les événements de bas niveau : activation de boutons de la souris et activation de touches claviers.

Par ailleurs, l'application est susceptible de recevoir des événements en provenance d'autres processus (ou applications). Il s'agit en particulier des comptes-rendus des actions accomplies sur d'autres sites, et devant être reproduites pour tous les participants.

L'application reçoit les événements l'un après l'autre en interrogeant le gestionnaire d'événements au moyen de la fonction *WaitNextEvent*. Cette fonction récupère le premier événement dans la file attribuée à l'application et le retourne dans une structure *EventRecord*. La fonction retourne une valeur booléenne *FALSE* si aucun événement n'est disponible.

La figure Fig. 2.9 illustre un exemple d'utilisation de la fonction *WaitNextEvent*. Dans cet exemple, la fonction retourne le premier événement quelque soit sa catégorie (bas-niveau, système ou haut-niveau).

```

VAR
  eventMask: Integer;
  event : EventRecord;
  cursorRgn: RgnHandle;
  mySleep: LongInt;
  gotEvent: Boolean;

  eventMask:= everyEvent;{accepter tous les événements}
  mySleep := MyGetSleep; {Donner une valeur appropriée}
  cursorRgn := MyGetRgn; {donner une région appropriée}
  gotEvent := WaitNextEvent (eventMask, event, mySleep,
  cursorRgn);

```

Fig. 2.9 : Utilisation de la fonction WaitNextEvent

Les paramètres de la fonction *WaitNextEvent* désignent respectivement un masque d'événements (pour préciser quelles catégories d'événements doivent être prises en compte), l'événement récupéré, le temps au bout duquel l'application libère le processeur si aucun événement ne lui est destiné, et, enfin, la région de l'écran pour laquelle le gestionnaire d'événements ne doit pas générer d'événements de mouvement de souris.

Généralement, la prise en compte des événements est faite à l'intérieur d'une boucle de traitements d'événements. La Fig. 2.10 illustre l'utilisation de la fonction *WaitNextEvent* dans une telle configuration.

```

Procedure MyEventLoop;
VAR
  cursorRgn: RgnHandle;
  gotEvent: Boolean;
  event: EventRecord;
BEGIN
  cursorRgn := newRgn; {donner une region vide pour le première
  appel}
  REPEAT
    gotEvent := WaitNextEvent(everyEvent, event, MyGetSleep,
    cursorRgn);
    IF (event.what<>kHighLevelEvent)AND(NOT gInBackground)
    THEN MyAdjustCursor(event.where, cursorRgn);
    IF gotEvent {event différent de null}
    THEN DoEvent(event) {traitement}
    ELSE DoIdle(event); {ne rien faire}
  UNTIL gDone {boucler jusqu'à ce que l'utilisateur quitte}
END

```

Fig. 2.10 : Exemple d'utilisation d'une boucle de traitements d'événements

L'application peut spécifier l'ensemble des événements auxquels elle s'intéresse en choisissant une valeur appropriée du masque. En général, une procédure de traitement dite *handler fonction* est prévue pour chaque type

d'événement spécifié. Dans l'exemple précédent, la fonction *DoEvent* est la procédure de traitement commune à tous les événements retournés dans *gotEvent*.

Afin de munir l'application de fonctions coopératives, il est possible de définir un ensemble d'événements spécifiques correspondant à une nouvelle classe d'événements. Par exemple, pour définir une classe d'événements de contrôle (pouvant être envoyés par un module de contrôle "coop" qui gère l'exécution de l'application) l'attribut *message* de la structure *EventRecord* (cf. Fig. 2.5) est initialisé par la valeur "coop". Dans ce cas, pour représenter une commande de contrôle "cmd1" provenant de "coop", nous utilisons un événement dont l'attribut *where* est initialisé par la valeur "cmd1". L'événement de haut-niveau est considéré de type "cmd1", et appartient à la classe d'événements "coop".

La figure Fig. 2.11 donne le schéma de la procédure *DoHighLevelEvent* qui prend en compte la réception d'un événement de haut-niveau. Il s'agit de l'événement *kMySpecialHLEventId* appartenant à la classe *kMySpecialHLEventClass*.

```

Procedure DoHighLevelEvent(event: EventRecord);
VAR
  myTarg: TargetID;
  myRefCon: LongInt;
  myBuff: Ptr;
  myLen: LongInt;
  myErr: OsErr;
BEGIN
  IF(event.message=LongInt(kMySpecialHLEventClass))      AND
  (LongInt(event.where)=LongInt(kmySpecialHLEventID))
  THEN BEGIN
    {événement de de haut-niveau qui n'utilise pas AEIMP}
    myLen:=0;
    myBuff:= NIL;
    myErr:=AcceptHighlevelEvent(mytarget,          myRefCon,
    myBuff, mylen);
    IF myErr = bufferIdSmall THEN
      BEGIN
        myBuff := NewPtr(myLen);
        myErr:=AcceptHLevelEvent(myTarget,          myRefCon,
        myBuff, myLen);
        IF myErr <> noErr THEN
          {traitement de l'événement ...}
      END
    IF myErr = noErr THEN
      DoError(myErr);{prendre en compte l'erreur}
    END
  ELSE
    BEGIN {l'événement est un Apple Event}
      myErr:=AEProcessAppleEvent(event);
      IF myErr<>noError THEN
        DoError(myerror);
    END;
  END;
END;

```

Fig. 2.11 : Traitement effectué lors de la réception d'un événement de haut-niveau.

La fonction *AcceptHighLevelEvent* permet de récupérer une éventuelle information supplémentaire concernant l'événement reçu. Il s'agit de l'identificateur ID de l'émetteur. L'information est récupérée dans une structure *TargetID* (cf. Fig. 2.12) dont les attributs permettent de reconnaître l'application émettrice, la référence de la session désignant la connexion avec cette application ainsi que le port utilisé et la localisation de l'application.

```

TYPE targetID =
  RECORD
    sessionID: LongInt; {identificateur de la session}
    name: PPCPortRec; {port de l'émetteur}
    location: LocationNameRec; {machine de l'émetteur}
    recvrname: PPCPortRec; {reserved}
  END;

```

Fig. 2.12 : La structure de données TargetID

L'attribut *sessionId* n'est autre que la référence attribuée par le module *Program-to-Program Communication*, défini dans Fig. 2.6, à la session de communication.

2) *L'émission des événements :*

La fonction *PostHighLevelEvent* permet d'envoyer un événement de haut-niveau à destination d'une application. Six informations sont nécessaires pour accomplir cette opération : une structure *EventRecord* dûment remplie, l'identité du destinataire, un identificateur unique qui définit l'émission de l'événement, un *buffer* pouvant contenir des informations supplémentaires, la longueur du *buffer* et des options d'émission. S'il s'agit d'un *Appel Event*, il est possible d'utiliser la fonction *AESend* du gestionnaire d'événements *Apple Event*.

L'exemple de la Fig. 2.13 illustre l'émission d'un événement de haut-niveau, de type "cmd1" appartenant à la classe "boff". La destination est une application du même site définie par la signature "boff". La signature est une information retournée par la fonction *GetProcessInformation*. Par exemple, la signature de l'application *TeachText* est "ttx".

```

PROCEDURE MyPostTest;
VAR
  myEvent:EventRecord;
  myRecvID: OSType;
  myOpts: LongInt;
  myErr: OSerr;
BEGIN
  myEvent.what:= kHighlevelEvent;
  myEvent.message:=LongInt('boff');
  myEvent.where:= Point(LongInt('cmd1'));
  {the receiver is identified by its signature and a return
  receipt is requested}
  myOpts:=receiverIDisSignature + nReturnReceipt;
  myRecvID:='boff';
  myErr:=PostHighLevelEvent(myEvent, ptr(myRecvID), 0, NIL,
  myOpts);
  IF myErr <> noErr THEN
    Doerror(myErr);
  END;

```

Fig. 2.13 : Exemple d'utilisation de la fonction PostHighLevelEvent pour l'émission d'un événement de haut-niveau

Les informations concernant le destinataire de l'événement peuvent être récupérées à l'aide de la fonction *GetProcessInformation*, fournie par le module *Process Manager* (cf. Fig. 2.6). Cette fonction peut être utilisée pour avoir la liste des processus s'exécutant sur le site. Les informations sont retournées dans une structure *ProcessInfoRec* définies dans Fig. 2.14.

```

TYPE ProcessInfoRec =
  RECORD
    processinfoLength : LongInt;
    processName: StringPtr;
    processNumber: ProcessSerialNumber;

    processType: longInt;
    processSignature: OSType;
    processMode: LongInt;
    processLocation: Ptr;
    processSize: longInt;
    processlauncher: ProcessSerialNumber;

    processLaunchDate: LongInt;
    processActiveTime: LongInt;
    processAppSpec: FSSpecPtr;
  END;

```

Fig. 2.14 : Structure de données retournée par la fonction GetProcessInformation

II.2.3 Schémas de mise en œuvre des architectures

II.2.3.1 Structuration modulaire du collecticiel

Nous nous intéressons à présent à la répartition des différents modules qui constituent un collecticiel dans un environnement client/serveur.

D'un point de vue conceptuel, un collecticiel synchrone peut être décomposé en trois modules dont les fonctions sont complémentaires. Au niveau le plus bas, le collecticiel s'appuie sur les fonctions d'un système de fenêtres (SF) qui sont utilisées pour l'affichage de l'interface utilisateur. Au niveau le plus haut se trouve l'application existante, initialement mono-utilisateur, que l'on intègre avec une des méthodes précédentes. Entre ces deux "modules", un module de contrôle de la coopération est nécessaire. Ce module remplit essentiellement les fonctions de contrôle d'accès aux données partagées et de mise en œuvre des interactions entre sites il contrôle donc l'accès à l'interface de l'application et se charge de reproduire les commandes d'un utilisateur sur tous les sites engagés dans un travail coopératif.

Les différents modules qui composent le collecticiel sont instanciés de la manière suivante :

- ***Un module SF*** est associé à chaque utilisateur participant au collecticiel. Ce module gère les interactions de l'utilisateur avec le collecticiel. Il transforme les actions perçues à travers les unités d'entrée standards de la machine (clavier, souris) en événements typés du SF (e.g. *XEvents* pour X-Windows) destinés aux applications s'exécutant dans le collecticiel. Par ailleurs, le module SF répond aux requêtes des applications pour afficher le compte-rendu de leurs exécutions sur l'interface utilisateur.
- ***Chaque module application partagée*** représente une instance d'une application partagée s'exécutant dans le collecticiel. Chaque application peut être soit centralisée, auquel cas elle est représentée par une instance unique s'exécutant sur un site donné, soit dupliquée, auquel cas elle est représentée par plusieurs instances, chacune d'elle exécutée sur le site d'un utilisateur. Dans les deux cas de figure, toutes les instances de l'application sont identiques, elles désignent l'application initialement mono-utilisateur.
- **Le module de contrôle** est chargé de mettre en œuvre les mécanismes de coopération. Sa fonction principale est de multiplexer les entrées/sorties [Lauwers 90] entre application(s) partagée(s) et SFs dans

le cas d'une intégration au niveau bas. Ce module est commun à toutes les applications du collecticiel, les fonctions de coopération qu'il réalise sont :

- a)** la connexion–déconnexion dynamique des utilisateurs (réalisée par le sous–module *Connexion/Déconnexion*),
- b)** l'accès à l'espace partagé (sous–module *Contrôle d'accès* aux applications partagées),
- c)** la gestion des interactions entre les utilisateurs (sous–module *Communication*),
- d)** le codage et le décodage des messages échangés entre les modules de contrôle (sous–module *Codeur/Décodeur*).

Pour la réalisation de ces fonctions, le module de contrôle maintient à jour des informations qui concernent essentiellement les utilisateurs (localisation, authentification, droits d'accès alloués) et l'espace partagé (applications partagées exécutées dans le collecticiel, ressources attribuées à chaque application, utilisateurs participant à chaque application, politiques de résolutions de conflits, ...).

Le fonctionnement général de ce module est le suivant : chaque tentative d'action accomplie par un utilisateur sur l'espace partagé (sur l'une des instances du collecticiel) est interceptée par le contrôleur avant d'être délivrée à l'application. Le contrôleur reconnaît l'action grâce au sous–module *Codeur/Décodeur*, fait appel au sous–module *Contrôle d'accès* afin de l'autoriser ou de l'interdire, puis se charge de la diffuser vers les autres sites à travers le sous–module *Communication*.

Le contrôleur a une structure modulaire. Il regroupe toutes les fonctions (représentées chacune par un module) de contrôle (pour l'accès aux données et la communication) qui peuvent être partagées par plusieurs applications. Ainsi, le module *Contrôle d'accès* implante une bibliothèque de politiques de contrôle et de gestion de la concurrence, le module *Connexion/Déconnexion* implante des politiques de connexion–déconnexion dynamique, et le module *Communication* implante les fonctions d'interaction entre utilisateurs. Le sous–module *Codeur/Décodeur* permet respectivement d'encoder et de décoder les messages échangés entre les sites. La nature de ces messages dépend du schéma d'architecture du collecticiel, ainsi que de l'approche d'intégration adoptée. En général, les informations contenues dans un message

permettent d'identifier l'utilisateur qui est l'origine de l'action, l'opération ayant provoquée la transmission (e.g. insertion d'une chaîne de caractère) et les ressources partagées touchées par l'opération (e.g. documents, fenêtre). Selon cette décomposition, le module de contrôle peut être facilement augmenté par d'autres fonctions (e.g. synchronisation, qualité de service (QoS)).

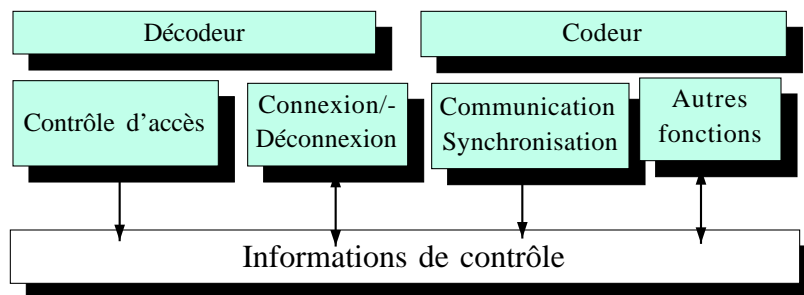


Fig. 2.15 : Fonctions principales du module de contrôle

L'architecture de mise en œuvre du collecticiel dépend essentiellement de la manière d'agencer et de répartir les modules "application" et "module de contrôle" entre les utilisateurs et sur les sites d'exécution.

II.2.3.2 Schéma d'architecture centralisé

Dans une architecture totalement centralisée, les différents modules du collecticiel sont organisés de la manière suivante (cf. Fig. 2.16) :

- une instance unique du module de contrôle est exécutée dans le collecticiel ; le choix du site d'exécution est généralement fait au lancement du collecticiel par le premier participant,
- une instance unique de chaque application partagée existe ; le site d'exécution est généralement désigné par l'utilisateur qui lance l'application en premier,
- un module SF s'exécute dans l'environnement de chaque utilisateur connecté au collecticiel.

Cependant, l'appellation centralisée est parfois déterminée uniquement par le nombre d'instances de l'application. Ainsi, certaines architectures centralisées peuvent faire intervenir plusieurs instances du contrôleur. C'est par exemple le cas pour XTV où une partie du module de contrôle est exécutée comme client virtuel du SF sur chaque site participant (cf Fig. 2.16). Cette configuration où une partie du module de contrôle chargée de la fonction de communication (PTP)

s'exécute sur chaque site utilisateur est indispensable pour prendre en compte les demandes de connexion de nouveaux sites après le lancement du collecticiel. Le cas échéant, les sites participants sont fixés définitivement avant le début de la coopération (i.e. pas de connexion dynamique de nouveaux participants).

L'implantation du module de contrôle dépend de l'architecture du collecticiel. Ainsi, la description des sous-modules, *Connexion-Déconnexion dynamique*, *Contrôle d'accès* et *Communication* doit être élaborée pour chaque schéma d'architecture présenté dans II.2.3.2 et II.2.3.3. Cependant, le fonctionnement reste fondamentalement le même pour tous les schémas d'architecture.

Fonctionnement dans le cas d'une intégration au niveau bas

Conformément à l'architecture centralisée, le fonctionnement du collecticiel dépend de l'approche d'intégration adoptée pour construire les applications partagées.

Dans cette architecture, le module de contrôle est placé entre l'application et le système de fenêtres. Par ailleurs, le module de contrôle est connecté aux systèmes de fenêtres s'exécutant sur tous les sites utilisateur distants (cf. Fig. 2.16).

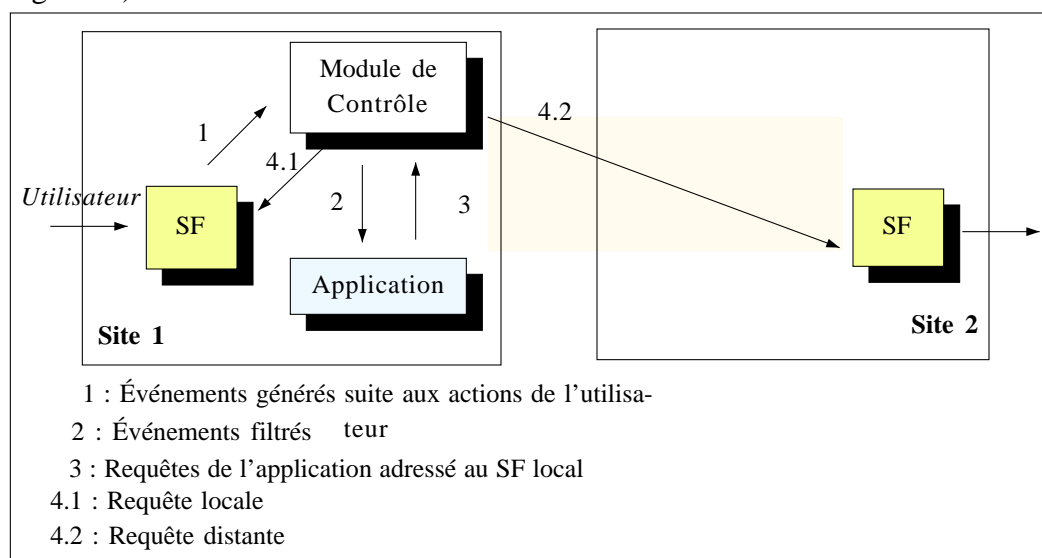


Fig. 2.16 : Architecture centralisée

Chaque action sur l'interface de l'application est transformée par le SF en un ensemble d'événements codés. Ces événements contiennent généralement des informations sur la nature de l'action (e.g. touche clavier, mouvement du pointeur de la souris) et sur l'application (e.g. fenêtre associée à l'application).

Le module de contrôle intercepte les événements en provenance des SFs (i.e. SF local ou l'un des SFs distants) et se charge de les filtrer avant de les transmettre à l'application. L'opération de filtrage consiste à :

- identifier le site sur lequel s'exécute le système de fenêtres émetteur des événements,
- vérifier les droits d'accès associés à l'utilisateur du site émetteur en fonction de la nature de l'action et de l'application,
- si l'utilisateur est autorisé à accomplir l'action, délivrer les événements à l'application, sinon ignorer les événements reçus.

En recevant les événements, l'application s'exécute et transmet, à son tour, les résultats de son exécution au module de contrôle. Celui-ci se charge de diffuser les résultats vers tous les sites du collecticiel.

Dans le cas du SF X-Windows, l'instance unique de l'application partagée doit pouvoir afficher ses sorties sur plusieurs écrans, ce qui reviendrait à dupliquer son interface utilisateur sur chaque site participant. Par ailleurs, et conformément aux hypothèses de construction par intégration, le code de l'application n'est en aucun cas modifiable. La distribution des sorties de l'application est donc prise en compte par le module de contrôle. Celui-ci se comporte en tant que client (application) virtuel de tous les serveurs X distants ; il duplique les requêtes de l'application, normalement transmises au serveur X local, pour les diffuser vers tous les serveurs X du collecticiel [Abdel-Wahab 91]. À cet effet, il gère pour chaque application partagée l'ensemble des ressources qui lui sont allouées par chaque serveur X, ainsi que la correspondance entre les ressources qui désignent un même élément de l'interface utilisateur (e.g. identificateur de fenêtre principale de l'application).

Dans cette architecture, le module de contrôle est connecté à l'application ouverte selon le schéma de la Fig. 2.1. L'application étant représentée par une instance unique, il n'est pas possible d'implanter une architecture centralisée conformément à la définition de l'intégration d'applications ouvertes (cf. II.2.1) dans laquelle les informations échangées entre les sites sont des événements délivrés par l'application ; ces derniers ne pouvant être interprétés que dans le contexte de l'application, la présence d'une instance de l'application sur chaque site devient indispensable. De ce fait, la seule mise en œuvre possible est fondée sur l'échange des informations générées par le SF ; elle consiste à découper le module de contrôle en deux sous-modules instanciés comme suit :

- Un sous-module de communication est placé entre l'application et le SF. Ses fonctions sont identiques à celles fournies par le module de contrôle dans le cas d'une intégration au niveau du SF, excepté la fonction de contrôle d'accès. Celle-ci est réalisée par le second sous-module.
- Un second sous-module qui assure le contrôle d'accès est connecté à l'application conformément au schéma de la Fig. 2.17. Cette configuration permet de bénéficier de la richesse de l'information contenue dans les événements délivrés par l'application pour mettre en œuvre les politiques de contrôle d'accès adaptées aux applications ouvertes tout en véhiculant des événements SF entre les sites.

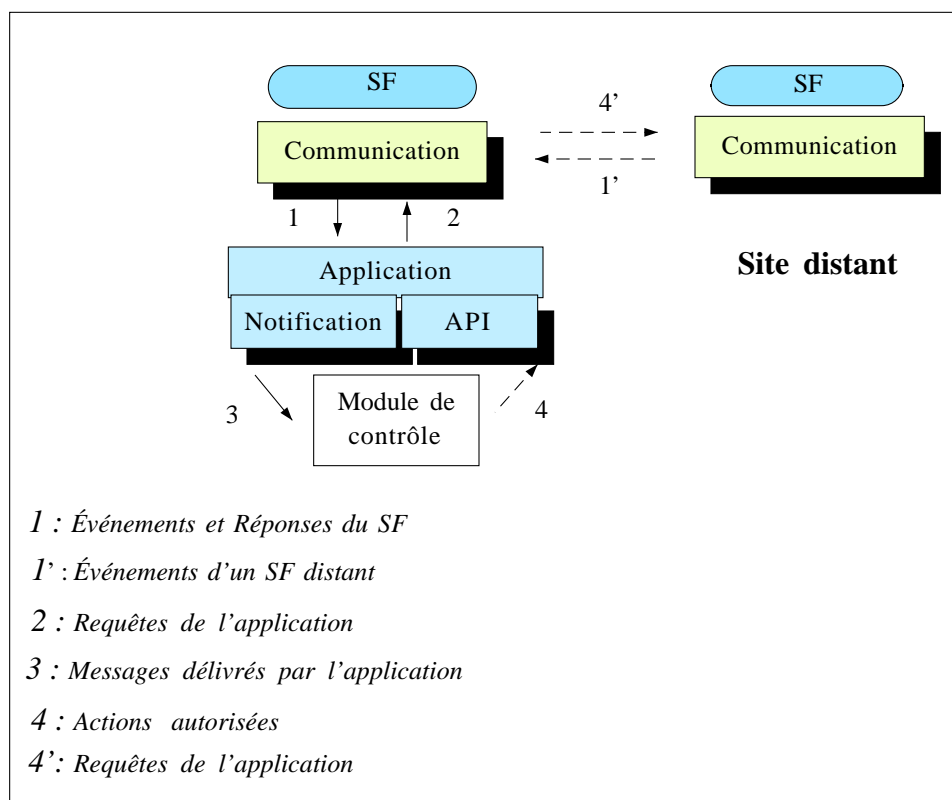


Fig. 2.17 : Architecture centralisée pour l'intégration d'applications ouvertes

Discussion

L'approche centralisée est simple à mettre en œuvre. Elle est adoptée dans plusieurs plates-formes pour collecticiels notamment dans XTV et Rendez-vous [Patterson 90]. En effet, Rendez-vous est un bon exemple d'architecture centralisée. Le système est basé sur une entité centrale connectée à

travers plusieurs vues attribuées aux utilisateurs. Ce schéma de connexion définit le paradigme ALV (Abstraction–Link–View Paradigm) [Hill 92].

Conformément à l'architecture ALV, chaque vue de l'application est implantée par un processus léger lancé à partir d'un même processus père. Dans le cas de n participants, si chacun d'eux effectue une action non conflictuelle (e.g. défilement dans une fenêtre), $o(n^2)$ messages sont véhiculés à travers le réseau. Chaque message nécessite une transmission vers l'entité centrale, et $n-1$ transmissions sont effectuées par l'entité centrale vers les autres vues. Pour l'ensemble des n utilisateurs, le nombre de messages émis est égal à $n*(n-1)$, soit $o(n^2)$.

En conclusion, cette architecture présente tous les avantages d'une implantation centralisée, notamment :

- une mise en œuvre simplifiée des fonctions de contrôle d'accès aux données partagées, celles-ci étant présentes en un seul exemplaire sur le site du module de contrôle,
- l'utilisation d'une seule instance de l'application partagée ; ceci peut être utile dans le cas où l'application n'est pas disponible sur tous les sites.

Cependant, une architecture centralisée présente quelques inconvénients, parmi lesquels :

- l'impossibilité d'appliquer cette architecture lorsqu'on intègre des applications ouvertes et que l'on souhaite échanger des événements générés par l'application entre les sites. Du fait que les événements échangés entre les sites sont des événements générés par l'application, celle-ci doit être exécutée sur chaque site impliqué dans la coopération,
- l'intégration d'une application ouverte conformément au schéma d'architecture de la Fig. 2.17 pose de sérieux problèmes pour la mise en œuvre du contrôle d'accès.

Sachant que si l'on dispose d'une seule instance centralisée de l'application, les échanges entre les sites ne peuvent être effectués que par le biais d'émissions d'événements délivrés par le système de fenêtres. Dans ce cas, le mécanisme API fourni par l'application n'intervient pas dans l'exécution des commandes en provenance de sites distants.

Par ailleurs, le module de contrôle réagit à des événements délivrés par le service de notification (mécanisme ECF dans le cas de l'éditeur Thot). Dans la plupart des cas, un événement généré par l'application synthétise une information contenue dans un ensemble d'événements du système de fenêtres local. Par exemple, pour la commande d'ouverture d'un

document, l'ECF génère une paire d'événements : *StdDocOpen.Pre*, *StdDocOpen.Post* alors que plusieurs événements X sont produits lors de la même opération. Si les événements ne proviennent pas du système local, il est indispensable que la source de ces événements soit connue par les fonctions du module de contrôle, la fonction *CheckFloor* dans l'exemple illustrant l'intégration de l'éditeur Thot (cf. II.2.1.1). Cette information est indispensable pour identifier l'utilisateur qui est à l'origine de la commande.

Pour remédier à cette limitation, il est donc indispensable de rajouter un module de contrôle supplémentaire entre le SF et l'application afin de pouvoir contrôler l'exécution de l'application en tenant compte des droits attribués aux participants,

- un risque non négligeable de goulot d'étranglement au niveau du site central ; cela peut survenir au niveau du module de contrôle si plusieurs utilisateurs effectuent simultanément des tentatives d'action,
- et une grande vulnérabilité aux pannes du site sur lequel s'exécute le module de contrôle.

II.2.3.3 Schéma d'architecture totalement dupliquée

Dans une architecture totalement dupliquée, les modules du collecticiel sont organisés comme suit :

- une instance du module de contrôle est exécutée sur chaque site utilisateur,
- une instance de chaque application partagée est exécutée sur chaque site utilisateur,
- un module SF est placé sur chaque site utilisateur connecté au collecticiel.

Le fonctionnement d'un collecticiel implanté selon une architecture totalement dupliquée est lié à l'approche d'intégration adoptée pour construire les applications partagées qui s'y exécutent.

Fonctionnement dans le cas d'une intégration au niveau bas

Dans cette architecture (cf. Fig. 2.18), les messages échangés entre les sites sont des événements du SF [Ohmori 92].

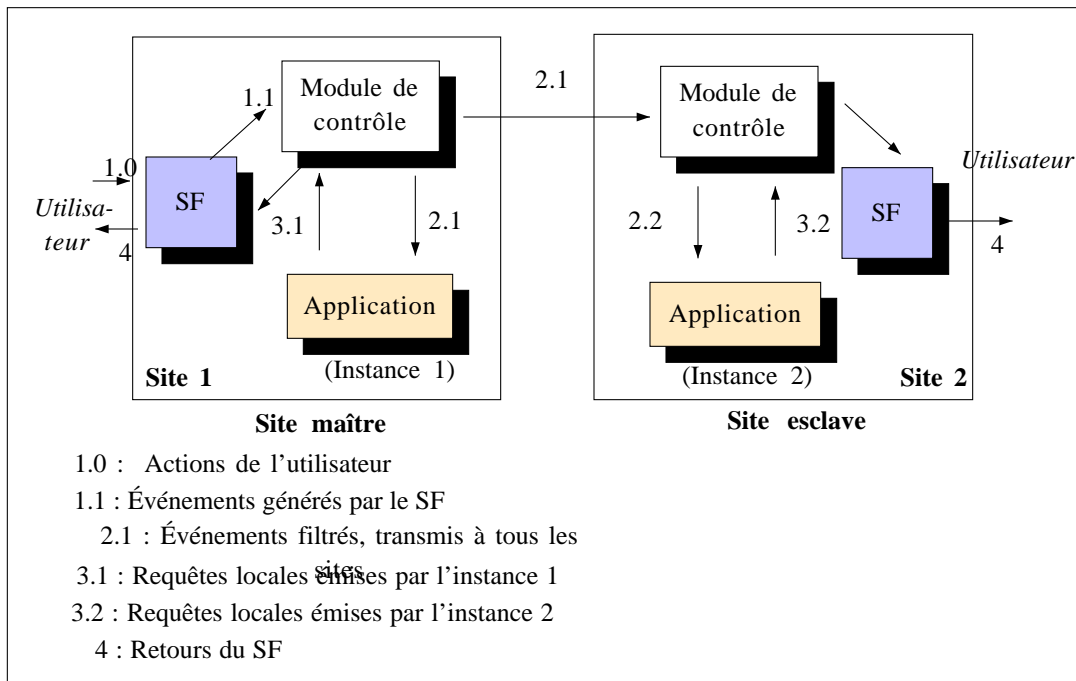


Fig. 2.18 : Architecture totalement dupliquée pour une intégration au niveau bas

Les phases principales du traitement d'une action d'un utilisateur sur l'interface d'une application sont les suivantes.

- **La génération** par le SF d'un ensemble d'événements associés à l'action de l'utilisateur. Les événements sont codés selon un format unique qui est indépendant de la nature de l'action et de la nature de l'application. Tous les événements générés sont transmis au module de contrôle local.
- **Le contrôle**, qui consiste à autoriser ou à ignorer l'exécution de l'action en fonction des informations contenues dans les événements du SF, et des droits d'accès associés à l'utilisateur (informations gérées par le module de contrôle). A cet effet, le module de contrôle décode les événements reçus et réagit comme suit en fonction de leur provenance :
 - a) si l'action de l'utilisateur est autorisée, les événements émis par le SF local sont communiqués à l'application destinataire et diffusés vers tous les modules de contrôle s'exécutant sur les autres sites ; ces événements sont ignorés dans le cas contraire,
 - b) les événements en provenance des SFs distants contiennent des informations liées au contexte d'une application s'exécutant sur un autre

site ; à partir de ces informations, le module de contrôle identifie l'application locale à laquelle les événements sont destinées et les lui communique.

- **L'exécution**, par opposition au cas centralisé (diffusion des requêtes vers tous les SF), les exécutions des applications partagées sont locales ; les interactions entre application et SF sont identiques aux interactions recensées dans le cas d'un usage de l'application dans un environnement mono-usager.

Fonctionnement dans le cas d'une intégration d'une application ouverte

Dans cette architecture, le principe de fonctionnement du collecticiel est le même que celui observé dans le cas de l'intégration au niveau bas. La différence essentielle réside dans la nature des informations échangées entre les sites ; il s'agit d'événements délivrés par l'application et non plus par le système de fenêtres.

Le module de contrôle est connecté d'une part à l'application qui s'exécute sur le même site, et, d'autre part, aux modules de contrôle qui s'exécutent sur les autres sites.

Les principales étapes du traitement d'une action d'un utilisateur sur l'interface de l'application sont les suivantes :

- **la génération** par l'application d'un ensemble d'événements associés à l'action de l'utilisateur. Les événements sont codés selon un format propre à l'application. Tous les événements générés sont transmis au module de contrôle local,
- **le contrôle**, qui consiste à autoriser ou à ignorer l'exécution de l'action en fonction des informations contenues dans les événements de l'application et des droits d'accès associés à l'utilisateur (informations gérées par le module de contrôle). Le module de contrôle décode ainsi les événements reçus :
 - a) si l'action de l'utilisateur est autorisée, il communique les événements à l'application à travers le module API (cf. Fig. 2.1), et les diffuse vers tous les modules de contrôle s'exécutant sur les autres sites,
 - b) si l'action n'est pas autorisée, les événements sont ignorés,
- **l'exécution** de l'action, qui est locale au site de l'utilisateur conformément aux règles de fonctionnement dans une architecture dupliquée.

Discussion

Les architectures dupliquées présentent plusieurs avantages, dont certains ont été présentés par Sarin et Greif dans [Sarin 85]. Les plus importants sont évoqués dans [Lauwers 90].

- Des temps de réponses meilleurs que ceux obtenus dans le cas centralisé. Ceci provient du fait que les exécutions de l'application sont toujours locales. Les mesures accomplies dans le cadre de l'expérimentation de la plate-forme CoopScan [Ben Atallah 95] montrent que, pour une architecture centralisée, les temps de réponse sont systématiquement augmentés des délais de transmission imposés par l'utilisation du protocole TCP/IP.
- Une meilleure souplesse de mise en œuvre [Cosquer 94]. Dans le cas de certains systèmes de fenêtres tel que SunView, certaines ressources matérielles d'une machine (e.g. l'écran) ne peuvent pas être utilisées par une application s'exécutant sur une machine distante. Dans un tel environnement, l'architecture centralisée ne peut pas être mise en œuvre.
- Une meilleure adaptation aux environnements systèmes et matériels hétérogènes (e.g. ressources disponibles, tailles et distances en pixels écran).

Toutefois, ces architectures présentent certains inconvénients.

- Le risque d'indisponibilité des applications et des ressources sur chaque site (e.g. espace mémoire, licences).
- La prise en compte des connexions et déconnexions dynamiques de sites est plus complexe.
- La synchronisation entre les différentes instances des applications partagées. Cette fonction est indispensable pour garantir la même vue pour tous les utilisateurs du collecticiel. Dans le cas du système de fenêtres X-Windows, la synchronisation des interactions entre sites doit respecter l'exécution du protocole X sur chaque site ; la communication d'un événement en provenance d'un SF distant peut causer une panne d'exécution si l'application est dans une phase d'exécution du protocole (e.g. rupture de séquence si l'application est en attente synchronisée d'un événement généré par l'émission d'une requête au serveur). Par exemple, certaines requêtes sont à l'origine de certains événements locaux⁽¹⁾.

(1) Ces événements ne sont pas échangés entre les sites ; seuls les événements générés suite à des opérations d'entrée-sortie (e.g. touche clavier) sont échangés entre les sites.

II.2.3.4 Schéma d'architectures hybrides

Les architectures discutées dans cette section sont les plus représentatives des environnements client–serveur faisant intervenir un système de fenêtres et des applications clientes. D'autres schémas d'architecture hybrides peuvent être obtenus en combinant les architectures totalement centralisées et celles totalement dupliquées, par exemple, en dupliquant le module de contrôle sur chaque site utilisateur tout en gardant une instance unique de l'application partagée (cf. Fig. 2.19). Cette architecture est très proche de l'architecture totalement centralisée (cf. Fig. 2.16). Les modules de contrôle échangent des requêtes d'affichage émises par l'application au SF. L'avantage de cette architecture est que certaines fonctions du module de contrôle sont réalisées en local (e.g. une action non autorisée sur un document partagé est traitée localement sans être communiquée au site central).

Inversement, nous notons qu'une architecture hybride faisant intervenir plusieurs instances de l'application et une instance unique du module de contrôle ne présente pas beaucoup d'intérêt.

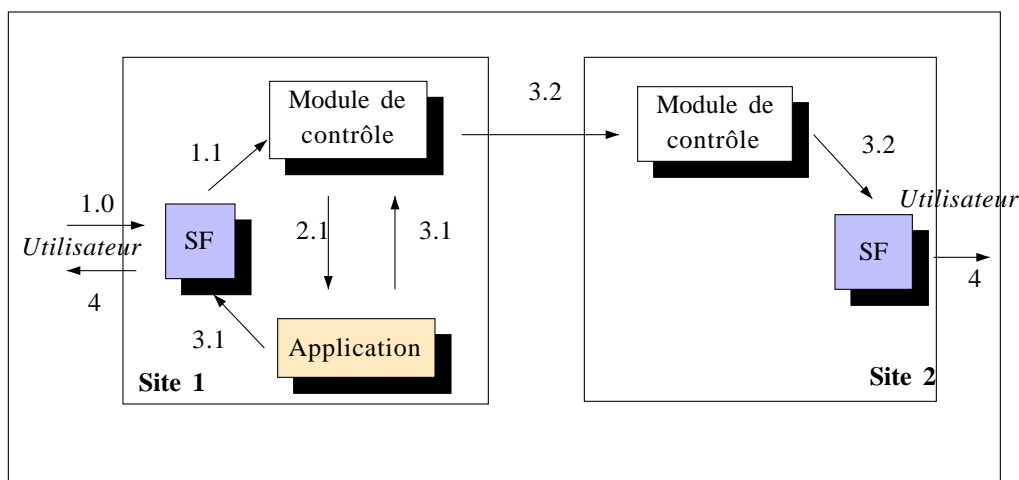


Fig. 2.19 : Architecture hybride (module de contrôle dupliqué)

Notre étude des architectures est fondée sur les échanges élémentaires des informations générées par les applications partagées. D'autres messages, générés par le module de contrôle, sont échangés entre les sites utilisateurs ; ils ne sont pas pris en compte explicitement dans notre travail. Par contre, l'évaluation des échanges élémentaires de ces messages (e.g. messages échangés lors de l'exécution d'un protocole de connexion ou d'élection d'un détenteur du droit de parole) est analogue à celle des messages générés par les applications

partagées, le module de contrôle pouvant être conceptuellement séparé en une interface de contrôle et en un noyau fonctionnel.

II.2.4 Discussion

Les éléments de comparaison qui interviennent dans le choix d'une architecture de collecticiel sont le coût de mise en œuvre et les performances. Nous classons ces critères selon deux points de vue : développement et fonctionnement.

Le **coût de développement** dépend de la complexité du module de contrôle. La généricité de la solution est dans ce cas, une conséquence directe de cette complexité. Les résultats de notre étude sont résumés dans le tableau suivant (cf. Fig. 2.20) :

Module de contrôle	Intégration au niveau bas	Intégration au niveau de l'application
Développement	complexe	peu complexe
Généricité	indépendant de l'application	dépendant du contexte de l'application

Fig. 2.20 : Coût de développement du module de contrôle

Pour le fonctionnement, les critères de choix d'une architecture sont les suivants.

- Le volume des **informations échangées** entre sites : il varie en fonction de l'architecture du collecticiel (répartition des modules et schémas d'intégration). Ce critère fait l'objet d'une étude d'évaluation basée sur une expérimentation décrite en Annexe de ce document. Dans un environnement d'exécution donné, ce critère détermine les temps de réponse et de réaction du collecticiel. Toutefois, ce critère ne peut être significatif que s'il est considéré pour un seul environnement d'exécution à la fois. Pour des opérations élémentaires typiques d'insertion de chaînes de caractères, l'expérimentation fait clairement apparaître que dans une architecture totalement dupliquée, l'information échangée entre les sites est inférieure en nombre de messages et en volume à celle échangée dans une architecture centralisée. Par ailleurs, la même expérimentation montre que le volume des informations échangées dans le cas d'une intégration au niveau de l'application est nettement inférieur à celui des

informations échangées dans le cas d'une intégration au niveau du système de fenêtres.

- Le protocole de communication (e.g. *multicast*, point à point) [Powell 96].
- L'environnement matériel (i.e. les caractéristiques du réseau de communication et des machines utilisées).
- La disponibilité des ressources logicielles sur les sites (i.e. les applications, les fontes, ...).
- Le nombre de participants au collecticiel.

II.3 CoopScan : Une architecture générique pour collecticiels

Nous avons étudié dans ce qui précède les propriétés et les architectures des collecticiels synchrones. Notre étude nous a permis d'identifier les fonctions coopératives fournies par les collecticiels, et d'analyser les architectures possibles pour leur mise en œuvre dans des environnements distribués.

Nous proposons dans cette section un modèle d'**architecture générique** pour la construction et l'exécution de collecticiels. L'implantation de ce modèle est réalisée à travers la mise en œuvre de la plate-forme *CoopScan*.

Le modèle que nous proposons permet la description et la construction de plusieurs applications partagées indépendamment de leurs spécificités (édition de documents, outil de dessin, ...).

Nous prouvons à travers la mise en œuvre de différents prototypes d'applications partagées le gain en termes de coût de développement atteint par l'utilisation de *CoopScan*.

Les aspects approfondis dans cette section sont les suivants :

- la définition du concept *généricité* dans le cadre des collecticiels synchrones,
- la description d'un modèle d'architecture pour collecticiel synchrone,
- la mise en œuvre du modèle pour la construction de diverses applications partagées,

À l'issue de cette étude nous dressons le bilan des objectifs atteints ainsi que des limitations de notre approche pour la construction de nouvelles applications partagées.

II.3.1 Objectifs

La plate-forme *CoopScan* ne vise pas la construction d'un collecticiel spécifique mais plutôt la mise en œuvre d'un ensemble de fonctions qui permettent de rendre coopératives des applications qui initialement ne le sont pas.

Notre objectif principal consiste à fournir un modèle d'architecture *générique*. La généralité que nous cherchons à satisfaire concerne les aspects suivants :

- *la construction* : il s'agit de faciliter le prototypage d'applications coopératives. Notre modèle doit fournir des modules logiciels qui peuvent être ré-utilisés, en totalité ou en partie, pour rendre des applications non coopératives, utilisables par un groupe d'utilisateurs dans le cadre d'un collecticiel synchrone,
- *le fonctionnement* : pour cet aspect, la généralité visée consiste à pouvoir utiliser des applications rendues coopératives dans *CoopScan* pour accomplir des tâches coopératives variées. Plus formellement, il s'agit de fournir des bibliothèques de fonctions de contrôle qui implantent les politiques d'accès aux applications partagées et la gestion de groupes (la connexion-déconnexion de participants et la gestion des pannes). Ces bibliothèques doivent pouvoir être instanciées pour une même application de sorte qu'on puisse l'utiliser pour réaliser des scénarii coopératifs variés. Par exemple, il serait souhaitable de pouvoir utiliser le même éditeur partagé afin de réaliser une télé-réunion, un séminaire ou une séance de télé-enseignement.
- *le support d'exécution* : il s'agit de prouver la généralité concernant la plate-forme d'exécution utilisée. Nous montrons que les choix de conception sont suffisamment généraux pour permettre le déploiement de l'application sur des plates-formes variées.

CoopScan permet de valider l'étude des architectures logicielles pour collecticiels. Nous y utilisons un schéma d'intégration d'applications à deux niveaux : au niveau du système de fenêtre et à l'aide des mécanismes fournis par les applications ouvertes (i.e. les *API*⁽²⁾ et les services de notification).

Par ailleurs, *CoopScan* fournit une application pilote qui sert de banc d'essai pour un modèle et un langage à base de composants développé dans le cadre du projet *SIRAC* et dont l'objectif est la construction d'applications par assemblage de composants logiciels et l'administration de ces applications. D'ailleurs, nous utilisons ce modèle pour décrire l'architecture *CoopScan*. Nous y donnons les

(2) *Application Programming Interface*.

schémas de composition utilisés pour faire communiquer des modules qui encapsulent des applications existantes et des *modules de contrôle* communs à toutes les applications du collecticiel.

II.3.2 Cahier des charges

La plate-forme *CoopScan* doit remplir le cahier des charges suivant :

- fournir des modules ré-utilisables pour rendre des applications existantes coopératives,
- faire coexister dans un même collecticiel plusieurs applications partagées,
- fournir des bibliothèques de politiques d'accès pouvant être instanciées pour des applications variées,
- fournir des services de gestion de groupe permettant la connexion-déconnexion de participants selon différentes politiques pendant l'exécution du collecticiel, et la gestion des pannes dans le collecticiel,
- changer dynamiquement les politiques d'accès aux applications partagées ainsi que le protocole de connexion de nouveaux membres, en garantissant un fonctionnement correct du collecticiel.
- mettre en œuvre *CoopScan* sur des plates-formes variées afin de prouver la généralité du modèle d'architecture proposé.

II.3.3 Choix de conception

Conformément aux objectifs que nous nous sommes fixés, et à l'issue de l'étude menée dans le chapitre I et II.2, nous retenons pour *CoopScan* les choix de conception suivants :

II.3.3.1 La construction d'applications partagées

Les deux approches (étudiées dans II.2) pour la construction de collecticiels sont complémentaires ; l'approche bas niveau est applicable à une grande variété d'applications mais elle ne permet pas d'appliquer des politiques élaborées de contrôle et de coordination des actions des utilisateurs. L'approche haut niveau présente les avantages et les inconvénients inverses. Elle permet essentiellement de manipuler des structures de données spécifiques à l'application qui sont souvent plus simples.

Nous souhaitons que *CoopScan* puisse permettre les deux modes d'intégration d'applications. Nous présentons dans la suite l'articulation de ces deux schémas à travers les modules de contrôle fournis par *CoopScan*.

II.3.3.2 Architecture de mise en œuvre

Selon les résultats de l'étude menée concernant les architectures de mise en œuvre d'un collecticiel, nous optons dans *CoopScan* pour une architecture totalement dupliquée. Ce choix est essentiellement motivé par :

- le temps de réponse : le caractère distribué de l'application peut entraîner des temps de réponses longs si une seule instance de l'application est exécutée.
- la disponibilité de l'application sur chaque site. Bien que cet argument ne soit pas déterminant, il fournit néanmoins une mesure de sécurité supplémentaire contre l'avènement de pannes dans le collecticiel. Nous estimons que la disponibilité de l'application sur chaque site favorise l'interactivité de l'application et par conséquent, des temps de réponse faible.

II.3.3.3 Service de gestion de groupes et politiques de droit de parole

Une application coopérative synchrone doit principalement fournir des fonctions de contrôle qui permettent :

- de gérer les conflits d'accès aux données partagées,
- de garantir l'adhésion du groupe de participants.

Afin de gérer l'accès aux applications de *CoopScan*, nous utilisons des **PDPs (Politiques de Droit de Parole)**. Ces politiques garantissent un accès exclusif aux données partagées. Le droit d'accès y est représenté par un *jeton* logiciel (désignant le droit de parole) que les utilisateurs échangent entre eux. Le protocole selon lequel s'accomplit l'échange du jeton définit la politique d'accès.

Nous recensons dans notre étude un autre type de politiques d'accès, notamment celles fondées sur le principe d'accès "libre" aux données partagées. Les protocoles qui mettent en œuvre ces politiques utilisent le concept d'horloge logique [Lamport 78] afin de garantir un ordonnancement identique des événements sur tous les sites du collecticiel.

Les protocoles garantissant un accès libre sont souvent étroitement liés aux données de l'application et à la sémantique des opérations effectuées sur ces données (cf. [Karsenty 93]). Dans un premier temps, nous ne nous intéressons

pas à cette classe de politiques, la mise en œuvre de solutions génériques étant l'objectif prioritaire.

Dans *CoopScan*, nous proposons un **SGG** (Services de Gestion de Groupes) garantissant la connexion de nouveaux participants, la déconnexion de participants et la gestion des pannes. Ces opérations sont réalisées par des protocoles de gestion dynamique qui vérifient les propriétés suivantes :

- *l'adhésion du groupe* : chaque site a une connaissance de tous les sites du collectif,
- *la gestion des pannes* : le SGG permet l'exclusion de tous les sites défaillants du collectif,
- *la cohérence* : le SGG garantit une vue identique de l'espace partagé,
- *la correction* : les protocoles de connexion et de déconnexion fournis par le SGG vérifient l'absence d'inter-blocage et la propriété de terminaison.

II.3.3.4 La session *CoopScan*

Conformément à la définition donnée dans le chapitre I, le terme *session* désigne un travail coopératif synchrone accompli par un groupe d'utilisateurs pendant une durée de temps finie. Une session *CoopScan* est définie par :

- *l'espace des acteurs* : il s'agit du groupe d'utilisateurs participant à une session *CoopScan*. Les participants sont identifiés de manière unique dans une session. Un identificateur est attribué à chaque utilisateur dès sa connexion. Il correspond à un numéro de port attribué au processus de lancement associé à une adresse *IP* de machine hôte. D'autres informations concernant le participant sont également enregistrées (le nom de *login*, et la date d'arrivée dans la session).
- *l'espace de coopération* : il regroupe l'ensemble des données manipulées par les applications intégrées dans *CoopScan*. Une application exécutée dans la session n'est pas forcément partagée par tous les membres du collectif. Le choix de participer à une application est laissé aux utilisateurs. Un utilisateur qui se connecte à *CoopScan* est donc dans un premier temps admis dans le groupe principal (dans la session), puis il est admis dans l'une ou l'autre des applications en fonction de son choix.

De ce fait, les applications intégrées dans *CoopScan* sont réparties, d'un point de vue utilisateur, en trois groupes dynamiques mutuellement exclusifs :

- *les applications disponibles* : au lancement de la session, ce groupe contient toutes les applications intégrées dans *CoopScan*. En fait, il s’agit des applications disponibles mais qui n’ont pas encore été lancées dans la session,
 - *les applications en session* : il s’agit des applications lancées dans la session, mais auxquelles l’utilisateur ne participe pas encore. Les applications de la session sont des applications qui étaient initialement dans le groupe des *applications disponibles*,
 - *Les applications de travail* : il s’agit des applications auxquelles l’acteur local participe. Les applications de ce groupe se sont trouvées précédemment dans le groupe des applications de la session.
- *l’espace de coordination* : cet espace définit les liens existants entre l’espace des acteurs et l’espace de coopération. Il détient principalement les informations suivantes :
 - *PartList* désigne la liste de tous les acteurs dans *CoopScan*,
 - *AppList* désigne la liste des applications partagée dans *CoopScan*. A chaque application est associée une politique de gestion du droit de parole et un service de gestion de groupe,
 - les rôles des participants : le rôle d’un participant évolue dynamiquement pendant le déroulement de la session. Un acteur peut avoir deux rôles : participant ou président. Le rôle de président est attribué, par défaut, à l’initiateur de la session. Ce rôle peut être cédé au profit d’un autre participant par l’initiative du président lui même, ou suite à une panne. Par ailleurs, le rôle de participant se précise par les droits d’accès qui lui sont dynamiquement attribués pour chaque application de *CoopScan*. Un participant peut être participant à une application sans avoir le droit de parole (*simple participant*), participant à une application et détenteur du droit de parole (*détenteur du jeton*), ou non participant à une application (*Null*). Les rôles des participants se présentent sous la forme d’une liste de tuples $[roll, application_id]$ associé à chaque participant.

Dans *CoopScan*, le président se réserve le droit de récupérer le jeton à n’importe quel moment de la coopération, en particulier lors de la phase de connexion.

 - les politiques de gestion du droit de parole (PDP) disponibles dans *CoopScan*,

- $\{(CurrentPDP, appId)\}$: chaque tuple désigne la politique de droit de parole utilisée pour chaque application partagée. $AppList[AppId].CurrentPDP$ désigne la politique de gestion de parole attribuée à l'application $appId$.
- les services de gestion de groupes (SGG) disponibles dans *CoopScan*,
- $\{(CurrentSGG, AppId)\}$: chaque tuple permet de déterminer la politique de connexion adoptée pour une application donnée. Si $AppId$ a la valeur *Null*, alors $CurrentSGG$ désigne la politique de connexion adoptée dans la session en cours. $AppList[appId].CurrentSGG$ désigne le service de gestion de groupe associé à l'application $appId$.

II.4 L'architecture de CoopScan

II.4.1 Description fonctionnelle de *CoopScan*

Le module central dans la construction d'une plate-forme fondée sur l'intégration d'applications est le module de contrôle (introduit dans II.2.3). Ce module assure les fonctions suivantes :

- la gestion dynamique de groupe dans les collecticiels,
- le contrôle d'accès aux applications partagées,
- la gestion des interactions entre les sites participants.

Afin de garantir la généricité de la plate-forme et d'atteindre les objectifs fixés, nous proposons une architecture modulaire pour construire le module de contrôle dans *CoopScan*.

Nous définissons les trois sous-modules suivants : *agent session*, *agent local*, *agent distant*. Le module *agent session* fournit principalement le SGG et les PDPs. Le module *agent local* est connecté directement à toutes les applications partagées du collecticiel. Il est chargé de l'exécution des PDPs pour chaque application. Enfin, le module *agent distant* est une sorte d'émissaire de l'agent local sur chaque site participant. Il se charge de reconstituer les commandes transmises par l'*agent local* du site distant et de les reproduire sur son site.

Comme l'illustre la figure Fig. 2.21, l'architecture de *CoopScan* est totalement dupliquée. Nous avons donc sur chaque $site_i$: un agent session (AgS_i), un agent local (AgL_i), et un ensemble de $(n-1)$ agents distants; $\{(AgD_j)\}$ où $j \neq i$ et n le nombre de sites.

La Fig. 2.21 illustre un schéma fonctionnel de *CoopScan*. L'exemple présente le fonctionnement de *CoopScan* dans le cas d'une session comprenant trois sites, et intégrant deux applications partagées (*App1* et *App2*). Le site 1 détient le droit de parole sur *App1*. Toutes les commandes *Cmd1* effectuées sur *App1* sont interceptées par l'agent local *AgL1*. Celui-ci vérifie les droits d'accès attribués à l'acteur local. Si l'acteur détient le droit de parole, L'*AgL* diffuse la commande sous la forme d'un message *Msg1* vers tous les sites.

Fig. 2.21 : Schéma fonctionnel de CoopScan et répartition des agents sur les sites participants.

Sur chaque site participant, l'agent distant *AgD1* récupère le message *Msg1* et se charge de lancer l'exécution de la commande *Cmd1* sur son site.

II.4.2 Modèle d'architecture générique

Nous présentons ici le modèle d'architecture selon lequel est construit *CoopScan*. Afin de montrer la généralité de ce modèle, nous décrivons *CoopScan* à l'aide d'un langage à base de *composants* et de *connecteurs*. Les composants sont utilisés pour représenter les différents modules *CoopScan*, alors que les connecteurs permettent de définir les interactions qui existent entre ces modules.

II.4.2.1 Le modèle OLAN

Un langage à base de composants utilise une abstraction similaire à celle utilisée dans les modèles à objets pour décrire des modules logiciels englobant des données et un comportement. Néanmoins, l'apport principal des composants se traduit par une définition plus claire de l'interface des modules et le pouvoir de structuration qui découle du modèle lui-même.

OLAN⁽³⁾ définit un langage et un modèle à base de composants logiciels interconnectables. Les concepts utilisés par OLAN sont les composants et les connecteurs.

– Les *composants* sont des entités logicielles qui encapsulent des modules logiciels existants (ou composés d'autres composants). Du fait de la volonté d'encapsuler n'importe quel "module" existant, il existe des composants ayant des schémas de communication, des structures d'exécution, des langages de programmation très variés.

– Les connecteurs : les composants communiquent à l'aide de connecteurs qui sont conçus pour réaliser trois choses : l'interconnexion entre les composants, la transformation des blocs de paramètres, et la mise en œuvre d'un protocole de communication en utilisant un ensemble de protocoles existants ; *IP*, *TCP*, *ORB*,... Les connecteurs sont instanciables en fonction des besoins en communication affichés par les composants.

Intérêt du modèle OLAN

Le développement d'un tel modèle a pour objectif :

– la construction d'applications coopératives en ré-utilisant des modules logiciels existants (souvent hétérogènes),

– la configuration de ces applications en répartissant d'une manière optimale (par exemple, en fonction de la charge des machines, du débit du réseau, ...) les composants sur les sites,

– la configuration d'applications en interchangeant des composants dans une même application sans pour autant en changer radicalement le fonctionnement.

Les caractéristiques principales du langage *OLAN* sont les suivantes :

- la séparation entre l'interface et l'implantation du composant : le composant est visible à travers son interface. La même interface peut être utilisée pour manipuler des implantations différentes. L'interface d'un composant est définie principalement par deux types de services :

(3) Outil et LANgage pour applications coopératives. Ce langage est développé dans le cadre du projet SIRAC.

- les *services fournis* : ils permettent d’invoquer l’exécution d’un service implanté par le composant. Par exemple, si le composant implante une librairie de fonctions, l’invocation du *service fourni* (à l’aide de paramètres) permet d’exécuter une fonction donnée de la librairie.
- les *services requis* désignent les points de sortie du composant. Ces services sont en général fournis par d’autres composants de l’application.
- la composition : les composants peuvent avoir une structure hiérarchique. Ce qui veut dire que des composants peuvent en contenir d’autres. La Fig. 2.22 illustre deux types de composants :
 - des composants *primitifs* : il s’agit de composants qui encapsulent du code exécutable, ou une librairie de fonction,
 - des composants *composites* : ils sont construits à partir de sous-composants connectés,
- la communication : les connecteurs permettent de définir les caractéristiques des communications entre composants. Le langage *OLAN* permet d’instancier différemment les connecteurs en fonction des besoins exprimés par les composants membres de la communication,
- les *collections* : *OLAN* utilise le concept de *collection* pour définir un ensemble d’instances de composants issues toutes de plusieurs instantiations d’un composant offrant la même interface, mais des implantations différentes. Ce nous sera très utile pour décrire les applications coopératives dont le fonctionnement dans *CoopScan* respecte un schéma d’exécution identique.

Fig. 2.22 : Description des composants OLAN

Une définition plus complète du modèle et du langage sont fournis dans [Bellissard 95]. Nous nous limitons dans cette section à en rappeler les

principales caractéristiques, et à utiliser les fondements de base du langage pour décrire la plate-forme *CoopScan* et les modules qui s'y exécutent.

II.4.2.2 La description OLAN de CoopScan

Conformément aux choix de conception, une instance de *CoopScan* est exécutée sur chaque site participant.

Fig. 2.23 : Le composant "site participant" représentant une instance de CoopScan.

La Fig. 2.23 illustre la description *OLAN* d'une instance de *CoopScan*. Le composant "site participant" contient un *AgS* et une *collection* d'applications partagées, chacune d'elles représentés par un composant *primitif OLAN*. Les composants *primitifs* décrivent les schémas d'architecture adoptés respectivement pour intégrer une application X (*Appl X*), et une application ouverte (*Appl. ouverte*).

Chaque Composant *primitif* (*Appl_part1* et *Appl_part2*) renferme une instance d'une application existante à laquelle sont "greffés" les deux modules de

contrôle : *AgL* et *AgD*. Nous détaillons dans la suite le fonctionnement de chacun des composants *OLAN* et des modules qu'ils encapsulent.

Les liens entre les deux ronds noirs (à l'intérieur d'un composant) désignent des communications reçues par le composant englobant et transmises à l'un des sous-composants. Par contre, les liens entre ronds blancs désignent des communications initiées par un sous-composant et dirigées vers l'extérieur du composant englobant. Les connexions entre ronds blancs et ronds noirs schématisent des communications réalisées entre composants (dans le sens rond blanc vers rond noir).

La Fig. 2.23 illustre une description conceptuelle de l'architecture *CoopScan*. L'instantiation des connecteurs est réalisée lors de la génération de l'instance exécutable de *CoopScan*. Cette phase est liée à la plate-forme d'exécution sous-jacente, deux implantations de *CoopScan* sont issues de cette description *OLAN* : la première est réalisée sur une plate-forme Unix, la seconde sur une plate-forme à objets répartis (*OOBE*). Les connecteurs utilisés pour mettre en œuvre la communication entre les composants "site participant" sont instanciés dans l'implantation Unix par des communications *TCP/IP* moyennant des sockets de la famille *Unix* (entre ronds blancs et ronds noirs dans le même composant), et des sockets de la famille *Inet* pour les communications entre sites. Sur la plate-forme *OOBE*, les connecteurs sont instanciés à l'aide d'appels de méthodes sur des objets partagés.

II.4.3 L'agent session (*AgS*) :

II.4.3.1 Description

L'*AgS* est le module à travers lequel est représenté l'acteur dans le collectif. Ce module est par définition complètement générique. Il est totalement répliqué sur tous les sites du collectif. L'*AgS* est créé dès le lancement de l'application *CoopScan*, il permet à l'acteur de créer une nouvelle session ou de rejoindre une session existante.

L'*AgS* est chargé de l'exécution des services de gestion de groupes (*SGG*) et des politiques de droit de parole (*PDP*) dans la session. Les *AgS* d'une même session *CoopScan* communiquent entre eux par envoi de messages.

Afin d'assurer ses fonctions, l'*AgS* fournit une interface-utilisateur qui permet au participant d'accomplir des commandes (émettre une requête de connexion, lancer une application, demander le droit de parole, ...). L'interface

permet également au participant de percevoir tous les changements opérés sur l'espace de coordination.

L'AgS communique avec l'agent local (*AgL*) de son site pour lui transmettre d'éventuelles mises à jour des paramètres de l'espace de coordination, essentiellement pour l'informer des changements affectant les rôles des participants durant la session.

II.4.3.2 Architecture de l'AgS

La figure Fig. 2.24 illustre le schéma d'architecture *OLAN* du composant *AgS*. Nous y définissons un module central (*AgS_services*) qui réceptionne toutes les commandes de l'acteur c'est à dire les messages transmis par les autres *AgS*s ainsi que les messages transmis par l'*AgL* du même site.

Fig. 2.24 : La description OLAN du composant AgS.

L'*AgS_services* a la charge de décoder les messages reçus et de les aiguiller vers les modules appropriés de l'*AgS*, en effectuant des appels de fonctions.

Nous utilisons le concept de collection pour regrouper les politiques de gestion du droit de parole et les services de gestion de groupes. Cette organisation

permet de donner une structuration modulaire et, par conséquent, générique au composant *agent session*.

II.4.3.3 Fonctionnement

Le fonctionnement de l'AgS est décrit à l'aide d'un graphe d'état illustré par la Fig. 2.25. Le graphe schématise les deux états reflétant l'appartenance ou la non appartenance du participant à une session.

Fig. 2.25 : Graphe d'états décrivant le fonctionnement de l'AgS

- *état de veille* : c'est l'état à l'initialisation de l'AgS. Dans cet état, l'AgS ne fait pas encore partie d'une session. Par contre, dans cet état l'AgS peut scruter le réseau afin d'être informé en temps opportun du lancement d'une session.

L'AgS passe dans un *état actif* dès que l'utilisateur est connecté à une session ou que l'utilisateur crée une nouvelle session dont il sera le président. La transition *de l'état de veille à l'état actif* est le résultat de l'exécution du protocole de connexion fourni par le SGG de CoopScan.

- *état actif* : l'AgS passe dans l'état actif dès l'achèvement de la phase de connexion. Dans cet état, l'utilisateur a le rôle de participant ou de président. L'AgS peut donc assurer toutes les fonctions définies par le SGG et les PDPs.

Les deux états du graphe sont exclusifs. Cela traduit notre volonté de ne faire participer un utilisateur qu'à une session à la fois.

Nous donnons dans la suite le squelette d'exécution de l'AgS dans les deux états du graphe. Les termes en **gras** désignent les commandes de l'acteur effectuées à partir de l'interface-utilisateur ou le type de message émis par d'autres modules (*AgS* ou *AgL*). Les termes en *italique* désignent les types de messages envoyés. Des parenthèses sont utilisées pour encadrer des {commentaires}.

Fonctionnement en état de veille

Les commandes produites par le composant *AgS_UI*, désignant l'interface utilisateur de l'AgS, sont envoyées à travers *sendUserCmd* au module *AgS_services*.

```

TANT QUE État = veille ALORS
  SELON Action FAIRE
    CAS NEW_SESSION
      Créer une nouvelle session;
      État = actif;
    Cas JOIN_SESSION
      SGG(JOIN_SESSION_REQ,CurrentSGG);
      Attente de connexion;
    CAS SESSION_DIRECTORY
      S'abonner au système de notification afin d'être averti
      de la création des nouvelles sessions;
    CAS QUIT_COOPSCAN
      Exit;
  FIN SELON
FIN TANT QUE

```

Algorithmel. Exécution de l'AgS dans l'état de veille.

Fonctionnement en état actif

Dans cet état, l'AgS réagit à trois types d'événements : les commandes émises à partir de l'interface-utilisateur, les messages transmis par les autres AgS, et les messages transmis par l'AgL local.

L'algorithme suivant décrit le fonctionnement de l'AgS dans un état actif :

```

TANT QUE État = actif ALORS
  SELON Action FAIRE
    CAS Commande Utilisateur :
      SELON Commande FAIRE
        CAS ASK_FLOOR
          PDP(AskFloor, CurrentPDP); à l'AgS de l'ac-
          tuel détenteur du droit de parole;
        CAS GIVE_FLOOR
          PDP(GIVE_FLOOR, CuurentPDP)
        CAS QUIT_SESSION
          SGG(QUIT_SESSION, CurrentSGG);
          État = Veille;
      FIN SELON Commande

    CAS Message émis par un AgS
      SELON Message émis par AgS FAIRE
        CAS JOIN_SESSION_REQ
          SGG(JOIN_SESSION_REQ,CurrentSGG)
          {Lancer le protocole de connexion}
        CAS ASK_FLOOR
          PDP(ASK_FLOOR, CurrentPDP);
        CAS GIVE_FLOOR
          PDP(GIVE_FLOOR, CurrentPDP);

```

```

        {Prendre le jeton;
         Envoyer SET_FLOOR à l'AgL;}
        ...
    FIN SELON Message émis par AgS

    CAS Message émis par un AgL
      SELON Message émis par AgL
        CAS EXIT_APP
          SGG(EXIT_APP, CurrentSGG);
          ...
        FIN SELON Message émis par AgL
    FIN SELON Action
    FIN TANT QUE.

```

Algorithme2 décrivant l'exécution de l'AgS dans l'état *actif*.

L'algorithme précédant est partagé en trois parties, chacune étant réservée pour une source particulière d'événements.

Cependant, malgré leurs provenances diverses, nous regroupons tous les événements reçus par l'AgS en deux classes d'événements : les événements concernant les politiques de gestion du droit de parole (PDP), et les événements concernant les services de gestion de groupes. Chaque événement reçu provoque l'appel d'un des services PDP ou SGG représentés par des collections dans Fig. 2.24.

La collection PDP

Les politiques de gestion du droit de parole se présentent sous la forme d'une collection de composants PDP, chacun fournissant un service à partir duquel il est possible d'activer un ensemble de fonctions de traitement. Chaque fonction est appropriée au traitement d'un type d'événements. Les principaux événements concernant les PDPs sont les suivants:

- ASK_FLOOR : ce message désigne une requête d'acquisition du jeton,
- GIVE_FLOOR : ce message désigne la réception du jeton pour une application donnée,
- CHANGE_FLOOR : le détenteur du droit de parole pour une application donnée vient de changer,
- SNAP_FLOOR : ce message est émis par le président pour réquisitionner le jeton.
- CHANGE_PDP : ce message est exclusivement émis par le président lors du changement de la politique de gestion du droit de parole concernant une application du collecticiel.

Les paramètres d'appels du service PDP sont la politique de gestion du droit de parole adoptée dans une application (*CurrentPDP*), et le message reçu.

Toutes les informations concernant l'application et la provenance de l'événement, et, éventuellement sa destination, sont contenues dans l'événement lui-même.

Nous proposons dans *CoopScan* deux classes de politiques pour la gestion du droit de parole :

- des politiques explicites : en adoptant ce type de politique, le détenteur du droit de parole est explicitement désigné par un autre participant. Deux variantes de politiques explicites sont implantées, elles permettent respectivement au détenteur du jeton de désigner son successeur (*Désignation*), ou au président de donner la parole aux différents intervenants (*Modération*). Le président se réserve le droit de retirer le jeton à n'importe quel moment de la coopération.
- des politiques implicites : dans ce cas, le détenteur du droit de parole libère le jeton sans désigner son successeur. Le PDP se charge de désigner un nouveau détenteur du droit de parole. Les critères selon lesquels est affecté le jeton sont les suivants :
 - l'ordre de réception des requêtes sur le site de l'ancien détenteur du droit de parole. Nous utilisons l'ordre FIFO pour affecter le jeton au site dont la requête est la première parvenue sur le site de l'ancien détenteur du jeton,
 - priorité instaurée entre les sites en fonction de leur date d'adhésion au collectif.

A titre d'exemple, nous décrivons dans ce qui suit le comportement du composant FIFO, contenu dans la collection PDP (cf. description *OLAN* illustrée par la Fig. 2.24).

```

CAS Événements provenant d'un autre AgS
  CAS Événement = ASK_FLOOR
    SI je suis détenteur du droit de parole
    ALORS
      SI c'est la première requête émise par le site
      ALORS
        Ajouter la requête dans la file d'attente
      SINON
        Rediriger la demande vers l'AgS détenteur du
        droit de parole;
  CAS Événement = GIVE_FLOOR
    SGG(sendTo_AgL(SET_FLOOR)); {envoyer le jeton à
    l'AgL}

CAS Événement = CHANGE_FLOOR
  Appl[CHANGE_FLOOR.AppId].floorHolder =

```

```
CHANGE_FLOOR.newFloorHolder; {modifier le détenteur
du jeton pour l'application désignée par
Appl[CHANGE_FLOOR.AppId]}
```

```
CAS Événements provenant de l'interface-utilisateur
CAS Événement = GIVE_FLOOR
Si je suis détenteur du jeton
ALORS
  SGG(sendTo_AgL(GET_FLOOR),CurrentSGG)
  Dest = le premier site de la file des requêtes
  SGG(sendTo_AgS(GIVE_FLOOR, Dest));{envoyer le
jeton au nouveau détenteur du droit de parole}
  SGG(Diffuser(GIVE_FLOOR,AppPart));{diffuser le
changement du droit de parole à tous les
participants de l'application}
```

Algorithme3 décrivant le fonctionnement du composant FIFO.

Le composant FIFO fait appel au SGG pour réaliser les opérations de communication. Pour cela, il passe en paramètre la signature de l'opération de communication devant être assurée par le SGG. Par exemple, pour céder le jeton à un participant *Dest*, le composant FIFO exécute l'appel `SGG(sendTo_AgS (GIVE_FLOOR, Dest))`.

Les politiques de gestion de droit de parole sont complètement modulaires, au sens où chacune d'elle peut être décrite par un composant. De ce fait, de nouvelles politiques peuvent être facilement intégrées dans la collection PDP.

Toutes les politiques partagent les mêmes variables d'état décrivant l'espace de coordination (les applications partagées, les participants par application, la politique de droit de parole appliquée pour chaque application, ...). De même, les politiques représentées dans la collection PDP utilisent toutes les mêmes types d'événements.

Le changement dynamique des politiques de gestion du droit de parole

L'une des caractéristiques de *CoopScan* consiste à pouvoir changer, pendant l'exécution du collecticiel, la politique de gestion du droit du parole sur chaque application partagée du collecticiel.

Le changement dynamique de politique consiste à choisir dynamiquement un composant parmi les composants instanciés dans la collection PDP.

Le composant *AgS_services* gère la variable d'état `AppList[app_Id].CurrentPDP` qui désigne la politique PDP utilisée pour l'application *app_Id*. L'*AgS_services* mets à jour cette variable suite à la réception d'une commande effectuée par le détenteur du jeton, ou suite à la réception d'un message `CHANGE_PDP` émis par l'*AgS* président. Ensuite, il diffuse vers tous les autres *AgSs*

de la session, le message CHANGE_PDP. La réception du message CHANGE_PDP annule l'historique concernant la PDP précédente. Typiquement, si l'AgS gère une file de requêtes émises par des participants pour l'obtention du droit de parole, la file est remise à zéro dès la réception du message de changement de PDP.

L'ajout de nouvelles politiques de gestion du droit de parole

L'organisation modulaire adoptée pour structurer les PDPs dans CoopScan facilite l'ajout et le remplacement de politiques de gestion de droit de parole dans le collecticiel. La seule contrainte imposée consiste à adopter le même codage des événements sur lequel est fondé le fonctionnement de la collection PDP.

La collection SGG

Les services de gestion de groupes sont organisés de la même manière que les politiques de gestion du droit de parole. Les paramètres d'appel du service SGG sont l'événement reçu par l'AgS et le service de gestion de groupe utilisé pour la session (resp. pour l'application désignée). Le SGG fournit des fonctions de traitement pour les événements concernant la connexion, la déconnexion et la communication de l'AgS avec les autres modules de CoopScan. Les principaux messages pris en compte par le SGG sont :

- JOIN_SESSION_REQ : il s'agit d'une requête de connexion à la session,
- QUIT_SESSION : cet événement signale le départ d'un participant de la session,
- NEW_TOOL : cet événement est généré par l'interface-utilisateur de l'AgS quand un participant souhaite lancer une application partagée (l'application en question doit être dans le groupe d'application *applications disponibles*).
- JOIN_TOOL_REQ : cet événement désigne une requête de connexion à une application (l'application en question doit faire partie du groupe *applications en session*),
- QUIT_TOOL : un participant vient de quitter l'application partagée *QUIT_TOOL.Appld*.
- JOIN_NOTIFY : message diffusé par le président lors de la phase de connexion pour avertir les participants de l'arrivée d'un nouveau membre
- JOIN_SESSION_ACCEPT : ce message désigne une réponse favorable émise par le président au demandeur de la connexion,
- FREEZE_APP : ce message est émis par le président pour geler l'activité d'une application partagée,

– ADD_CONTEXT : ce message est émis par le président pour demander à un membre de mettre à jour son contexte. La nouvelle valeur du contexte est envoyée dans le message. Plusieurs émissions de ce message peuvent s'avérer nécessaires pour reconstituer la totalité du contexte du nouvel arrivant,

– END_CONTEXT : ce message désigne la fin de la phase de restitution du contexte.

– JOIN_VALID : c'est le message envoyé par le nouvel arrivant en signe de validation de la phase de connexion. A la réception de ce message, le site président dégèle la session.

Nous proposons actuellement deux services de gestion de groupes (*Free*, et *Sponsored*) qui diffèrent par la politique adoptée pour connecter un nouveau membre à la session *CoopScan* ou à l'une des applications s'il s'agit d'une requête de participation à une application donnée.

Le protocole de connexion adoptée dans le composant *Free* permet la connexion d'un participant sans l'intervention du président, alors que dans le SGG *Sponsored* la connexion est tributaire de l'acceptation du président.

Les services de gestion de groupe sont utilisés pour l'adhésion à la session ou à une application partagée de *CoopScan*.

Les deux politiques (*Free*, *Sponsored*) sont fondées sur un protocole de connexion dynamique à deux phases utilisant une technique de "gel"⁽⁴⁾. Le protocole est déclenché par le site président pour répondre à une requête JOIN_SESSION_REQ, et par le détenteur du jeton s'il s'agit d'une requête JOIN_TOOL_REQ. Pendant l'exécution du protocole, l'activité du groupe est *gelée* (c'est à dire aucune action n'est autorisée sur l'espace partagée), le temps que le nouvel arrivant soit complètement connecté à la session. La connexion complète est atteinte quand le nouvel arrivant détient une vue cohérente de l'espace partagé.

La déconnexion d'un site peut être le résultat d'une demande explicite de déconnexion formulée par un participant, ou d'une panne de site et/ou du réseau. Les SGGs fournis par *CoopScan* prennent en compte les deux cas de déconnexion d'une manière identique.

Un site est déclaré défaillant si, à l'expiration d'un délai de garde, toutes les émissions à destination de ce site échouent.

(4) Les services de gestion de groupes sont étudiés séparément dans le chapitre IV de ce mémoire.

L'arrivée d'un utilisateur dans *CoopScan* est traitée de façon asynchrone. Pour se connecter à une session *CoopScan*, les utilisateurs peuvent s'abonner à un gestionnaire de session (*session manager*) qui les averti, en temps réel, du lancement d'une nouvelle session.

Le changement dynamique du service de gestion de groupes

CoopScan permet de changer dynamiquement le service de gestion de groupe pendant le déroulement du travail coopératif. Le changement peut aussi bien concerner la session que l'une ou plusieurs des applications qui y sont en cours d'exécution.

- *Changement du SGG de la session* : dans la version actuelle, le président est l'unique participant habilité à changer le SGG de la session. Cette opération peut être faite à n'importe quel moment de la coopération. C'est à partir de l'interface utilisateur de l'AgS que l'opération de changement est lancée. Le traitement de cette opération est réalisé séquentiellement avec l'exécution du protocole de connexion d'un nouveau membre (i.e. le président ne peut pas lancer la commande de changement du SGG si le protocole de connexion est en cours d'exécution).
- *Changement du SGG d'une application partagée* : le détenteur du jeton est potentiellement le seul participant habilité à modifier le service de gestion de groupes d'une application partagée. Pour cela, le participant émet une commande à travers l'interface-utilisateur de l'AgS. L'AgS-*service* se charge de modifier la variable d'état *AppList[appId].CurrentSGG* en lui affectant la nouvelle valeur du SGG.

L'ajout de nouveaux services de gestion de groupes

Conformément à l'architecture de *CoopScan*, l'ajout d'un nouveau SGG consiste à instancier un nouveau composant dans la collection SGG. La seule contrainte imposée par l'ajout d'un nouveau SGG consiste à utiliser le même codage des événements que celui utilisé par le composant *AgS_services*.

II.4.4 L'agent local (*AgL*)

II.4.4.1 Description

La fonction principale de l'*AgL* est de contrôler l'exécution des applications partagées. D'un point de vue "développement", l'*AgL* constitue le module ajouté à toute application pour intercepter les commandes accomplies à partir de son interface afin d'en contrôler l'exécution.

L'AgL intercepte toutes les commandes effectuées par le participant avant leur exécution. Il vérifie ensuite le rôle de l'utilisateur en fonction des données qui lui sont communiquées par l'AgS. Dans le cas où le participant est le détenteur du droit de parole, l'action est exécutée localement puis transmise vers les autres sites, le cas échéant, la commande est ignorée et un retour via un message d'alerte est affiché à l'utilisateur.

Toutes les commandes effectuées sur l'espace de coopération sont mémorisées dans une structure de données dont l'utilité est primordiale lors de la phase de restitution du contexte à un nouvel arrivant.

Les applications partagées dans *CoopScan* sont regroupées dans la collection "applications partagées" selon le schéma illustré par Fig. 2.23.

II.4.4.2 Fonctionnement

Pour chaque application intégrée, l'*agent local* doit prévoir un module approprié dont la fonction est de "filtrer" toutes les commandes lancées à partir de l'interface-utilisateur de l'application. Dans le cas de l'intégration d'une application *ouverte* (cf. II.2.1), l'AgL s'abonne au service de notification (*UpCalls* en anglais) fourni par l'application pouvant ainsi être averti des actions pertinentes menées sur les données de l'application.

Lors de l'exécution du protocole de connexion à une application de la session, l'AgS commande la restitution du contexte de l'application partagée à son AgL. Pour cela, il lui envoie un message SET_CONTEXT lui demandant de restituer le contexte d'une application à un nouvel arrivant. L'AgL envoie alors vers le site désigné le contexte (liste des commandes) de l'application.

Nous donnons ci-dessous un extrait de l'algorithme régissant le fonctionnement de l'AgL dans une session *CoopScan*.

```

TANT QUE État = Vrai ALORS
  CAS Interception d'une commande
    SI FLOOR = Vrai {je suis détenteur du jeton}
      ALORS
        Exécuter la commande;
        Diffuser la commande à tous les participants
          connectés à l'application;
      SINON
        Avertir le participant en lui rappelant qu'il n'a
          pas le jeton;
    CAS Réception d'un événement de l'AgS
      CAS SET_FLOOR
        FLOOR = Vrai;{Je deviens le détenteur du jeton}
      CAS SET_CONTEXT
        ...

```

...
FIN TANT QUE

Algorithme4 décrivant le fonctionnement de l'AgL

Dans *CoopScan*, le module *agent local* est représenté par plusieurs modules *AgL*, chacun d'eux étant lié à une application partagée. Techniquement, cette solution est incontournable du moment que chaque application fournit une *API* et un module de notification qui lui sont spécifiques.

Fig. 2.26 : Schémas d'architecture des applications partagées dans CoopScan.

La Fig. 2.26 illustre les deux schémas d'architecture adoptés pour intégrer des *applications X* (s'exécutant sous le système de fenêtre *X-windows*) et des *applications ouvertes* (offrant une *API* et un service de notification *ad-hoc*). Pour les applications *X*, l'*AgL* et l'*AgD* sont regroupés dans un module intercalé entre l'application et le serveur *X*. L'*AgL* et l'*AgD* sont implantés par un processus qui gèrent les fenêtres allouées à l'application sur chaque site (i.e. ressources allouées par le serveur *X* s'exécutant sur le site). Le processus communique d'une part avec l'application *X*, et, d'autre part, avec le serveur *X* à l'aide de canaux *TCP/IP*.

Dans le composant intégrant une *application ouverte*, le module *AgL* est lié aux fonctions du service de notification de l'application.

Les fonctions de l'AgL qui sont communes à toutes les applications sont les suivantes :

- la fonction *CheckFloor()* qui permet de consulter la variable d'état FLOOR afin de vérifier les droits d'accès attribués au participant,
- la fonction *sendTo_AgD* qui permet de diffuser les commandes autorisées vers les autres sites.

Les autres fonctions de l'AgL sont spécifiques à l'application partagée. Parmi elles, on trouve les fonctions qui permettent la manipulation des données partagées de l'application pour mémoriser les actions menées par le participant (e.g. modifier la chaîne de caractère *chaine1* dans le paragraphe *paragraphe2* du document *document1*).

II.4.5 L'agent distant (AgD)

II.4.5.1 Description

Nous associons à chaque agent local de CoopScan un ensemble d'agents distants répartis sur les sites du collecticiel. Il s'agit d'émissaires capables de recevoir des messages émis par l'AgL. La fonction de l'AgD est de décoder le message, de reconstituer la commande, et ensuite de lancer l'exécution des fonctions appropriées de l'API.

II.4.5.2 Fonctionnement

L'AgD fournit un module de connexion spécifique à chaque application de sorte que les actions reçues puissent être reproduites. D'un point de vue implantation, l'AgD est constituée d'un ensemble de modules liés aux applications partagées.

Le schéma d'exécution de l'AgD est donné par l'algorithme suivant :

```
TANT QUE Application DANS {Application de travail}
ALORS
    Recevoir Message;
    Reconstituer Action;
    Lancer l'exécution de l'API concernée par
    l'action.
FIN TANT QUE
Algorithme d'exécution de l'AgD.
```

II.5 Conclusion

L'objectif principal d'une plate-forme pour la construction de collectif est de réduire le coût de développement d'une application coopérative. Dans ce chapitre nous avons présenté la plate-forme *CoopScan* qui atteint cet objectif en adoptant une approche fondée sur les deux concepts :

- la réutilisation d'applications existantes,
- la possibilité de fournir des services de gestion de groupes (SGG) et des politiques pour la gestion du droit de parole (PDP) applicables à plusieurs applications, ce qui permet de réaliser des scénarii coopératifs variés.

CoopScan est décrite à l'aide d'un modèle d'architecture générique mettant en œuvre des composants logiciels communicants ; l'*agent session*, l'*agent local* et l'*agent distant*.

Ce modèle d'architecture répond aux objectifs que nous nous sommes fixés au début de ce travail, concernant la généricité de la plate-forme, et ce, à différents niveaux :

– *la construction* : *CoopScan* fournit un ensemble de modules logiciels ré-utilisables pour plusieurs applications. Parmi ces entités, le composant *AgS* responsable de garantir l'accès aux applications partagées ainsi que l'adhésion des différents groupes constitués au cours de la session est ré-utilisable à 100 % pour différentes applications partagées. Par exemple, aucune modification n'est opérée sur ce module pour passer d'une session comprenant une application *X* (xedit) à une session comprenant l'éditeur *Thot* ou le navigateur *WWW*.

Le module *AgL* fournit trois types de fonctions : les fonctions de filtrage des actions menées à partir l'interface-utilisateur, les fonctions de journalisation des actions menées sur les données partagées de l'application, et les fonctions de communication avec les *AgD* implantés sur les autres sites, et avec l'*AgS* du site local. Parmi ces fonctions, seules celles concernant la journalisation sont spécifiques à l'application partagée. En tirant le bilan de notre expérimentation, l'*AgL* est fonctionnellement 75% ré-utilisable. Cependant, ce module est évidemment conceptuellement identique d'une application à l'autre.

Le module *AgD* est dans sa majorité dépendant de l'*API* fournie par l'application. Il assure principalement deux fonctions : la communication (réception et décodage des messages) et l'appel des fonctions de l'*API*. Seule la première fonction de l'*AgD* demeure inchangée d'une application à l'autre. Le volume de code ré-utilisable pour ce module n'exède pas 10%.

– *le fonctionnement* : CoopScan permet de faire coexister dans une même session de travail plusieurs applications partagées. CoopScan fournit également des bibliothèques de fonctions de contrôle regroupées en deux collections de composants fournissant chacune :

- des politiques de gestion du droit de parole (PDP) pour contrôler l'accès aux applications du collecticiel.
- des services de gestion de groupes (SGG) permettant l'adhésion des utilisateurs à une session CoopScan ou aux applications d'une session CoopScan selon plusieurs politiques de connexion. Ils permettent également la déconnexion de participants (d'une session ou d'une application) et la gestion des pannes dans le collecticiels.

Les PDP et les SGG sont dynamiquement interchangeable pendant l'exécution de *CoopScan*. Par ailleurs, le modèle d'architecture facilite l'ajout de nouvelles politiques d'une exécution à l'autre du collecticiel.

– *le support d'exécution* : l'architecture *CoopScan* a été implantée sur deux types de plate-formes différentes (Unix, Corba). Ces implantations sont décrites dans le chapitre V.

Chapitre II

Architectures logicielles pour collecticiels

II.1 Introduction	35
II.1.1 Organisation du chapitre.	36
II.2 Les approches architecturales pour la construction de collecticiels	37
II.2.1 L'intégration d'applications ouvertes.	37
II.2.1.1 Etude de cas pour l'intégration d'une application ouverte : l'éditeur de documents Thot [Quint 94]	38
II.2.1.2 Conclusion	45
II.2.2 Les approches d'intégration au niveau d'un système de fenêtres : La plate-forme X-Windows.	46
II.2.2.1 L'extension d'applications clientes.	47
II.2.2.2 L'approche pseudo-serveur.	47
II.2.2.3 Intégration d'applications dans l'environnement Macintosh	50
II.2.3 Schémas de mise en œuvre des architectures	64
II.2.3.1 Structuration modulaire du collecticiel	64
II.2.3.2 Schéma d'architecture centralisé	66
II.2.3.3 Schéma d'architecture totalement dupliquée	71
II.2.3.4 Schéma d'architectures hybrides	75
II.2.4 Discussion	76
II.3 CoopScan : Une architecture générique pour collecticiels	77
II.3.1 Objectifs	78
II.3.2 Cahier des charges.	79
II.3.3 Choix de conception	79
II.3.3.1 La construction d'applications partagées.	79
II.3.3.2 Architecture de mise en œuvre	80
II.3.3.3 Service de gestion de groupes et politiques de droit de parole	80
II.3.3.4 La session <i>CoopScan</i>	81

II.4	L'architecture de CoopScan	83
II.4.1	Description fonctionnelle de <i>CoopScan</i>	83
II.4.2	Modèle d'architecture générique	84
II.4.2.1	Le modèle OLAN	85
II.4.2.2	La description OLAN de CoopScan	87
II.4.3	L'agent session (<i>AgS</i>) :	88
II.4.3.1	Description	88
II.4.3.2	Architecture de l' <i>AgS</i>	89
II.4.3.3	Fonctionnement	90
II.4.4	L'agent local (<i>AgL</i>)	97
II.4.4.1	Description	97
II.4.4.2	Fonctionnement	98
II.4.5	L'agent distant (<i>AgD</i>)	100
II.4.5.1	Description	100
II.4.5.2	Fonctionnement	100
II.5	Conclusion	101

Chapitre III

Plates–formes systèmes pour la construction de collecticiels

III.1 Introduction

Les plates–formes réparties constituent un support informatique approprié à la conception et l’exécution des applications réparties en général et des collecticiels en particulier. En effet, il existe une adéquation significative entre, d’une part, les fonctions requises par un collecticiel (partage d’information, communication, interaction, ...) et d’autre part, les services communément offerts par les plates–formes réparties.

Nous nous intéressons dans cette section à l’utilisation des plates–formes réparties en tant que support logiciel pour déployer et exécuter des collecticiels synchrones. Ce chapitre comporte une étude des plates–formes à objets répartis ainsi qu’une étude concernant l’infrastructure World Wide Web en vue de son utilisation pour supporter du travail coopératif.

III.2 Plates–formes à objets répartis

Nous nous intéressons dans cette section à l’utilisation des plates–formes à base d’objets répartis en tant que support logiciel pour exécuter des collecticiels synchrones.

CORBA, développée par l’OMG, représente un standard pour les modèles d’architectures supportant des exécutions d’objets répartis.

Le principal objectif de l’OMG est de fournir une vue unifiée et simplifiée d’un système distribué hétérogène. Cet objectif consiste à :

- permettre la ré–utilisation de composants logiciels,
- garantir la portabilité et l’interopérabilité,

Le modèle d'architecture CORBA (illustré dans Fig. 3.1) est fondé sur un mécanisme appelé ORB (Object Request Broker) dont la fonction est de permettre un accès uniformisé à trois types d'objets :

- des objets spécifiques à l'application,
- des objets qui implantent des services,
- des objets communs qui implantent des utilitaires.

Fig. 3.1 : Modèle d'architecture CORBA

D'autres architectures sont utilisées dans le monde objet, notamment OODCE, une plate-forme C++ de l'environnement d'exécution distribué (DCE) de l'OSF [Dilley 95], et OLE qui est la technologie développée par Microsoft pour l'intégration d'objets distribués.

Caractéristiques des environnements d'exécution distribués :

- Transparence vis à vis des protocoles de communication dans le cas où l'application fait intervenir des réseaux de communication hétérogènes. Dans ce cas, l'ensemble de protocole qui relie les différents modules de l'application peut utiliser une famille de protocoles LAN ou WAN parmi : TCP/IP, X.25, ou Novell IPX/SPX. Tous ces protocoles supportent des communications point-à-point. Cependant, ils présentent certaines particularités qui les distinguent les uns des autres et qui peuvent entraver la portabilité et l'interopérabilité du logiciel. Pour remédier à ces problèmes, les environnements CORBA cachent à l'application la manipulation des piles de protocoles utilisées pour interconnecter les différents modules.

- **Transparence vis à vis des supports d'exécution** ; les utilisateurs d'une application coopérative sont connectés à partir de leur propre station de travail. Ces machines ne sont pas nécessairement identiques.

Les environnements obeissant au modèle d'architecture CORBA fournissent des outils du type IDL (Interface Définition Langage) et des compilateurs qui leur sont associés et qui se chargent de réaliser des opérations d'empaquetage des paramètres d'appels des méthodes entre des objets s'exécutant sur des machines différentes. De ce fait, un objet codé en C++ ou SMALLTALK, peut donc être modifié au moment de l'assemblage dans un langage objet différent de son langage d'implantation.

Par ailleurs, un effort de standardisation est en cours. Il se traduit par l'émergence de deux normes mondiales basées sur deux modèles d'objets différents : CORBA et OLE/COM de Microsoft. Il sera possible de faire coopérer des objets issus de chacun des deux modèles.

Apports d'une plate-forme à objets répartis pour la construction d'applications coopératives :

- **La structuration de l'application** : la modélisation objet permet une meilleure structuration de l'application. Elle permet par ailleurs une séparation entre la description des modules et leur implantation. Cette caractéristique peut s'avérer très utile pour le déploiement d'une application sur des environnements d'exécution hétérogènes.
- **La communication** : Les environnements CORBA permettent la réutilisation de modules logiciels, ce qui facilite le développement de nouvelles applications. Cette propriété de CORBA favorise la construction de collecticiels à partir d'applications existantes. Ceci décharge le concepteur de l'application des aspects de communication et lui permet de se focaliser sur les aspects spécifiques de l'application.

Dans le cas d'une plate-forme Unix-TCP/IP, le développeur de collecticiels prend à sa charge la mise en œuvre de la communication. Au niveau de chaque site, le développeur doit maintenir à jour l'ensemble des adresses des processus du collecticiel.

- **La répartition** : Les objets sont manipulés indépendamment de leur localisation physique. Cet avantage inclut la résolution des problèmes d'hétérogénéité des machines.

L'utilisation d'une plate-forme à objets induit implicitement l'utilisation d'une approche différente de celle communément utilisée dans les systèmes classiques (par exemple, Unix) lors du choix de l'architecture d'une application coopérative. En effet, la dualité entre architectures centralisées et dupliquées dans les systèmes classiques peut s'avérer moins importante dans un environnement CORBA.

III.3 L'infrastructure World Wide Web

Le développement d'une infrastructure répartie permettant la mise en œuvre d'un espace d'information commun et sa manipulation à travers une représentation uniforme des données constitue un moyen idéal pour supporter un travail de groupe à grande échelle. Particulièrement, pour supporter les tâches dans lesquelles sont impliquées des utilisateurs appartenant à des organisations différentes. Les membres de tels groupes n'utilisent nécessairement pas les mêmes moyens informatiques. Ils sont souvent confrontés à des problèmes d'hétérogénéité de machines, de protocoles de communication, d'outils logiciels...

Malgré l'essor que connaît la recherche dans le domaine du TCAO depuis une dizaine d'années, les applications *emails*, *ftp*, *talk* constituent pratiquement le seul état de l'art pour les applications coopératives à grande échelle, notamment celles offrant des solutions aux problèmes d'hétérogénéité, des produits industriels tels que *Lotus Notes* sont la preuve tangible de ce succès. Bien que ces applications facilitent l'échange de données entre utilisateurs, elles ne fournissent qu'un faible support pour le partage (modes d'interactions, droits d'accès...). Par exemple, l'application *talk* bien que largement diffusée, ne permet qu'un échange de données textuelles. L'application *email*, ne laisse pas le choix à l'utilisateur quant au choix du mode d'interaction, il y est définitivement asynchrone.

Les plates-formes CORBA apportent des solutions concrètes aux limitations que présentent les systèmes classiques pour faire communiquer des outils logiciels développés sur des plates-formes différentes (e.g. UNIX, Windows). Toutefois, les solutions apportées par CORBA ne concernent pas les aspects d'interface-utilisateur pour fournir une présentation commune des informations contenues dans l'espace partagé. La mise en œuvre de ces aspects demeure à la charge du développeur du collecticiel.

L'utilisation d'une plate-forme commune offrant des solutions générales aux problèmes d'hétérogénéité et de représentation des informations dans un espace partagé semble indispensable pour contribuer à une large diffusion des applications coopératives jusqu'à là utilisées uniquement en tant qu'outils "de luxe" dans des environnements restreints et souvent spécifiques (e.g. réseaux locaux, machines homogènes).

Patterson dans [Patterson 91] et Trevor dans [Trevor 94] évoquent l'importance d'une telle approche pour décharger les développeurs d'applications des problèmes d'intégration et leur permettre de se focaliser sur les problèmes spécifiques à la coopération.

C'est dans cette optique que notre étude concernant les architectures pour collecticiels synchrones s'intéresse au World Wide Web (WWW) en tant qu'infrastructure largement diffusées favorisant la construction d'applications pour le TCAO.

III.3.1 Définition et caractéristique de l'infrastructure WWW

WWW est une ressource introduite en 1992 par le CERN (Centre Européen de Recherche Nucléaire). Il s'agit d'un ensemble de serveurs Internet offrant des documents sous forme d'hypertextes. Ces documents permettent de naviguer sur le réseau Internet plus facilement et d'obtenir des informations qui proviennent d'autres ressources comme Gopher⁽¹⁾, Usenet, Wais⁽²⁾, Ftp, Telnet, etc. Les serveurs peuvent être accédés à partir d'applications clientes capables de visualiser les informations récupérées.

WWW (ou W3) est un système d'information multimédia coopératif, hétérogène et distribué qui peut être défini par :

- Un ensemble de concepts : WWW est un espace "décousu" conforme à un modèle d'architecture client/serveur dans lequel toute information provenant de n'importe quelle source peut être facilement accédée. Les concepts introduits par W3 sont :

Navigation universelle : Si une information est disponible sur le réseau, elle est accessible à partir de n'importe quelle machine du réseau. L'utilisateur n'a qu'à utiliser un programme approprié pour y accéder.

(1) Ressource logicielle qui permet de naviguer Internet à l'aide de menus arborescents. Le Gopher a été réalisé à l'université de Minnesota en 1991.

(2) Wide Area Information Server est un moteur de recherche et d'indexation qui fonctionne avec le protocole TCP/IP. C'est un outil qui permet de gérer des dépôts de textes et de données.

Hypertexte : W3 utilise une présentation hypertexte des informations accessibles, celles-ci pouvant être textuelles, graphiques, animées ou sonores.

Recherche : W3 permet une recherche par index sur les documents hypertextes.

Modèle client/serveur : Les communications clients/serveurs ne sont soumises à aucun contrôle centralisé. Les utilisateurs peuvent librement proposer des informations sur des serveurs W3.

Négociation de formats : La communication entre client et serveur est assurée par un protocole qui permet la négociation des formats de données échangées. A l'émission de la requête, le client signifie au serveur l'ensemble des formats qu'il peut décoder, en fonction de ces informations, le serveur transmet les données sous une forme appropriée.

- Un ensemble d'outils et de protocoles uniformisés.

URL (*Uniform Resource Location*) est un format de représentation uniforme d'adresses. W3 utilise les URLs pour accéder des serveurs W3 et résoudre les liens dans les documents hypertextes accédés. Les URLs permettent de localiser des informations indépendamment des protocoles d'accès utilisés pour y accéder. Cette approche permet d'atteindre des données stockées dans des archives Ftp, des serveurs Wais ou Gopher.

HTTP (*HyperText Transfer Protocol*) : Le Web utilise une variété de protocoles (e.g. Ftp, Wais, NNTP). Toutefois, il propose son propre protocole HTTP qui est extensible et, surtout, met en œuvre une négociation de format entre clients et serveur.

HTML (*HyperText Markup Language*) : Il s'agit du format de représentation le plus répandu chez les clients W3 de diverses plates-formes. Les documents HTML sont conformes à une description SGML augmentée par des liens.

Caractéristiques de WWW

Les caractéristiques de cette infrastructure sont énumérées de la manière suivante dans [Bentley 95]:

- transparence vis à vis de la plate-forme d'exécution, du système d'exploitation et du réseau de communication. En effet, l'utilisation d'un *browser* W3 à partir d'une plate-forme d'exécution donnée (Windows,

Unix/X–Windows, Macintosh) permet d'accéder des informations présentes sur des serveurs W3 indépendamment des réseaux de communication reliant le *browser* aux différents serveurs, et des plates–formes hôtes des serveurs,

- intégration facile des outils logiciels usuels figurant dans l'environnement de travail de l'utilisateur.
- interfaces utilisateur simples, disponibles sur différentes plates–formes (e.g. Netscape, Mosaic, Links...). La plupart des *browsers* W3 offrent quasiment la même présentation des informations,
- présence d'une API facilitant l'extension de certaines applications clientes. Par exemple, les CCI⁽³⁾ fournis par NCSA Mosaic permettent d'enrichir le client W3 Mosaic en lui greffant des services à travers les CCI. Le *browser* Netscape pour Macintosh offre une API qui lui permet d'interagir avec d'autres applications Macintosh en utilisant les événements AppleEvents (cf. Chapitre 3).
- rapidité de prototypage vu l'ensemble des facilités fournies par W3 : un modèle d'adressage uniforme des informations, un langage de description des données reconnu par plusieurs plates–formes, et un protocole de communication largement diffusé.

III.3.2 WWW, une infrastructure pour construire des collecticiels

Les avantages que nous venons d'énumérer nous incitent à étendre notre étude des architectures pour collecticiels à l'infrastructure W3. Plus précisément, il s'agit pour nous d'étudier les approches adoptées pour construire des collecticiels synchrones, et de les situer dans le cadre de notre étude initiale concernant les approches d'intégration d'applications conformes à un modèle d'architecture client/serveur. Afin d'exploiter au mieux les possibilités offertes par le Web, nous maintenons les concepts fondamentaux introduits par W3, en l'occurrence le modèle d'architecture client/serveur, prévoyant pour chaque utilisateur un client W3 . Ces hypothèses nous placent directement dans le contexte de l'intégration d'applications ouvertes.

(3) CCI : Common Client Interface

III.3.3 Approches de construction de collecticiels sur WWW

Différentes approches sont adoptées pour construire des applications coopératives utilisant l'infrastructure WWW. Elles peuvent être regroupées en trois catégories, en fonction des extensions qu'elles engendrent sur le standard W3 :

1) Solution conforme au standard W3 (Purely W3-based solution)

Cette solution utilise des clients et des serveurs W3 standards, conformes aux descriptions HTML et au protocole HTTP. La coopération est mise en place en rajoutant de nouvelles fonctions supplémentaires à l'ensemble des fonctions classiques d'un serveur W3. Les fonctions rajoutées sont spécifiques à la coopération. Elles sont mises en œuvre au moyen des interfaces CGI⁽⁴⁾ accessibles par le serveur.

Généralement, les fonctions fournies par un serveur W3 permettent l'accès à des documents du serveur, la résolution d'URL à partir des liens présents dans des documents, etc.

Le principe de fonctionnement des CGI est simple. Ce mécanisme permet d'activer des procédures de traitement à partir de descriptions HTML. L'exécution de ces procédures est déclenchée à partir de clients W3 distants sur le site du serveur.

Dans le cas de la distribution NCSA du serveur *HTTPd*, un répertoire spécial appelé *cgi-bin*, est dédié au stockage des programmes CGI. Ces programmes peuvent être écrits dans l'un des langages de programmation suivants :

- *C/C++*
- *Fortran*
- *PERL*
- *TCL*
- *Tous les Shell Unix*
- *Visual Basic*
- *AppleScript*

L'exemple illustré par Fig. 3.2 montre l'exécution d'un programme CGI de nom *exemple*, écrit dans le langage *Shell sh*. Le programme *exemple* permet l'affichage de quelques informations relatives à la session en cours. L'exécution du CGI *exemple* est le résultat de la résolution de l'URL :

(4) CGI : Common Gateway Interface, <http://www.org.com/WWW/CGI/>

<http://pukapuka.inrialpes.fr/cgi-bin/exemple>

Fig. 3.2 : Exemple d'utilisation des CGI dans une infrastructure WWW

Quelques applications coopératives sont développées selon cette approche, en utilisant uniquement les CGI. Parmi elles nous pouvons citer le système coopératif décrit dans [Frivold 94] et le système BSCW décrit dans [Bentley 95].

Le principal avantage de cette approche est qu'elle n'engendre aucune modification sur l'infrastructure WWW standard. Par contre, en étant très proche de l'architecture W3, elle est tributaire du mode de communication établi par le protocole HTTP entre le client et le serveur. Selon ce mode, il n'est pas possible de réaliser des interactions synchrones (temps-réels) entre différents clients partageant les mêmes pages.

2) Clients W3 personnalisés (Customized clients)

Cette solution est fondée sur la modification du client W3. Souvent des modifications sont nécessaires afin de supporter des *tags* HTML non standardisés, ou des APIs spécifiques à des clients W3. Cependant, aucune modification n'est faite pour l'utilisation du protocole HTTP. Les systèmes *Ubique's Virtual Places* [Ubique] et [Worlds] sont inspirés de cette approche de construction.

Cette approche apporte des extensions incontournables pour établir des communications directes entre différents clients W3. Cependant, l'extension de clients W3 peut s'avérer très coûteuse voire impossible pour certains tels que Netscape, qui n'offre aucune API.

3) **Serveurs W3 personnalisés (Customized servers)**

Cette solution est semblable à la précédente, mais elle nécessite un serveur W3 spécifique offrant des services autres que ceux offerts par les CGI standards. Cela peut affecter la portabilité du serveur. InterNotes [Worlds] est une infrastructure qui adopte une solution inspirée de ce principe.

La classification présentée précédemment traduit le degré de divergence entre un système coopératif construit sur un support W3 et le support W3 initial. Ce degré de divergence augmente en partant de la solution 1 vers la solution 3. Ce résultat est tout à fait prévisible ; en revanche dans une solution purement W3 (solution 1), le développeur exploite les services standards offerts par W3, sans modifier ni le client ni le serveur.

À l'issue de cette classification, notre objectif est d'étudier les solutions apportées par l'infrastructure W3 afin de pouvoir l'utiliser pour construire des collecticiels tout en tenant compte des aspects : architecture, et *besoins des applications coopératives (e.g. mode d'interaction synchrone/asynchrone)* [Rodden 92]. Nous essayons également de trouver, si elles existent, les restrictions imposées par un modèle d'architecture dans le choix de l'une des approches W3 ?

III.3.4 Exemples d'applications

BSCW Shared Workspace System (*Approche 1*)

L'application BSCW [Bentley 95] est une application coopérative qui permet la gestion d'une base de documents partagés à travers le Web. Le système est défini de la manière suivante : plusieurs serveurs gèrent un ensemble d'espaces de travail, accessibles à partir de clients W3 standards non modifiés. Chaque espace de travail contient un ensemble d'objets partagés. Les utilisateurs de

l'application (clients W3) peuvent consulter, modifier, et demander des informations supplémentaires concernant les objets contenus dans les différents espaces de travail. Les objets sont des documents, des liens (vers des pages W3 et vers d'autres espaces de travail), des groupes et des membres.

Chaque serveur BSCW maintient une liste d'index désignant les espaces de travail qu'il gère. Les utilisateurs accèdent la liste d'index en remplissant un formulaire dans lequel ils donnent un nom d'utilisateur (*user_name*) et un mot de passe (*password*).

Les utilisateurs de BSCW sont organisés en groupes. Chaque groupe est autorisé à accéder un ensemble d'espaces de travail. L'adhésion d'un nouveau membre est assurée par l'un des membres du groupe. La vue globale de l'ensemble des espaces de travail est identique pour tous les utilisateurs. Par contre, dans un espace de travail, l'utilisateur a le choix de la vue de l'espace.

Particularités de la mise en œuvre

Le serveur BSCW est une extension du NCSA httpd W3. Aucune modification n'est faite sur le client standard W3. Toutefois, les clients sont assistés par des "applications helpers", modules spécifiques à la plate-forme d'exécution du client (cf. Fig. 3.3). La fonction de ces modules est de permettre le transfert de documents du client vers le serveur BSCW. Les documents qui constituent l'espace de travail sont stockés sur une plate-forme UNIX. Le serveur gère toutes les informations sur ces documents : propriétaires, date de création, etc.

D'un point de vue mise en œuvre, l'architecture de BSCW est complètement centralisée. Tous les traitements sont accomplis à partir des CGI localisés sur le site d'exécution du serveur.

Fig. 3.3 : Architecture du système BSCW.

PUT est le nom de la procédure qui était utilisée dans les premières versions du système pour déposer un document modifié par un client sur le site du serveur. Cette procédure a été supprimée dans la nouvelle version de BCSW. L'opération est actuellement accomplie moyennant l'utilisation de la requête POST en soumettant une *form* HTML.

Les modules "applications helpers" sont activés à partir d'un client convenablement configuré, quand celui-ci reçoit un message du type MIME-type⁽⁵⁾ envoyé par le serveur. Ils fournissent sur chaque plate-forme client, un *browser* de documents appropriés pouvant dialoguer avec le système de fichier local.

Les Aspects coopératifs

A part les problèmes d'extension de l'infrastructure W3 initiale, le système BSCW ne répond pas complètement à certains besoins essentiels de coopération, ceux de fournir :

- un support évolué pour l'édition et le contrôle d'accès,
- un mode d'interaction et de notification permettant aux utilisateurs de communiquer en temps réel en échangeant n'importe quel type de données (texte, paquets audio, trames vidéo, images).

(5) MIME-type : Multipurpose Internet Mail Extension.

Les limitations affichées par le système BSCW pour supporter un mode d'interaction synchrone sont imposées par l'infrastructure W3.

InterNotes [InterNotes]

Lotus InterNotes Web Publisher est un serveur Notes qui permet à des utilisateurs de mettre des documents initialement Notes sur le Web. L'approche adoptée pour cela consiste à convertir des formes Notes, des documents, des vues et des bases de données en documents HTML.

InterNotes Web Publisher permet l'utilisation de l'environnement d'édition coopérative de Notes ainsi que de ses fonctions Workflow.

Fig. 3.4 : Architecture générale de InterNotes

Particularité de fonctionnement

Le fonctionnement de l'application est donné par le schéma de la Fig. 3.4. Quand un utilisateur demande l'affichage d'une information de la base Notes, le système réagit de la manière suivante :

- le module WEBPUD convertit les informations de la base en documents HTML et les place dans le repertoire du serveur W3
- le browser Web émet une requête de chargement d'une page HTML (GET),
- le serveur Web répond à la requête du client en transférant la page demandée,
- le browser affiche la page.

Quand un utilisateur soumet une forme dans une requête POST :

- le serveur transmet la requête au programme CGI de nom *inotes*,
- le CGI *inotes* transmet à son tour l'information au module interactif INOTES
- enfin, le module Inotes crée un nouveau document Notes dans la base.

Aspects coopératifs

Inotes apporte une solution pour la création et la manipulation de document Notes à partir de clients W3 standard. L'extension du serveur pour la prise en compte des descriptions Notes place cette solution dans la classe des applications construites selon l'approche *extension de serveurs*.

Pour ce qui est des aspects coopératifs, le partage d'une même source d'information est à l'origine, par défaut, d'une coopération asynchrone. Par ailleurs, la jonction entre l'environnement Notes et W3, par l'intermédiaire de InterNotes, permet l'utilisation de l'environnement coopératif de Lotus Notes [Lotus Notes].

III.3.5 Bilan et discussion des solutions apportées par WWW

L'essor indéniable que connaît le World Wide Web nous impose incontestablement une réflexion au sujet de l'utilisation de W3 comme support de mise en œuvre d'applications coopératives. W3 apporte des solutions à un ensemble de problèmes "historiques" de compatibilité et d'hétérogénéité, souvent rencontrés dans le monde informatique.

Les problèmes rencontrés lors de la mise en œuvre d'une application distribuée sont, pour la plupart, dus à la diversité des supports informatiques mis en

jeu. Dans le cas des applications coopératives, ces problèmes apparaissent à plusieurs niveaux de l'application :

- dans le choix de l'**interface utilisateur**,
- pour la représentation de l'**information partagée**,
- et dans le choix des **protocoles de communication**.

En réponse à ces problèmes, l'infrastructure W3 introduit trois concepts de base [Berners-Lee 96] : un espace d'adressage de ressources uniforme (URL⁽⁶⁾), un protocole de transfert de documents HyperText (HTTP⁽⁷⁾) et un langage de définition de documents (HTML⁽⁸⁾). Ces concepts permettent respectivement de fournir un modèle d'adressage uniforme des données indépendamment de leur localisation physique sur le réseau, un protocole de communication capable de transporter des données multimédia, et enfin un langage de description et de présentation de documents qui fournit une interface pour le traitement et la manipulation des données.

En dépit de ces avantages, l'approche qui consiste à utiliser le Web pour coopérer laisse transpar tre certaines insuffisances, particuli rement dans le cas des t ches synchrones n cessitant des interactions fortes entre les utilisateurs. L'architecture client/serveur d'une part, et, d'autre part, le fonctionnement du protocole HTTP, privil gient un mode de fonctionnement asynchrone. Les interactions y sont unidirectionnelles, impliquant souvent des donn es fig es (quelquefois calcul es dynamiquement).

Limitations du protocole HTTP

Conform ment   la sp cification du protocole HTTP/1.0 [Berners-Lee 96], les membres d'une connexion HTTP constituent une paire *client/serveur*, le serveur pouvant  tre le serveur d'origine, un *proxy*, un *gateway*, ou un tunnel. G n ralement, le r le du client consiste   demander une ressource dont l'identification est donn e par une URL, alors que le serveur repr sente le site contenant la ressource ou responsable de son transfert vers le client. Ce mod le d'architecture peut  tre recommand  pour mettre en  uvre des fonctions de recherche de ressources dans une base de donn es. Par contre, il semble moins convenir   nos besoins :

(6) Uniform Ressource Locators.

(7) HyperText Transfer protocol.

(8) HyperText Markup Language.

- Communications synchrones entre les clients ;
- HTTP est *stateless* (sans états), alors que le mode d'interactions synchrones nécessite l'utilisation d'un protocole "à états", plus adapté pour supporter un service de notification.
- La communication entre un client et un serveur est nécessairement déclenchée par l'initiative du client (résolution d'une URL). En aucun cas, le serveur est l'initiateur de la communication. Ceci rend immédiatement une communication client-clients temps-réel, même par l'intermédiaire du serveur, quasiment irréalisable.

De ce fait, la communication entre différents clients W3 ne pourrait se faire que par le biais de serveurs communs, selon un mode producteur/consommateur. Dans ce cas, la coopération (partages de données) est mise en œuvre au moyen des *Fill-Out Forms* fournis par HTML. Le schéma de fonctionnement est alors fondé sur l'activation de scripts CGI, préalablement placés sur le site serveur, en utilisant la méthode POST du protocole HTTP.

Par exemple, dans le système BSCW [Bentley 95], le travail coopératif est réalisé moyennant deux types de pages HTML. Alors que le premier type de pages sert à stocker et à présenter de l'information produite par la coopération (e.g. éditorial d'un journal), le deuxième type de pages permet de manipuler des structures de contrôle. Celles-ci contiennent des informations concernant les participants, les espaces de travail auxquels ils ont accès... Typiquement, un nouvel arrivant qui désire se joindre à une tâche coopérative, passe systématiquement par une page "d'admission". Dans cette page, l'utilisateur doit préciser certaines informations le concernant, *user_name*, *password*. Les informations sont utilisées pour la mise en œuvre d'une procédure d'authentification et comme moyen d'identification de l'utilisateur au sein du groupe de participants. La connexion consiste à stocker les informations saisies à travers la page d'admission au sein des structures de contrôle. Une fois l'opération achevée, le serveur W3 ne dispose d'aucun moyen pour avertir les clients de l'arrivée d'un nouveau membre. C'est uniquement au moment de l'établissement de sa prochaine connexion HTTP avec le serveur qu'un client est susceptible d'accéder à l'information.

Fig. 3.5 : Les Extensions de l'architecture W3 pour supporter la coopération

Face à ces limitations, l'établissement d'une communication directe, bidirectionnelle entre les clients W3, nous semble une solution appropriée qui permettrait de mettre en œuvre un mode d'interaction conforme aux exigences d'une coopération synchrone.

Dès lors, l'architecture standard W3 passerait d'un modèle purement client/serveur à un modèle d'architecture hybride, associant les concepts de fonctionnement du modèle client/serveur et les concepts d'un modèle de fonctionnement "client/client" Fig. 3.5.

Dans la Fig. 3.5, nous employons l'appellation "client/client" pour désigner la communication entre deux clients (clients potentiels d'un même serveur HTTP).

Cette approche consiste à doter les clients d'un comportement similaire à celui d'un serveur.

Nous proposons dans le chapitre V une mise en œuvre de cette approche dans la plate-forme CoopScan. Les clients W3 intégrés au sein de l'architecture CoopScan sont assimilés à des applications ouvertes dont l'exécution est contrôlée à partir de modules génériques fournis par CoopScan.

L'extension de l'infrastructure W3 par des communications synchrones entre clients place notre solution dans la catégorie des solutions *clients W3 personnalisés*.

Notre argumentation concernant ce choix s'appuie sur les critères suivants :

- Étendre les fonctions du client pour supporter une communication synchrone fournit une solution complémentaire au fonctionnement des solutions utilisant une communication asynchrone.
- Une telle architecture offre tous les avantages d'une architecture répliquée.

III.4 Conclusion

Dans ce chapitre, nous avons étudié les plates-formes à objets répartis et l'infrastructure World Wide Web en tant que support logiciel pour construire des collecticiels synchrones. Nous avons présenté dans la section 3 l'architecture standard W3, et étudié les schémas d'extension opérés sur cette architecture pour qu'elle puisse répondre aux besoins des collecticiels en général et à ceux des collecticiels synchrones en particulier. D'un point de vue utilisateur, ces besoins se traduisent par des fonctions logicielles qui permettent essentiellement le partage des données et l'interaction entre utilisateurs selon un mode temps-réel. D'un point de vue développeur, les besoins à satisfaire sont la genericité de la solution, et le faible coût de développement.

Fig. 3.6 : Classification des architectures pour collecticiels synchrones dans un espace tridimensionnel

En conclusion, notre étude des architectures se schématise, comme le montre la Fig. 3.6, par un espace tridimensionnel dont les axes désignent respectivement : les plates-formes systèmes, la répartition des modules du collecticiel et les schémas d'intégration d'applications existantes.

Les exemples cités dans cette illustration sont :

- BSCW présenté plus haut,
- *CoopScan* est notre plate-forme expérimentale décrite dans le chapitre II. Cette plate-forme est construite à partir d'un modèle d'architecture générique permettant aussi bien l'intégration d'applications au niveau du

système de fenêtres *X-Windows*, que l'intégration d'applications *ouvertes*.

- *CoopGrail* : est une application partagée faisant partie de la plate-forme *CoopScan*. *CoopGrail* est navigateur WWW coopératif permettant à plusieurs utilisateurs de participer à des visites "guidées" sur le Web.
- *CoopScan_Orb* désigne une implantation de *CoopScan* sur Orbix. Ce travail à été réalisé en collaboration avec le CNET (équipe ARCADE).
- *CoopScan_Unix* désigne désigne l'implantation de *CoopScan* sur une plate-forme Unix (chapitre V).
- *CoopScan_OODE* désigne l'implantation de *CoopScan* sur une plate-forme à objets répartis (*OODE* [Bellissard 96]).

Chapitre III

Plates-formes systèmes pour la construction de collecticiels

III.1 Introduction	99
III.2 Plates-formes à objets répartis	99
III.3 L'infrastructure World Wide Web.	102
III.3.1 Définition et caractéristique de l'infrastructure WWW	103
III.3.2 WWW, une infrastructure pour construire des collecticiels.	105
III.3.3 Approches de construction de collecticiels sur WWW	106
III.3.4 Exemples d'applications	108
III.3.5 Bilan et discussion des solutions apportées par WWW	112
III.4 Conclusion	116

Chapitre IV

Gestion de groupes dans les collecticiels synchrones

IV.1 Introduction

La première partie de cette thèse ayant été dédiée à l'étude des architectures pour la construction de collecticiels, nous étudions à présent les services de gestion de groupes et plus précisément les protocoles de connexion-déconnexion dynamiques de sites.

Le problème

Un collecticiel synchrone peut être assimilé à un lieu de réunion virtuelle. Des utilisateurs distribués peuvent s'y donner rendez-vous et, probablement, se "retrouver" pour y travailler ensemble (par exemple, pour rédiger un document commun, ou pour se concerter afin de prendre une décision commune). Toutefois, la disponibilité des uns et des autres pour se retrouver au même endroit et au même moment est incertaine. L'arrivée des participants s'effectue souvent après le démarrage d'une tâche coopérative.

C'est pourquoi un collecticiel synchrone doit fournir un service de connexion flexible permettant à des retardataires de rejoindre un travail coopératif en cours et d'être informés de ce qu'ils auront pu manquer depuis son début.

Par ailleurs, un collecticiel synchrone est avant tout une application répartie. Les utilisateurs y ont une vue cohérente⁽²⁾ d'un ensemble de ressources qu'ils partagent à partir de leur station de travail.

Dans une telle configuration, le départ de l'un ou de plusieurs utilisateurs, qu'il soit intentionnel (selon la volonté de l'utilisateur) ou accidentel (suite à une panne d'un site ou du réseau), ne doit pas engendrer le dysfonctionnement de l'application.

(2) Nous donnons la définition de la cohérence dans les collecticiels synchrone dans la suite.

C'est pourquoi un collecticiel synchrone doit fournir un service de déconnexion permettant à des participants de quitter un travail coopératif d'une manière intentionnelle ou suite à une panne, en préservant la cohérence des ressources partagées pour tous ceux qui y restent.

Objectifs et démarche

Notre objectif est de proposer des services garantissant la connexion et la déconnexion dynamiques⁽³⁾ de sites dans les collecticiels synchrones. La démarche que nous adoptons pour étudier ce problème est la suivante :

- cadre du travail : un collecticiel est avant tout une application répartie. De ce fait, nous situons notre problème dans le cadre de la gestion d'un groupe de processus dans un système distribué. Ici, la notion de processus est utilisée pour représenter une activité faisant partie du travail coopératif et exécutée sur l'un des sites du système (elle représente l'action d'un participant dans le collecticiel),
- nous analysons les principaux résultats et propriétés des systèmes distribués concernant les aspects de gestion de groupe,
- nous essayons ensuite de nous inspirer des solutions proposées par les systèmes distribués pour répondre aux besoins spécifiques des collecticiels synchrones,
- dans la section IV.3, nous donnons la description du service de gestion de groupes mis en œuvre dans la plate-forme *CoopScan*. Nous y proposons un protocole de connexion et un protocole de déconnexion de participants pour les collecticiels synchrones,
- nous terminons ce chapitre en donnant une évaluation du protocole de connexion et en discutant les choix de conception retenus pour sa mise en œuvre et les résultats de cette expérimentation.

IV.2 La gestion de groupes dans les systèmes distribués

IV.2.1 Le cadre de l'étude

Selon la terminologie employée pour les systèmes distribués, notre étude se situe dans le cadre des systèmes distribués dits *asynchrones*. Il s'agit de systèmes connectés par un réseau et présentant chacun une unité de calcul séparée.

(3) Pendant l'exécution du collecticiel.

Les processus qui s'exécutent dans ce système ne peuvent communiquer que par échange de messages.

Dans ce contexte, le terme *asynchrone* signifie que le système n'a pas d'horloge globale synchronisant l'exécution de toutes les unités de calcul. Par ailleurs, aucune hypothèse n'est faite sur les vitesses d'horloges, les vitesses de traitement, et les délais de transmission des messages. Ces hypothèses sont réalistes dans un contexte distribué impliquant des machines reliées par un réseau général.

Le concept de groupe

La notion de groupe est omniprésente dans le langage humain, souvent utilisée pour désigner un ensemble d'individus ayant des caractéristiques communes. Ces caractéristiques peuvent être statiques (e.g. nationalité, sexe) ou dynamiques (faisant intervenir le temps ; telles que l'âge), décrivant une activité commune sur des objets partagés.

D'une manière analogue, le concept de groupe est utilisé pour décrire l'exécution de systèmes répartis et pour en représenter les propriétés non fonctionnelles telles que la disponibilité ou la sécurité.

Dans un système distribué, l'utilisation des groupes peut être limitée à la désignation d'un ensemble de processus utilisant un espace d'adressage commun (e.g. liste d'adresses électroniques). Cependant, l'apport principal de l'utilisation des groupes repose sur la capacité de coordonner les activités d'un ensemble de processus engagés dans l'accomplissement d'une tâche commune [Powell 96].

Le concept de groupe dans les collecticiels

Le collecticiel constitue un groupe dont les membres partagent les caractéristiques suivantes :

- statiques : chaque membre du groupe est représenté dans le collecticiel par un processus s'exécutant sur une machine donnée (ce qui correspond exactement au choix de mise en œuvre adoptée dans *CoopScan*, en associant à chaque acteur un module $AgS^{(4)}$),
- dynamiques :
 - chaque membre du groupe détient une vue globale et cohérente de tout le groupe (i.e. espace de coordination),

(4) *AgS* : *Agent Session*, nom du module principal de *CoopScan* garantissant le partage d'informations, la communication et l'interaction entre les sites du collecticiel.

- les membres du groupe partagent le même ensemble de ressources (i.e. espace de coopération),
- les membres du groupe agissent sur les ressources partagées de manière asynchrone. Ici, le terme asynchrone signifie que chaque membre peut essayer d'accéder les données partagées à n'importe quel moment, et que l'accès n'est pas régi par une horloge globale du collecticiel,
- les membres du groupe ont la possibilité de rentrer dans le collecticiel et d'en sortir pendant son exécution.

L'apport de chaque participant dans le travail coopératif peut donc être assimilé à une activité accomplie par un processus s'exécutant sur un site du collecticiel.

Notre démarche consiste à modéliser le collecticiel à l'aide d'un groupe de processus afin de ramener le problème de connexion/déconnexion dynamique dans un collecticiel au problème classique d'*adhésion dans un groupe* [Birman 94].

IV.2.2 Services de gestion de groupes dans les systèmes distribués

Le problème d'adhésion à un groupe de processus (traduction du terme anglais *Group Membership*) soulève plusieurs problèmes classiques dans les systèmes distribués (tolérance aux pannes, entrée/sortie dans un groupe, cohérence des données distribuées, ...). De nombreux travaux de recherche s'intéressent à ces problèmes (e.g. [Fischer 85], [Ezhilchelvan 95], [Jahanian 93], [Kaashoek 91], [Mishra 91], [Melliar-Smith 94], [Ricciardi 91], [Verissimo 92]).

Afin de prendre en compte le problème d'adhésion à un groupe, les systèmes distribués fournissent des **SGGs** Services de **G**estion de **G**roupes.

Le rôle principal d'un service de gestion de groupes est de refléter la même vue du contexte partagé pour tous les processus participants, de permettre à de nouveaux processus de rentrer dans le groupe, et à des anciens processus de quitter le groupe. Le service de gestion de groupes doit garantir l'intégrité⁽⁵⁾ du groupe sachant qu'un processus peut quitter le groupe volontairement ou suite à une panne.

(5) Le service de gestion de groupes doit avoir connaissance de tous les processus présents au sein du groupe. Cette information est essentielle pour le fonctionnement d'autres services du système.

Les pannes pouvant survenir dans un système distribué sont de deux origines: pannes physiques des sites ou bien dysfonctionnement du réseau de communication. Les services de gestion de groupe se chargent de détecter les pannes et d'exclure du groupe tous les sites défectueux.

En résumé, le service de gestion de groupe a pour rôle principal de fournir une vue identique du groupe de processus constituant le système. Pour cela, il gère:

- l'entrée et la sortie de processus dans le groupe,
- la détection des pannes et l'exclusion des nœuds défectueux,
- alerter les autres services du système ainsi que les applications de toutes les modifications concernant la composition du groupe.

Les services de gestion sont utilisés aussi bien dans les systèmes offrant des protocoles de communication de groupes que dans les systèmes utilisant des communications *point-à-point*.

- *Protocoles de communication de groupes* : un service de gestion de groupes est un composant essentiel dans un système de communication de groupe. Il garantit l'atomicité de la diffusion dans les protocoles de communication de groupes qu'ils soient à ordre total ou causal. La mise en œuvre d'un service de gestion de groupe peut se faire à différents niveaux d'un système de communication de groupe :
 - dans le protocole lui-même, comme c'est le cas dans le protocole de Chang et de Maxemchuk [Chang 84],
 - au dessus d'un protocole de diffusion, comme c'est le cas dans le système Totem [Moser 96],
 - dans une couche protocole particulière, de façon à ce qu'il puisse être utilisé par les autres couches du système. C'est le cas du système de communication de groupe Horus [van Renesse 96].
- *Systèmes utilisant des communications point à point* : l'architecture d'un tel système est relativement simple. Le service de gestion de groupe se situe généralement au dessus de la couche communication. Typiquement, dans un environnement constitué d'un ensemble de stations de travail Unix, le service de gestion de groupe est un programme qui utilise l'interface du protocole communication *TCP/IP*. Cette configuration est celle que nous adoptons dans l'implantation de *CoopScan*.

Dans une configuration de ce type, les seules propriétés fournies par le système concernant la communication sont celles fournies par le protocole

de communication sous-jacent. Dans notre cas, *TCP/IP* fournit un mode de communication connecté offrant les propriétés suivantes :

- un mode FIFO est garanti pour la transmission des données entre l'émetteur et le récepteur,
- il n'y a pas de pertes de messages ; les erreurs d'émission sont signalées à l'émetteur.

IV.2.2.1 Propriétés d'un système de gestion de groupes

Pour distinguer les différentes propriétés d'un système de gestion de groupes, Chandra et Toueg [Chandra 92], répertorient les protocoles de gestion de groupes selon deux critères : la vivacité⁽⁶⁾ et la précision⁽⁷⁾. Un service de gestion de groupes est dit complet si tout membre défectueux fini par être exclu du groupe. La vivacité consiste à garantir que les processus qui ont un fonctionnement normal restent dans le groupe.

IV.2.2.2 Consensus dans un groupe de processus

Pour statuer sur la défaillance d'un processus, l'ensemble des processus du groupe doivent parvenir à un consensus, suite auquel la sentence d'exclusion est mise en application. L'exclusion d'un processus du groupe consiste à modifier la vue du contexte partagé sur chaque processus. La nouvelle vue du contexte est égale à l'ancienne vue sans le processus défaillant.

L'exclusion suite à un consensus ne peut être appliquée que dans le cas d'un système synchrone. En effet, seulement dans un tel système un consensus peut être atteint. Fischer et al. prouvent formellement dans [Fischer 85] qu'il est impossible de parvenir à un consensus dans un système distribué asynchrone, même s'il n'admet qu'une seule panne.

Dans ces conditions, la précision d'un système asynchrone sujet à des pannes ne peut pas être vérifiée. Ce résultat s'applique également aux collecticiels qui, dans la plupart des cas, sont des applications réparties à grande échelle ; par conséquent, assimilés à des systèmes répartis asynchrones.

Dans notre étude des protocoles de gestion de groupes nous nous intéressons exclusivement à la propriété de **précision**. Ainsi, les noeuds soupçonnés de défaillances seront exclus du groupe sans atteindre un consensus. Cette

(6) Liveness en anglais. Elle garantit que la propriété est nécessairement vérifiée au bout d'un temps fini.

(7) Safety en anglais. Elle signifie que la propriété est tout le temps vérifiée (tout au long de l'exécution).

démarche présente le risque d'exclure des nœuds non défaillants. Ce peut être le cas si, pour cause d'un délai de transmission élevé, un processus en attente d'un message déclare un autre processus défaillant, et l'exclut du groupe.

IV.2.2.3 Pannes et partitionnement de groupes

Le partitionnement d'un groupe survient quand plusieurs sous-groupes dis-joints se créent à partir d'un même groupe initial. Il en résulte que les processus dans chacun des sous-groupes n'ont plus la même vue du contexte partagé puisque chaque sous-groupe considère que les autres nœuds ne font plus partie du système. Les sous-groupes créés constituent ainsi des *partitions*.

Les partitions peuvent être de deux types : réelles ou virtuelles. Les partitions réelles sont dues à des pannes physiques du réseau de communication (par exemple, l'isolement d'un groupe de machines, toutes connectées sur le même réseau local), alors que les partitions virtuelles sont dues à des temps de communications élevés ou à des temps d'exécution disproportionnés entre les processus.

La technique souvent employée pour parer à l'apparition de partitions dans un groupe consiste à utiliser le vote majoritaire (i.e. un consensus partiel) avant chaque opération d'exclusion. Cette approche est inspirée de la propriété d'adhésion *faible* dans un groupe décrite par Chandra et al. [Chandra 95]. Son fonctionnement est fondé sur les règles suivantes :

- si aucune requête de connexion ou de déconnexion n'est formulée, rien n'est fait pour vérifier la présence de tous les membres du groupe,
- les événements concernant la connexion et/ou la déconnexion déclenchent l'exécution des services garantissant l'adhésion du groupe.

IV.2.3 Modélisation d'un collecticiel par un groupe de processus communicants

Conformément aux choix de conception adoptés dans *CoopScan*, nous modélisons un collecticiel par un groupe de processus de la manière suivante :

- le collecticiel est constitué d'un groupe de processus communicants dont chacun représente une instance de l'application,
- l'émission des messages est complètement asynchrone. Les messages sont générés suite à des actions des participants sur l'interface du collecticiel, ou en réponse à des requêtes émises par d'autres processus,
- les instances du collecticiel communiquent uniquement par le biais d'un service de communication par échange de messages,

- nous considérons que chaque processus est exécuté sur un site particulier. Nous utilisons l'appellation *site* ou *nœud* pour désigner le lieu d'exécution d'un processus.

Le collecticiel est défini formellement par un doublet (Ψ, C) , où $\Psi = \{P_1, \dots, P_n\}$ désigne l'ensemble des processus du groupe et $C = \{C_1, \dots, C_n\}$ désigne les différentes vues du contexte du collecticiel sur chaque site.

Chaque processus P_i s'exécutant sur un *site* i gère une vue C_i du contexte partagé C . La cohérence du contexte est vérifiée quand tous les processus ont une vue identique.

IV.3 Protocoles de connexion/déconnexion dynamique dans les collecticiels

Le fonctionnement du collecticiel peut être décrit à l'aide d'un graphe d'états comme l'illustre la Fig. 4.1. Chaque état du graphe regroupe un ensemble de propriétés du collecticiel.

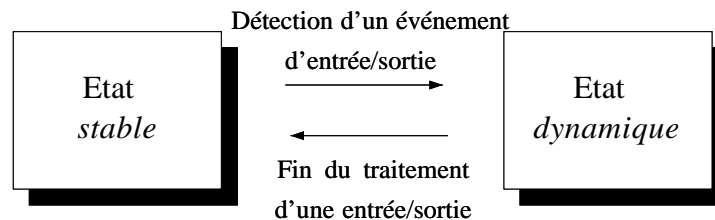


Fig. 4.1 : Les états décrivant le fonctionnement du collecticiel

– État *stable* du groupe : il définit le fonctionnement du collecticiel pour un nombre stable de participants. Cet état est caractérisé par :

- le mode d'interaction synchrone. Ce mode est indispensable pour atteindre un fonctionnement WYSIWIS⁽⁸⁾ du collecticiel,
- la cohérence des données partagées, les données étant dupliquées sur tous les sites conformément à un schéma d'architecture totalement répliqué.

– État *dynamique* du groupe : dans cet état, le collecticiel traite les opérations d'adhésion au groupe. Le collecticiel doit :

(8) What You See Is What I See.

- pouvoir traiter toute requête de connexion/déconnexion au bout d'un temps fini en garantissant que les protocoles de connexion et de déconnexion ne provoquent pas de situations d'inter-blocage,
- fournir au nouvel arrivant le contexte du collectif (espace de coordination et espace de coopération), définis dans le chapitre introduction,
- prendre en compte le départ d'un participant, qu'il soit volontaire ou résultant d'une défaillance,
- modifier la vue du contexte partagé. Cette opération a pour but d'avertir les membres du collectif de l'arrivée ou du départ d'un participant,

En résumé, dans un état *stable*, le collectif doit vérifier la propriété de cohérence alors que dans un état *dynamique* il doit vérifier la propriété de correction.

Nous définissons la correction de la manière suivante : un protocole est correct s'il ne génère pas d'inter-blocages et s'il se termine au bout d'un temps fini (en donnant une réponse favorable ou négative) [Ellis 89].

IV.3.1 Protocoles de gestion de groupes

La coopération synchrone est définie au moyen de règles d'interaction et de partage d'information notamment la communication temps-réel entre les participants et le partage de documents en mode WYSIWIS. L'entrée et la sortie d'un membre dans le/du groupe devra se faire conformément aux règles établies.

Hypothèses et objectifs

Pour la mise en œuvre des protocoles de gestion groupes, nous partons des hypothèses suivantes :

- la propriété de vivacité est traitée en priorité par rapport à la propriété de précision. Nous essayons donc de garantir uniquement que tout processus défectueux est exclu au bout d'un temps fini,
- les actions des participants utilisant une application partagée sont soumises à une politique d'exclusion mutuelle,
- le traitement des requêtes de connexion et de déconnexion est totalement asynchrone. Le site demandeur doit disposer d'une réponse au bout d'un temps fini.

Services de gestion de groupes

De même, la déconnexion d'un acteur revient à informer tous les membres du groupe de son départ afin qu'ils puissent modifier leurs vues respectives du contexte partagé.

Par ailleurs, nous ne faisons aucune hypothèse sur l'environnement d'exécution par rapport à la machine ou au réseau utilisé. Les pannes du réseau et des sites doivent être prises en compte par l'application. Le départ d'un site peut donc être volontaire ou lié à une panne.

Le protocole de déconnexion peut donc être déclenché sous deux conditions : la détection d'une défaillance ou la demande explicite d'une déconnexion.

La détection des défaillances est appliquée quand le collecticiel est dans un état de fonctionnement *dynamique*. C'est uniquement dans cet état que la cardinalité du groupe risque d'être modifiée.

IV.3.2 Architecture

IV.3.2.1 Architecture totalement dupliquée

Les données du contexte partagé sont totalement dupliquées sur les sites du collecticiel. Chaque processus P_i gère donc sa propre vue du contexte C_i . Conformément au modèle défini dans le chapitre I, le contexte est formé de deux espaces : un espace de coordination et un espace de coopération.

IV.3.2.2 Structure de données

Les principales informations contenues dans l'espace de coordination sont :

- l'identificateur de la session,
- la liste de tous les processus au sein du collecticiel. Chaque processus est représenté par un doublet : [*process_ID*, *info*].

L'attribut *process_ID* permet de définir de manière unique un processus du groupe (i.e. un acteur du collecticiel). L'attribut est composé d'un attribut *port_num* définissant le numéro du port attribué au processus et d'un attribut *adresse_IP* définissant l'adresse IP de la machine hôte du processus.

L'attribut *info* permet d'accéder à plusieurs informations du processus, parmi lesquelles :

- le nom du participant,
- le rôle de l'acteur : il s'agit des droits attribués à l'acteur pour chacune des applications partagées du collecticiel. Cette information se présente sous la forme d'une liste de tuples [*rôle*, *application_id*] où *rôle* prend

ses valeurs dans $\{Lire, Ecrire, Null\}$ et *application_id* est l'identificateur de l'application partagée. Le doublet $[Null, application_id]$ signifie que l'acteur n'utilise pas l'application dont l'identificateur est *application_id*.

- l'identificateur du site président,
- l'état courant du processus : exclu, en cours de connexion, connecté, en attente d'un jeton, ...
- les politiques de passage du jeton disponibles⁽⁹⁾,
- le *statut* de chaque application du collecticiel, sachant que dans CoopScan, trois groupes disjoints d'applications⁽¹⁰⁾ sont définis : les applications disponibles, les applications de la session, et les applications de travail. L'appartenance de l'application à un groupe définit son *statut*.

L'*espace de coopération* contient les applications partagées qui sont utilisées dans le collecticiel. Les principales informations contenues dans cet espace sont :

- les participants à chaque application,
- la politique de passage du jeton adoptée pour chaque application,
- l'historique des opérations d'écriture exécutées par chaque application. Formellement, pour chaque processus P_i , il s'agit de la restriction de la vue C_i à chaque application partagée dans le collecticiel.

IV.3.2.3 Support de la communication

Pour l'acheminement des messages entre les processus du groupe, nous utilisons un service de communication point-à-point fiable respectant l'ordre FIFO. Ce service correspond au niveau de transport du modèle de communication OSI.

Les fonctions utilisées pour l'émission et la réception de messages utilisent des primitives de communication fournies par la famille de protocoles TCP/IP.

La communication entre les processus est mise en œuvre par les fonctions suivantes :

(9) Nous utilisons actuellement dans CoopScan deux politiques pour la gestion du droit de parole. La première est fondée sur une désignation explicite. Dans la seconde, le passage du jeton se fait conformément à un ordre FIFO d'arrivée des requêtes sur le site détenteur du jeton.

(10) La définition des trois groupes d'applications est donnée dans le chapitre II.

SendMessage (dest, msg) : cette fonction permet l'émission d'un message *msg* vers une destination *dest*. La fonction retourne un entier positif dans le cas où l'émission est correcte.

DiffuseMessage (dest_list, msg) : cette fonction permet la diffusion d'un message *msg* vers les destinations contenues dans la liste *dest_list*. La fonction retourne la liste des sites vers lesquels le message a été correctement transmis.

La réception des messages est effectuée de manière asynchrone sur chaque site du collecticiel.

IV.3.3 Connexion

Le protocole de connexion permet à un utilisateur de participer à une session⁽¹¹⁾ de travail. Il permet également à un processus défectueux de se reconnecter et intégrer de nouveau le groupe.

Le protocole de connexion est fondé sur un principe de validation à deux phases dont la première est menée par le site *président* et la seconde menée conjointement par le site *président* et le nouveau membre lui-même. Les deux phases du protocole sont :

Phase 1 : modification des vues sur chaque site.

Phase 2 : confirmation de la rentrée du nouveau membre dans le groupe et reprise de l'activité du groupe.

Par ailleurs, le protocole de connexion utilise une *technique de gel* qui consiste à bloquer (*geler*) les activités du groupe sur le contexte partagé durant la phase de connexion.

Messages échangés lors de la connexion

Les messages échangés entre les sites du collecticiel sont conformes au format : $\langle \text{MESSAGE_CODE}, \text{message_body} \rangle$. *message_body* représente la partie données, alors que *MESSAGE_CODE* désigne le code du message. Au cours de la phase de connexion, *message_code* prend les valeurs suivantes :

(11) Défini dans le chapitre I, ce terme permet d'identifier de manière unique le groupe de processus qui forment le collecticiel. La définition de ce terme est précisée pour CoopScan dans le chapitre II.

(12) Dans la version actuelle du prototype CoopScan, un seul jeton permet l'accès à toutes les applications. Le fonctionnement du protocole de connexion est néanmoins identique dans les deux cas.

- *JOIN_SESSION_REQ* : il s'agit de la requête transmise par le nouvel arrivant au site président pour demander à rejoindre le collecticiel.
- *JOIN_NOTIFY* : le site président envoie ce message à tous les membres du groupe pour les avertir de l'occurrence d'une demande de connexion.
- *JOIN_NOTIFY_ACK* : c'est le message transmis par chaque membre du groupe en réponse au message *JOIN_NOTIFY*.
- *JOIN_NOTIFY_NACK* : ce message est transmis par un nœud qui répond négativement à la notification *JOIN_NOTIFY*. C'est utile dans le cas où l'admission d'un nouveau membre est soumise à l'acceptation de tous les participants ou à une partie d'entre eux.
- *JOIN_SESSION_ACCEPT* : il s'agit du message transmis par le site président au nouvel arrivant en signe d'acceptation de sa requête de connexion.
- *JOIN_NOT_ACCEPT* : le site président transmet ce message en signe de refus de la connexion.
- *FREEZE* : le site président envoie ce message pour "geler" l'activité du collecticiel.

Nous distinguons dans *CoopScan* deux variantes de ce message, respectivement *forte* et *faible* : *FREEZE_APP*, *FREEZE_SESSION*. La variante *forte* *FREEZE_APP* est utilisée pour stopper **toutes les activités** dans le collecticiel, qu'elles soient accomplies sur l'espace de coordination ou sur l'espace de coopération (le but étant d'interrompre les mises à jour sur les applications et sur les paramètres de l'espace de coordination). La variante *faible* *FREEZE_SESSION* concerne uniquement les activités susceptibles de modifier l'espace de coordination.

Afin de garantir la flexibilité du collecticiel, un utilisateur qui se connecte à une session *CoopScan* est successivement, connecté au groupe des participants (espace de coordination), puis, à sa demande, aux sous-groupes formés par les applications partagées.

Dans *CoopScan*, la variante *faible* est donc utilisée pour connecter le nouvel arrivant à la session. A l'issue de la connexion, le participant a une vue cohérente de l'espace de coordination.

La variante *forte* du message *FREEZE_SESSION* est utilisée lors de la connexion d'un participant à une application du collecticiel.

Cependant, dans un cas ou dans l'autre, le protocole de connexion se déroule toujours selon le même principe de validation à deux phases.

Nous utilisons dans la suite, le terme *FREEZE* pour désigner les deux variantes du message.

- *FREEZE_ACK* : c'est le message envoyé par les membres du collecticiel à destination du président, suite à la réception du message *FREEZE*.
- *ADD_CONTEXT* : ce message est émis par le président pour demander à un membre de mettre à jour son contexte. La nouvelle valeur du contexte est envoyée dans la partie *message_body* du message. Plusieurs émissions de ce message peuvent s'avérer nécessaires pour reconstituer la totalité du contexte.
- *END_CONTEXT* : ce message désigne la fin de la phase de restitution du contexte.
- *JOIN_VALID* : c'est le message envoyé par le nouvel arrivant en signe de validation de la phase de connexion. A la réception de ce message, le site président dégèle la session.

Fonctionnement du site président

Le graphe de transition de la figure Fig. 4.2 illustre le fonctionnement du site président lors de la phase de connexion.

Fig. 4.2 : Graphe décrivant l'évolution des états du site président pendant l'exécution du protocole de connexion

- `IN_SESSION` : c'est l'état dans lequel se trouvent tous les processus quand le collecticiel est dans un état *stable*. Le site président quitte cet état dès la réception d'une requête de connexion.

À l'issue de la phase de connexion, le nouveau membre est dans un état `IN_SESSION`.

- `FREEZE_STATE` : le président passe dans l'état `FREEZE_STATE` dès qu'il reçoit une requête de connexion. Il transmet ensuite des messages `FREEZE` à destination des sites du collecticiel.

Le site président attend les réponses jusqu'à l'expiration d'un délai de garde. Les sites n'ayant pas répondu au bout de ce délai sont exclus par le président. La déconnexion est accomplie selon le protocole de déconnexion que nous présentons plus loin.

- `CON_STATE` : le président transite de l'état `FREEZE_STATE` à l'état `CON_STATE` dès la réception des messages `FREEZE_ACK` ou à l'expiration du délai de garde. Dans cet état, le président dispose de la vue du contexte la plus récente. En effet, deux cas de figures sont possibles : les nœuds détenteurs de jetons envoient les messages `FREEZE_ACK`, auquel cas les messages contiennent les vues les plus récentes de l'espace de coopération. Le cas échéant, le collecticiel déclenche le protocole de déconnexion en considérant sa vue du contexte comme étant la vue la plus récente.

Dans cet état, le président diffuse un message `JOIN_NOTIFY` à destination de tout le groupe (conformément à sa vue du contexte).

- `VAL_STATE` : le président passe dans cet état quand il reçoit tous les messages `JOIN_NOTIFY_ACK` / `JOIN_NOTIFY_NACK` ou à l'expiration du délai de garde.

Fonctionnement du site demandeur de la connexion

Le principe de fonctionnement consiste à envoyer une requête, récupérer le contexte du groupe, créer une vue locale du contexte et valider la connexion.

La figure Fig. 4.3 illustre les étapes d'exécution du protocole de connexion pour le site demandeur de la connexion.

Fig. 4.3 : Graphe d'exécution du site demandeur de la connexion

- **OUT_SESSION** : dans cet état, l'utilisateur est considéré en dehors de la session coopérative. L'utilisateur quitte cet état suite à l'émission d'une requête de connexion.
- **WAIT_CONNECT** : le site rentre dans cet état après l'émission avec succès de sa requête d'adhésion.
- **GET_CONTEXT** : le site rentre dans cet état dès la réception du message *JOIN_ACCEPT*. Il construit sa vue du contexte en récupérant les données contenues dans les messages *ADD_CONTEXT* qui lui sont envoyés par le président. Le site quitte cet état dès la réception du message *END_CONTEXT*.
- **IN_SESSION** : le site demandeur passe dans cet état dès qu'il reçoit le message *END_CONTEXT* lui indiquant la fin de la phase de reconstitution du contexte. Le nouveau membre émet alors le message de validation *JOIN_VALID* confirmant ainsi son adhésion au groupe.

Fonctionnement des autres sites du collecticiel

A la réception du message *JOIN_NOTIFY*, tous les sites récupèrent le nouveau contexte transmis par le président.

Toutefois, les sites n'effectuent la modification de leurs vues respectives qu'une fois le message de validation *JOIN_VALID* leur soit parvenu.

IV.3.4 Déconnexion

Le protocole de déconnexion permet de sortir un processus du groupe en préservant la cohérence des vues du contexte partagé.

L'exclusion d'un nœud est réalisée en deux étapes dont la première consiste à détecter le nœud défaillant et la seconde à supprimer le nœud dans toutes les vues du groupe.

La détection

La déconnexion d'un nœud peut avoir deux causes :

- une demande explicite formulée par un participant désirant quitter le collectif. Dans ce cas, le site diffuse un message $\langle QUIT_SESSION, member_id \rangle$ vers tous les membres du groupe, suite à quoi, il quitte la session.
- une défaillance détectée par l'un des participants suite à une erreur de communication.

La suppression

Le site ayant détecté la défaillance diffuse un message $\langle QUIT_SESSION, member_id \rangle$ vers tous les autres. Chaque membre qui reçoit le message *QUIT_SESSION*, recherche le processus identifié par *member_id* dans sa vue. S'il y est, le membre modifie sa vue et rediffuse le message aux autres. Sinon, il ignore le message. La technique de retransmission est classique. Elle est utilisée dans le but de garantir la réception du message par tous les membres.

Dès la réception d'un message *QUIT_SESSION*, le président vérifie que le site partant (ou défaillant) n'avait pas un jeton en sa possession. Si c'est le cas, le président se charge d'en générer un nouveau.

IV.3.5 Évaluation du SGG dans *CoopScan*

Les services de gestion de groupes (SGG) présentés dans ce chapitre sont mis en œuvre dans la plate-forme *CoopScan*. Afin de simplifier la compréhension des protocoles, nous faisons dans ce chapitre quelques restrictions sur l'utilisation réelle de ces protocoles dans *CoopScan*.

La démarche adoptée lors de la connexion à une session *CoopScan* est la suivante : un utilisateur se connecte d'abord à une session *CoopScan*. A l'issue de cette phase, le participant récupère une vue cohérente de l'espace de coordination (une partie du contexte partagé). Ensuite, le participant a le choix de

participer à une ou plusieurs applications partagées dans la session. Ces applications peuvent être déjà utilisées par d'autres participants (*applications de la session*⁽¹³⁾), ou encore inutilisées (*applications disponibles*⁽¹³⁾).

Si l'utilisateur souhaite participer à une application de la session, le service de gestion de groupe prend en compte sa requête et lance l'exécution du protocole de connexion. Dans ce cas, le participant récupère le contexte de l'application à laquelle il souhaite se connecter.

La connexion d'un nouvel arrivant à la session se fait selon le protocole décrit précédemment. Le site président *gèle* toutes les activités sur l'espace de coordination. Les opérations susceptibles de rompre la cohérence sont le lancement d'une nouvelle application, le passage du droit de parole, et la déconnexion d'un site.

Lorsqu'un participant faisant déjà partie de la session souhaite utiliser une application du collecticiel, il émet une requête de connexion au site président. Celui-ci, déclenche l'exécution du protocole de connexion en utilisant la variante *forte* du message *FREEZE*.

IV.3.5.1 Motivation

L'objectif poursuivi à travers ce travail est de pouvoir évaluer le comportement des protocoles fournis par le SGG CoopScan lors d'une utilisation réelle du collecticiel. Plus précisément, nous souhaitons calculer les temps de réponses délivrés par le système lors de la connexion de nouveaux participants à travers plusieurs manipulations.

Les concepts qui interviennent dans l'évaluation sont les suivants :

- la taille du contexte partagé du collecticiel,
- le nombre de participants,
- le temps de réponse de l'application.

Le temps de réponse désigne la période écoulée entre l'instant où l'utilisateur formule une requête de connexion et l'envoi au président, et l'instant où le nouvel arrivant reçoit la totalité du contexte.

Il ne s'agit pas pour nous d'évaluer des temps de réponse absolus, ces temps étant étroitement liés à l'application et ne peuvent être comparés d'une application à une autre (par exemple, le contexte d'un éditeur tel que *xedit* est, en général, sensiblement moins important que celui d'un éditeur graphique).

(13) groupe d'applications de CoopScan présenté dans le chapitre II.

IV.3.5.2 Mise en œuvre

Afin de tester les protocoles proposés, nous mettons en place une maquette d'exécution dans laquelle nous essayons de restituer le plus fidèlement possible les hypothèses d'une utilisation réelle du collecticiel.

L'environnement informatique dans lequel sont évalués les protocoles de gestion de groupe est le suivant :

- un ensemble de stations Unix connectés à travers un réseau ETHERNET. Deux types de machines sont utilisées : des stations Estrella Bull Power PC et des stations SUN Sparc 20,

- chaque utilisateur lance l'exécution d'une instance de *CoopScan* intégrant l'application partagée *CoopGrail*,

- nous testons le fonctionnement du protocole pour une dizaine d'utilisateurs. Cependant, l'application a fait l'objet d'une utilisation plus importante lors d'une démonstration organisée dans notre laboratoire. Lors de cette utilisation, 17 personnes ont participé à la session de coopération pendant une durée moyenne de 30 minutes. Malgré des temps de réponse qui commençaient à être importants pour les derniers participants, la démonstration s'est bien déroulée.

- le protocole de connexion permet à un utilisateur d'adresser sa requête de connexion à n'importe quel site du collecticiel, la requête est alors automatiquement re-dirigée vers le site président. L'adhésion d'un nouveau membre n'est pas soumise à son acceptation par tout le groupe ou à la décision d'un utilisateur quelconque.

- les paramètres d'évaluation sont le temps de réponse du protocole et le nombre de participants du collecticiel. Afin d'avoir des résultats les plus significatifs possible, nous considérons un scénario dans lequel l'espace de coopération ne subit pas de modifications importantes entre deux connexions successives.

Le contexte partagé dans *CoopGrail* est essentiellement composé des navigateurs W3 ouverts depuis le début de la session. Chaque navigateur est affichée dans une fenêtre indépendante. Les fenêtres de *CoopGrail* affichent chacune une page *html*.

Dans le temps de réponse, nous ne comptabilisons que le temps d'ouverture des fenêtres. Le temps requis par le navigateur pour résoudre l'*URL*⁽¹⁴⁾, et télé-charger les pages *html* n'est pas pris en compte.

(14) En anglais, Uniform Resource Location.

Fig. 4.4 : Évaluation du protocole de connexion.

IV.3.5.3 Interprétation des résultats

Chaque courbe de la figure Fig. 4.4 représente la variation des temps de réponse collectés pour la connexion de 10 utilisateurs à l'application *CoopGrail*. Le temps (en ordonnées) est affiché en millième (10^{-3}) de seconde. En réalité, chaque point du graphe correspond à une valeur moyenne du temps mesuré lors de **trois** expérimentations du protocole. Les tests ont été réalisés pour trois configurations du contexte partagé contenant respectivement 1, 2, 3 navigateurs.

Les tendances affichées par les trois courbes sont conformes à nos prévisions fondées sur le principe de fonctionnement du protocole d'adhésion. Les résultats de la Fig. 4.4 s'interprètent de la manière suivante :

- les temps de réponse croissent en fonction du nombre d'utilisateurs connectés. Ce résultat est parfaitement conforme au schéma d'exécution du protocole de connexion. En effet, le nombre de communications point-à-point croît en fonction du nombre de site concernés par la diffusion de messages.

- les temps de réponse croissent en fonction de la taille du contexte partagé devant être reconstitué sur le site du nouvel arrivant,
- les temps affichés pour les utilisateurs *user 3* et *user 7* marquent une certaine rupture dans la progression de la courbe. Ils sont tous les trois sur les machines les plus rapides du collectif.

IV.3.6 Discussion

La différence essentielle distinguant les collecticiels synchrones du reste des systèmes distribués "classiques" est la présence humaine quasi-permanente tout au long de l'exécution de l'application. En plus des interactions asynchrones intervenant entre les processus du groupe, les participants interagissent doublement avec le système. Cela s'opère de deux façons:

- l'utilisateur est l'instigateur des interactions entre processus,
- l'utilisateur constitue une "entité" de contrôle qui est capable d'évaluer de manière continue le fonctionnement du système et de prendre, aux moments opportuns, les décisions qui s'imposent (par exemple, demander à être reconnecté au groupe suite à une exclusion jugée injuste).

Nous essayons dans *CoopScan* de tenir compte de ces spécificités dans la conception des services de gestion de groupes.

Les protocoles de gestion de groupes proposés dans *CoopScan* couvrent plusieurs aspects importants du fonctionnement d'un collecticiel. Nous les regroupons dans ce qui suit de la manière suivante :

IV.3.6.1 Atomicité et cohérence

Les protocoles de gestion de groupes doivent garantir l'atomicité des opérations d'entrée/sortie dans le groupe. Cette propriété permet aux membres du groupe d'avoir la même vue du contexte partagé.

Fig. 4.5 : La propriété d'atomicité pour l'opération de connexion.

La Fig. 4.5(a) illustre une diffusion non atomique effectuée du site 3 vers les sites 1 et 2 pour leur signaler son arrivée dans le collecticiel. En conséquence, le *site 1* et le *site 2* n'ont pas la même vue du contexte partagé, et le nouvel arrivant ne reçoit pas les messages diffusés par le *site 1*. Par contre dans le cas (b), la connexion est atomique et les sites 1 et 2 ont une vue identique du contexte.

Afin que la connexion préserve la cohérence des vues sur tous les sites, notre protocole adopte une politique de "gel" qui consiste à bloquer l'activité du groupe pendant la phase de connexion.

Dans l'état stable, du fait qu'un seul site à la fois diffuse des messages vers les autres, la diffusion atomique suffit pour garantir la cohérence du contexte.

Cependant, de telles conditions peuvent alourdir le fonctionnement du collecticiel et génèrent souvent des temps de réponse trop longs.

Dans le but de garantir des temps de réponses les plus courts possibles, nous considérons que tous les sites reçoivent correctement les messages diffusés. Ce fonctionnement peut être à l'origine d'incohérences entre les vues des processus.

La cohérence est rétablie au moment où s'effectue le passage du jeton. L'ancien détenteur du jeton envoie avec le jeton une image de son contexte dans un message SET_CONTEXT afin que le nouveau détenteur du jeton puisse comparer sa vue du contexte avec le contexte reçu, et qu'il puisse mettre à jour sa vue du contexte au cas où les deux vues sont différentes.

Dans l'implantation actuelle du service de gestion de groupe dans *CoopScan*, nous nous intéressons essentiellement aux aspects dynamiques du collecticiel, à

savoir l'entrée/sortie de participants. La cohérence de l'espace de coopération est vérifiée lors de l'exécution du protocole de connexion.

IV.3.6.2 Panne d'un site

Défaillance d'un site détenteur du jeton

Le site défaillant peut être détecté lorsqu'un autre site du collecticiel émet vers lui une requête pour l'acquisition du jeton. Le site émetteur se met en attente d'un acquittement de la part du détenteur du jeton. Si à l'expiration d'un délai de garde, l'acquittement de la requête n'est toujours pas parvenu au site, celui-ci déclenche le protocole de déconnexion.

Nous justifions l'adoption d'une telle démarche par le caractère interactif du collecticiel. En effet, au bout d'un temps fini, les participants ne percevant plus de modifications sur l'espace de coopération demandent nécessairement le jeton.

Défaillance du site président

La défaillance du site président peut survenir au moment de l'exécution du protocole de connexion, ou pendant le déroulement de la coopération. Dans ce cas, le traitement de la défaillance est identique à celui fait pour les autres sites.

Plusieurs variantes du protocole de déconnexion peuvent exister. Par exemple, s'agissant du site président, les membres du groupe peuvent procéder par vote pour vérifier que le site est bien défaillant avant de l'exclure.

Élection d'un nouveau président

Le collecticiel génère, selon un ordre croissant, des identificateurs qu'il attribue aux participants dès leur adhésion au groupe. C'est le site qui détient l'identificateur le plus petit (le site le plus ancien dans le collecticiel) qui est élu nouveau *président* en cas d'exclusion du site *président*. La mise à jour de cette information dans l'espace de coordination (pour modifier le paramètre président) est faite systématiquement par chaque site dès qu'il détecte la défaillance du président. Le schéma répliqué facilite la mise en œuvre de cet algorithme.

IV.3.6.3 Reconstitution du contexte

Chaque site gère une file d'événements associée à chacune des applications partagées du collecticiel. Les événements qui y figurent représentent les actions exécutées par l'application en question. La mémorisation des actions accomplies sur l'espace de coopération est sélective. En effet, seuls les actions d'"écriture" susceptibles de modifier l'espace de coopération y sont représentées.

IV.3.6.4 Détection des pannes

Un site peut être déclaré défaillant à l'expiration d'un délai de garde pendant lequel plusieurs tentatives d'émission échouent.

Afin de ne pas alourdir le fonctionnement du collecticiel, la défaillance des sites n'est pas prise en compte lors de la diffusion des actions accomplies sur l'espace de coopération. Seules les émissions de messages de contrôle pendant les phases de connexion, de passage du jeton, de demande du jeton, ou de déconnexion, permettent de détecter d'éventuelles défaillances. La cohérence des vues, et notamment celle de l'espace de coopération, peut alors ne pas être vérifiée en permanence.

Pour cela, nous utilisons dans *CoopScan* le protocole de communication *UDP* pour échanger les données de l'espace de coopération.

IV.3.6.5 Déconnexion volontaire d'un site

Si le site souhaitant se déconnecter possède un jeton, il est contraint de s'en dessaisir avant de quitter la session.

IV.3.6.6 Constitution du groupe

Chaque participant peut, à tout moment, consulter la liste des acteurs du collecticiel. La liste des "présents" ainsi que plusieurs informations sur le déroulement du travail sont en permanence affichées sur l'interface du collecticiel.

IV.3.6.7 Prise en compte du partitionnement du groupe

Habituellement, les services d'adhésion d'un système distribué partitionnable fournissent un protocole (voire plus) de fusion des partitions.

Le mode d'interaction synchrone et la présence d'acteurs humains dans les collecticiels rendent le problème de partitionnement moins délicat qu'il ne l'est dans les systèmes distribués, et cela pour les raisons suivantes :

- dans un collecticiel synchrone les interactions entre participants se font en temps réel. Il est inutile de garder des sites dans le groupe sachant qu'ils ne perçoivent pas les messages transmis par les autres membres en temps réel.
- le participant est immédiatement averti à travers l'interface du collecticiel de son exclusion (les pannes réseau sont la cause la plus fréquente). Il peut donc demander à se reconnecter de nouveau au groupe.

IV.4 Conclusion

Le problème d'adhésion dans un système asynchrone constitue un domaine de recherche à part entière. Notre objectif n'étant pas de le traiter dans sa globalité, nous nous inspirons dans notre travail des principaux résultats établis dans ce domaine afin de répondre à des besoins spécifiques d'une classe d'applications particulière : les collecticiels synchrones.

Notre étude est fondée sur les spécificités des collecticiels synchrones : la communication temps réel, la cohérence des données partagées et la "flexibilité" d'utilisation. Le terme flexibilité est utilisé pour caractériser les fonctions du collecticiel permettant aux utilisateurs de faire partie d'un groupe, de participer à plusieurs tâches coopératives, de quitter un groupe à n'importe quel moment et de pouvoir s'y reconnecter.

Afin de mettre en place un travail coopératif incluant plusieurs utilisateurs, nous proposons un service de gestion de groupes qui fournit les protocoles suivants :

- un protocole de connexion dynamique qui permet l'adhésion de nouveaux participants dans le collecticiel. Le protocole fournit à un nouvel arrivant une vue cohérente du contexte partagé et modifie les vues respectives de chaque membre pour y ajouter les références du nouveau membre.

Le protocole de connexion est implanté dans *CoopScan* pour deux applications coopératives : un éditeur partagé et un navigateur W3 coopératif. La correction et la cohérence du protocole ont également été validées à travers une modélisation et la vérification d'une première version du protocole dans le langage LOTOS [Kerbrat 95]. Ce travail a été accompli en deux étapes :

- la description en LOTOS de l'architecture *CoopScan* : les composants primitifs (composants OLAN) sont représentés par des processus LOTOS (i.e. le composant *AgS*) dont le comportement est une transcription directe de l'implantation du composant lui-même. L'ensemble des composants *AgS* est représenté par un processus de contrôle englobant tous les processus déclarés pour chaque *AgS*. Ce processus de contrôle permet de définir toutes les communications possibles entre les composants *AgS* lors de l'exécution du protocole de connexion. La description complète de *CoopScan* contient environ 3000 lignes de code LOTOS.
- la vérification des propriétés : *Liveness* et *Safety* du protocole de connexion à une session implanté dans *CoopScan*. Nous utilisons pour cela la boîte à outils CÆSAR–ALDEBARAN [Fernandez 92] qui

intègre un compilateur LOTOS et des outils pour la détection d'inter-blocage (en anglais *deadlock*), la simulation, et la vérification de formules dans une logique temporelle,

– un protocole de déconnexion qui permet aussi bien l'exclusion d'un nœud défaillant que la déconnexion d'un participant souhaitant quitter le collecticiel.

Plusieurs aspects de la gestion de groupes sont abordés dans cette étude, notamment l'entrée/sortie d'un processus dans un groupe, la cohérence des vues d'un ensemble de processus, et les pannes de sites (virtuelles ou réelles).

Par contre, le service de gestion de groupes que nous proposons ne prend pas en compte la fusion automatique des partitions dans un groupe. Nous estimons qu'un protocole de fusion peut affecter l'interactivité du collecticiel.

Chapitre IV

Gestion de groupes dans les collecticiels synchrones

IV.1 Introduction.	119
IV.2 La gestion de groupes dans les systèmes distribués	120
IV.2.1 Le cadre de l'étude	120
IV.2.2 Services de gestion de groupes dans les systèmes distribués	122
IV.2.2.1 Propriétés d'un système de gestion de groupes	124
IV.2.2.2 Consensus dans un groupe de processus	124
IV.2.2.3 Pannes et partitionnement de groupes.	125
IV.2.3 Modélisation d'un collecticiel par un groupe de processus communicants	125
IV.3 Protocoles de connexion/déconnexion dynamique dans les collecticiels.	126
IV.3.1 Protocoles de gestion de groupes	127
IV.3.2 Architecture	128
IV.3.2.1 Architecture totalement dupliquée	128
IV.3.2.2 Structure de données	128
IV.3.2.3 Support de la communication	129
IV.3.3 Connexion	130
IV.3.4 Déconnexion	135
IV.3.5 Évaluation du SGG dans <i>CoopScan</i>	135
IV.3.5.1 Motivation.	136
IV.3.5.2 Mise en œuvre.	137
IV.3.5.3 Interprétation des résultats	138
IV.3.6 Discussion	139
IV.3.6.1 Atomicité et cohérence	139
IV.3.6.2 Panne d'un site	141
IV.3.6.3 Reconstitution du contexte	141
IV.3.6.4 Détection des pannes.	142
IV.3.6.5 Déconnexion volontaire d'un site	142

IV.3.6.6	Constitution du groupe	142
IV.3.6.7	Prise en compte du partitionnement du groupe . . .	142
IV.4	Conclusion	143

Chapitre V

Réalisations

V.1 Introduction

Nous avons proposé dans le chapitre II un modèle d'architecture générique pour la construction et l'exécution de collecticiels. Notre étude nous a permis d'identifier les fonctions coopératives fournies par les collecticiels, et d'analyser les architectures possibles pour leur mise en œuvre dans divers environnement distribués.

Nous proposons dans ce chapitre une description de l'implantation de ce modèle sur deux types de plates-formes. A l'issue de l'étude des environnements de mise en œuvre effectuée dans le chapitre III, nous avons implanté *CoopScan* sur une plate-forme Unix et sur une plate-forme à objets répartis obéissant au modèle d'architecture CORBA. Par ailleurs, nous proposons à travers *CoopScan*, une extension de l'infrastructure WWW afin qu'elle puisse supporter le mode de coopération synchrone.

Nous terminons ce chapitre en dressant un bilan de cette étude ainsi que les perspectives de ce travail.

V.2 Réalisations

Nous décrivons dans cette section la mise en œuvre de la plate-forme *CoopScan* intégrant deux applications partagées : un éditeur de documents partagé et un navigateur *World Wide Web* coopératif.

Deux implantations de *CoopScan* sont issues de la description *OLAN* présentée dans le chapitre II. *CoopScan* a été implantée sur deux types de plates-formes : une plate-forme Unix, et une plate-forme à objets répartis OODE⁽¹⁾ (*Object-Oriented Development Environment*). Le prototype intègre *Thot*, un éditeur de documents multimédia développé à l'INRIA-IMAG [Quint 94]

(1) *Object-Oriented Development Environment* : plate-forme à objets répartis, réalisée dans le cadre d'un projet commun à la société Bull et à l'IMAG.

, et un navigateur *WWW* [van Rossum 96]. Nous nous limiterons dans ce chapitre à présenter l'implantation Unix du prototype.

V.2.1 Déploiement de *CoopScan* sur une plate-forme Unix

L'instantiation de la description *OLAN* de *CoopScan* sur une plate-forme Unix consiste à attribuer à chaque site du collecticiel trois processus : le premier met en œuvre l'*AgS*, et les deux autres processus représentent les *applications intégrées*. Nous désignons par application intégrée : le module *AgL*, l'application initiale et le module *AgD*. Ce schéma de mise en œuvre nous est imposé par la spécificité des interfaces offertes par les applications, chacune d'elles fournissant sa propre *API* et son propre mécanisme de notification. De ce fait, l'*AgL* est réparti en plusieurs sous-modules (dans ce cas 2) qui sont d'une part liés aux applications, et d'autre part, connectés à l'*AgS*.

De cette manière, l'application est représentée par un processus complètement autonome. En effet, ce processus est capable de rendre-compte des exécutions de l'interface-utilisateur à l'*AgL*. Il permet également aux fonctions de l'*AgD* de reproduire les actions exécutées sur d'autres sites du collecticiel.

V.2.1.1 L'instanciation des connecteurs *OLAN* sur une plate-forme Unix

La communication entre les différents processus est mise en œuvre à l'aide des protocoles de communication fournis par *TCP/IP* [Comer 92]. L'instanciation des connecteurs *OLAN* sur la plate-forme Unix est réalisée de la manière suivante :

- communication *AgS–AgS* : les communications entre *AgS* peuvent être réalisées selon deux modes : un mode connecté en utilisant des canaux *TCP/IP* entre chaque paire d'*AgSs* mettant en œuvre des transmissions de données selon le mode *FIFO*⁽²⁾, ou un mode déconnecté (*Datagram*) en utilisant le protocole *UDP*,
- communication *AgS–AgL* : l'*AgS* et l'*AgL* sur le même site échangent des messages via des communications internes au domaine Unix de la machine. Ces communications sont mises en œuvre par le protocole *UDP*,
- communication *AgL–AgDs* : ces communications sont mises en œuvre à l'aide du protocole *UDP*.

(2) FIFO : en anglais *First In First Out*

Fig. 5.1 : La communication dans CoopScan

La Fig. 5.1 illustre la mise en œuvre de la communication dans CoopScan.

Le choix du protocole dépend du type de l'information échangée entre les modules. Typiquement, le protocole *TCP* est utilisé pour la communication des messages de contrôle entre les *AgSs* (lors de l'exécution des protocoles de connexion, d'exclusion, ou d'échange du droit de parole) ; les pertes n'étant pas tolérées pour ce type de messages.

Pour cela, les primitives d'émission fournies par le protocole *TCP/IP* permettent de contrôler l'émission des messages et d'avertir l'émetteur d'un éventuel échec de la communication.

Par contre, pour la diffusion des actions effectuées sur les applications partagées, le mode déconnecté nous semble plus approprié pour garantir l'interactivité du collectif.

Tous les modules (processus) *CoopScan* sont munis de leur propre fonction de codage/décodage de messages. Cette fonction permet à chaque module de coder des informations de façon à ce que les autres sites puissent les déchiffrer et,

par conséquent, identifier le type de chaque message reçu afin de lui associer le traitement approprié.

Dans *CoopScan*, les messages ont la structure suivante : $\langle MESSAGE_CODE, message_body \rangle$. L'attribut *MESSAGE_CODE* permet d'identifier de manière unique chaque type de messages. L'attribut *message_code* représente la partie "données" du message.

Nous définissons dans *CoopScan* deux primitives d'émission :

- *SendMessage* (*msg*, *dest*, *mode*) : permet d'émettre un message *msg* vers une destination *dest* selon le mode de communication *mode*. Le paramètre *mode* peut prendre deux valeurs : *C* (pour Connecté) ou *D* (pour Déconnecté), pour désigner respectivement les protocoles : *TCP* ou *UDP*. Dans le cas où l'on utilise le protocole *TCP/IP*, la fonction *SendMessage* retourne un entier désignant le nombre d'octets émis dans le cas où la communication s'est correctement déroulée. Le cas échéant, la fonction retourne une valeur négative.

Lors des échanges d'informations de contrôle (données de l'espace de coordination) et, particulièrement pour l'exécution du protocole de connexion, la fonction *SendMessage* est utilisée en mode gardé. Il s'agit de vérifier à l'issue de chaque émission le retour de la fonction, et, en cas d'erreur, de ré-émettre de nouveau jusqu'à l'expiration d'un délai de *garde*.

Au terme d'un certain nombre d'émissions incorrectes (si *G_SendMessage* retourne une valeur négative), le site destinataire est déclaré défectueux et une procédure d'exclusion est lancée à son encontre.

Il est à noter que le mode gardé de la fonction *SendMessage* utilise obligatoirement le protocole *TCP*.

- *DiffuseMessage* (*msg*, *dest_list*, *mode*) : la fonction *DiffuseMessage* permet la diffusion d'un message vers une liste de destinations moyennant des communications point-à-point utilisant la fonction *SendMessage*.

De même que *G_SendMessage*, cette fonction peut également être utilisée en mode *gardé*. Dans ce cas, la fonction de diffusion permet de donner en retour la liste des sites vers lesquels la communication aura échoué.

V.2.1.2 L'agent session

L'AgS est représenté par un processus Unix qui est lancé dès l'initialisation de *CoopScan* (AgS dans l'état de veille).

L'AgS fournit au participant une interface-utilisateur graphique construite à l'aide du langage de programmation et boîte à outils *Tcl/tk* [Welch 95]. À partir

de cette interface, le participant peut facilement configurer la session au lancement (*état de veille*), activer des commandes de contrôle dès que la session est créée, et être continuellement informé de l'évolution de la session (par exemple, visualiser la liste des participants, visualiser la liste des applications selon leur utilisation). La conception de l'interface-utilisateur est conforme aux propriétés IHM devant être fournies par une application interactive multi-utilisateurs [Salber 95] : l'observabilité, l'honnêteté, la conformité et la stabilité du temps de réponse. La définition de ces propriétés est donnée dans le chapitre I.

La Fig. 5.2 illustre l'interface fournie par l'AgS quand il est dans un état de veille. Les trois boutons permettent respectivement de :

- créer une nouvelle session,
- demander à se connecter à une session existante,
- visualiser les sessions existantes.

Fig. 5.2 : L'interface de lancement de CoopScan.

En activant le bouton *NewSession*, une fenêtre de dialogue s'affiche afin que l'utilisateur (qui est devenu automatiquement président) précise les paramètres de lancement, et donc configure la session. Le président peut choisir la politique d'attribution du droit de parole et la politique adoptée pour gérer la connexion de nouveaux participants.

Fig. 5.3 : L'interface de configuration.

Trois politiques pour la gestion du droit de parole sont fournies par l'application. Les politiques *Designation* et *Modération* consistent à attribuer le jeton de manière explicite à un participant. La différence vient du fait que dans la première (*Désignation*) l'actuel détenteur du jeton choisit son successeur alors que dans la seconde, le président désigne les intervenants.

Selon la politique FIFO, l'attribution du jeton est implicite. C'est-à-dire que le participant libère le jeton et que le système se charge de désigner un successeur en tenant compte de l'ordre de réception des requêtes pour l'obtention du jeton préalablement reçues par le site.

Le président a également le choix de la politique adoptée pour répondre aux requêtes de connexion. Deux politiques sont fournies par *CoopScan*. Dans la première, le président supervise la connexion alors que dans l'autre la connexion est totalement transparente. Techniquement, les deux politiques sont mises en œuvre par le même protocole de connexion moyennant des petites modifications.

Afin de rejoindre une session en cours, l'utilisateur peut activer la commande *Join Session* ou scruter le réseau à l'aide de la commande *Session Directory*, en attente de la prochaine session *CoopScan*.

– *Join Session* : si l'utilisateur choisit de rejoindre une session en cours, l'AgS affiche la boîte de dialogue illustrée par la Fig. 5.4. L'utilisateur désigne alors l'adresse de l'AgS président, i.e. le créateur de la session, sachant que le président peut changer depuis le début de la session. Le protocole de connexion prend en compte ce cas de figure.

Fig. 5.4 : L'interface de connexion activée par la commande Join Session.

– *Session Directory* : en activant cette commande, l'utilisateur s'abonne à un processus *démon* dont la fonction est d'informer tous ceux qui s'y sont abonnés du lancement d'une nouvelle session *CoopScan*.

Fig. 5.5 : L'Abonnement au démon Session Directory

La Fig. 5.5 illustre l'interface activée par la commande *Session Directory* quand aucune session n'est en cours d'exécution.

La Fig. 5.6 illustre l'annonce du lancement d'une session *CoopScan* sur le port 7777 de la machine *zanoubia*.

Fig. 5.6 : La création d'une nouvelle session

À l'issue de l'exécution du protocole de connexion, l'AgS affiche l'interface principale de *CoopScan* illustrée par la Fig. 5.7.

Les informations qui y sont affichées sont les suivantes :

- la liste de tous les participants de la session. Cette liste est affichée en haut de la fenêtre. L'ajout ou le retrait d'un participant est immédiatement répercuté sur cette liste.
- *On Tools* désigne la liste des applications de la session,

Fig. 5.7 : L'interface utilisateur de l'AgS.

- *In Tools* désigne la liste des applications de travail,
- *Floor Holder* et *Chairman* désignent respectivement le détenteur du jeton et le président de la session,
- *Floor Control* : si le participant est le détenteur du jeton, l'activation de ce bouton se traduit par la libération du jeton conformément à la PDP adoptée lors de la configuration de la session. Par contre, si le participant n'a pas le jeton, l'activation du bouton se traduit par l'émission d'une requête pour l'acquisition du droit de parole,
- *New Tool* : ce bouton permet de lancer la fonction *AddNewTool*. L'activation du bouton permet de visualiser la liste des applications disponibles dans la session afin d'en choisir une et l'activer,
- *Join/Drop* : ce bouton permet de rejoindre une application de la session (*Join*) ou de quitter une application de travail. L'activation de cette

commande lance l'exécution du protocole de connexion ou le protocole de déconnexion pour l'application sélectionnée dans l'une des deux listes : *On Tools* ou *In Tools*,

- Le bouton portant l'image d'une *tête d'homme* ne peut être activé que par le président. L'activation du bouton permet au président de récupérer le droit de parole, de désigner un nouveau président, ou de reconfigurer la session en cours (cf. Fig. 5.8).

Fig. 5.8 : Les fonctions réservées au président

V.2.1.3 L'éditeur partagé construit à partir de Thot

Le schéma d'intégration de l'éditeur *Thot* est détaillé dans le chapitre II, dans le cadre de l'étude des architectures pour la construction d'applications coopératives, construction qui est fondée sur l'intégration d'applications *ouvertes* (cf. II.2).

L'application *Thot* est utilisée dans *CoopScan* pour fournir aux acteurs d'une session un support de coopération synchrone moyennant des documents multimédia, structurés. Les documents qui font l'objet de la coopération sont préalablement installés, de manière à ce que chaque participant de l'application puisse y accéder directement sur son site ou à travers des montages *NFS*⁽³⁾.

Fonctions fournies par l'application

L'application permet à un groupe de participants de partager plusieurs documents *Thot* moyennant un mode d'interaction synchrone. Les fonctions fournies par l'application sont :

- la *gestion des documents partagés* : les documents ouverts par le détenteur du droit de parole sont systématiquement ouverts sur tous les sites participant à l'application. La liste des documents ouverts au cours d'une session, ainsi que la liste des opérations d'écriture concernant chaque document ouvert (actions susceptibles de modifier le contenu d'un

(3) En anglais, *Network File System*

document), sont dupliquées sur tous les sites participants à l'application. La liste des opérations effectuées sur chaque document permet la reconstitution du contexte lors de la connexion d'un nouveau membre,

- les fonctions d'édition : quasiment toutes les fonctions d'édition fournies par l'application *Thot* sont rendues coopératives. Les commandes effectuées par chaque participant sont automatiquement interceptées par l'AgL. En fonction des droits attribués au participant, l'AgL permet l'exécution de la commande et la diffuse vers les autres sites, ou interdit l'action et avertit le participant. Les commandes autorisées par l'AgL sont diffusées vers tous les AgDs *Thot* de la session afin que l'action du participant puisse être exécutée sur les autres sites.

La liste des documents partagés constitue l'espace de coopération défini par l'éditeur *Thot*. L'AgL a la charge de maintenir cet espace cohérent afin de garantir le bon fonctionnement des services de gestion de groupes.

L'AgL *Thot*

Il s'agit du module *AgL* connecté à l'application *Thot* conformément au schéma illustré par la Fig. 5.1. L'AgL est abonné au service de notification *ECF*⁽⁴⁾ fourni par *Thot*. Ce service déclenche l'émission d'événements en réaction à des commandes menées sur l'interface-utilisateur de l'application (telles que le positionnement du curseur dans un document, l'inversion de texte, la destruction d'un élément, ...). L'*ECF* permet donc à des programmes d'application associés à l'éditeur de réagir à des commandes utilisateurs. Dans *CoopScan*, l'AgL joue le rôle de l'un de ces programmes.

L'abonnement se fait en associant aux événements émis par l'*ECF* des fonctions de traitement de l'AgL par le biais d'un langage de description de schémas d'interface, appelé langage *I*. Par ailleurs, un schéma d'interface *Thot* est relié à un schéma de structure décrivant la structure logique d'un document. Le langage *I* permet ainsi de définir des actions associées à des types d'éléments définis dans un schéma de structure. En conséquence, des réactions différentes peuvent être associées à la même commande s'appliquant à des schémas de structure différents (par exemple, schéma *Article*, schéma *Lettre*, ou des schémas de structure construits par l'utilisateur).

L'*ECF* délivre deux types d'événements : **Pre**, et **Post**. Les événements **Pre** précèdent l'exécution effective de la commande, alors que les événements de type **Post** succèdent à son exécution. L'exécution effective de la commande

(4) *External Call Facilities* ; pour désigner des mécanismes d'appels externes.

dépend de la valeur *booléenne* retournée par la fonction reliée à l'événement de type **Pre** (*True* : pas d'exécution de l'action, *False* : exécution de l'action).

L'AgL exploite ce mécanisme en associant à chaque événement de type **Pre** la fonction *CheckFloor*. Cette fonction permet de vérifier le droit de parole attribué au participant.

Après l'exécution d'une commande, un événement de type **Post** est délivré par l'ECF. L'événement renferme une structure de données contenant tous les paramètres de l'action exécutée (identificateur du document, le type de l'action, ...). L'AgL associe aux événements de type **Post** le traitement suivant :

- Codage : la fonction *CodeAction* permet de coder l'action exécutée à l'intérieur d'un message,
- Diffusion : l'AgL diffuse le message codé par *CodeAction* vers l'ensemble des *AgDs* *Thot* de la session,

L'AgD *Thot*

Sur les autres sites, chaque *AgD* reçoit le message diffusé par l'AgL. Les informations contenues dans le message permettent à l'*AgD* de reconstituer la commande exécutée sur le site de l'AgL. Pour cela, l'*AgD* associe à chaque type d'actions un traitement approprié qui consiste à déclencher les fonctions de l'API capables de reproduire la commande utilisateur qui est à l'origine du message reçu. En résumé, l'*AgD* associe aux messages reçus, le traitement suivant :

- décodage : afin de récupérer les paramètres de l'action diffusée, l'*AgD* décode les messages à l'aide de la fonction *DecodeAction*,
- les paramètres récupérés constituent les paramètres d'entrée des fonctions de l'API *Thot*.

V.2.1.4 Le navigateur Coopératif *CoopGrail*

Présentation générale

CoopGrail est une application partagée construite à partir du navigateur *World Wide Web Grail* [van Rossum 96]. Ce navigateur est similaire aux navigateurs *Netscape* et *Mosaic*. Il supporte les protocoles et les formats communément trouvés sur le *Web* (tels que *HTTP*, *FTP* est *HTML*).

CoopGrail permet à des utilisateurs distribués de participer à des sessions de navigation collectives sur le *Web*. Le détenteur du droit de parole dirige la visite, et propose aux autres participants des *URLs*⁽⁵⁾ qu'il récupère dans son espace de travail privé.

(5) *Uniform Adress Ressources*

Fig. 5.9 : Interface principale de l'application CoopGrail

Nous adoptons pour *Grail* une approche similaire à celle adoptée pour l'intégration de *Thot*. Nous construisons, à partir de *Grail*, une *application intégrée* complètement autonome qui est capable d'émettre des événements vers

l'"extérieur" (i.e. vers d'autres modules s'exécutant sur le même site ou sur des sites distants), et qui peut être "pilotée" à partir de modules externes (*AgDs*).

AgL Grail

Grail n'offre pas vraiment de mécanisme spécifique pour la transmission d'événements. Toutefois, nous disposons des fichiers sources de l'application. Ce qui nous permet de développer notre propre service de notification, l'idée étant de construire un module générique utilisable par plusieurs applications extérieures.

Le service de notification consiste à associer un événement à chaque action sur l'interface-utilisateur de l'application. Les événements sont transmis vers un module particulier. À l'intérieur de ce module, nous rattachons à chaque type d'événements (donnant tous les paramètres de la commande) une procédure de traitement.

L'*AgL* est complètement décrit dans le langage *Python*. Sa fonction consiste à récupérer tous les événements produits par le service de notification et de vérifier leur validité. La fonction de notification est implantée par une méthode de la classe décrivant l'*AgL*. La définition des classes *AgL/AgD* est donnée dans l'Annexe A de ce mémoire.

À la réception d'une notification, l'*AgL* adopte le même fonctionnement qu'il adopte pour l'éditeur *Thot* : il vérifie les droits attribués au participant à l'aide de la fonction *CheckFloor*. Si le participant est le détenteur du jeton alors il code l'action dans un message et ensuite diffuse le message.

Afin de garantir la gestion dynamique de groupes, l'*AgL* gère l'espace de coopération défini par l'application *Grail*. Il s'agit de l'ensemble des navigateurs *WWW* lancés en mode coopératif dans la session *CoopScan*. Cet espace est représenté par un dictionnaire de ressources (*browser_dict*) dont chacune est référencée de manière unique sur le site. Chaque site participant détient une copie du dictionnaire de ressources. Par ailleurs, l'*AgL* gère un dictionnaire de ressources privé (*local_dict*) désignant son espace de travail privé. En mode coopératif, toutes les commandes sont menées sur l'espace de coopération.

En plus de ces fonctions classiques, nous définissons de nouvelles fonctions de l'*AgL Grail*. Il s'agit de fonctions coopératives spécifiques à l'application qui permettent de visualiser la liste des participants à partir de l'interface de l'application, d'envoyer des messages vers un utilisateur, de diffuser un message vers tous les utilisateurs, de quitter le mode coopératif pour un mode mono-utilisateur, de rejoindre la session à partir de ce mode en récupérant

l'espace de travail privé, d'intégrer des données de l'espace privé dans l'espace de coopération.

AgD Grail

Grail ne fournit pas de bibliothèques externes (*API*) permettant de "piloter" l'application comme c'est le cas avec l'*API* de *Thot*. Toutefois, *Python* étant un langage interprété, il fournit une primitive d'évaluation des commandes. Afin de reproduire des commandes à distance, il suffit d'activer la fonction d'évaluation en lui passant en paramètre la commande. Nous utilisons donc la fonction d'évaluation fournie par *Python* pour construire une *API* de l'application.

Les deux fonctions principales fournies par l'*AgD* sont les suivantes:

- la réception des messages diffusés par l'*AgL* et l'extraction des expressions,
- ensuite, l'appel de la fonction d'évaluation qui permet de reconstituer l'action.

Fonctions fournies par l'application

- les fonctions coopératives : il s'agit des fonctions du navigateur que nous rendons coopératives. Ces fonctions permettent d'agir sur les données partagées de l'application, les différentes instances du navigateur ouvertes en mode coopératif (*browser_dict*). Les fonctions coopératives sont essentiellement celles qui permettent la création de nouvelles instances du navigateur et celles réalisant les fonctions de navigation :

NewWindow : cette fonction permet d'ouvrir une nouvelle instance du navigateur. Chaque nouvelle instance sera immédiatement insérée dans une table contenant les identificateurs des navigateurs coopératifs créés depuis le début de la session.

OpenFile : elle permet d'ouvrir un fichier sur l'une des instances du navigateur.

Les opérations de navigation classiques (OpenLocation, Go, etc...),

Le passage d'un mode coopératif à un mode mono-utilisateur : le participant a la possibilité de quitter *CoopGrail*, en remplissant les conditions du protocole de déconnexion (céder le droit de parole). L'opération peut être réalisée de deux manières : en quittant définitivement *CoopGrail* ou en basculant en mode mono-utilisateur.

- les fonctions du navigateur à caractère privé. Il s'agit des fonctions dont les effets ne sont pas "répercutés" sur les autres sites. Le terme répercuté est employé entre guillemets afin de préciser que les messages

envoyés entre sites ne contiennent pas de résultats d'exécution mais des appels de fonctions qui devront être exécutées sur les sites distants. Parmi ces fonctions nous citons :

ViewSource : elle permet la visualisation du source d'un document html,

SaveAs : elle permet l'enregistrement d'un document dans l'espace de travail personnel,

Print : elle lance l'impression d'un document,

AddBookmark : elle ajoute une adresse *URL* chargée dans l'une des instances coopératives du navigateur dans la liste des adresses d'*URL* privées de l'utilisateur local. Cette fonction permet aux utilisateurs de l'application d'échanger, d'une façon implicite, des *URLs* de sites visités durant la coopération.

V.3 Bilan

Deux implantations de CoopScan sont issues de la description OLAN : l'une sur une plate-forme Unix et l'autre sur un système distribué.

Les deux prototypes ont fait l'objet de plusieurs utilisations, notamment au sein de notre laboratoire. Le prototype développé sur la plate-forme *OODE* a fait l'objet d'une démonstration dans le cadre d'un concours organisé par l'*AF-CET (AFCET'95)* récompensant le meilleur transfert technologique.

Le prototype actuel (développé sur une plate-forme Unix) intègre également le navigateur coopératif *CoopGrail*. Il fournit un service de gestion de groupes permettant l'utilisation de *CoopScan* par un nombre variable de participants. Le prototype actuel a été testé dans une configuration de plusieurs machines Unix connectées par un réseau local.

Nous retenons de ces expérimentations les observations suivantes :

- l'interactivité de l'éditeur *Thot* et du navigateur *Grail* ont été préservée. Ceci est essentiellement dû au schéma totalement répliqué adoptée dans *CoopScan*,
- les temps de notification (temps mis par une action pour être répercutée sur les autres sites) sont acceptables. Ces temps ne font pas l'objet d'une évaluation. Toutefois, ces temps sont relativement bas dans une configuration de réseau local,
- Pour *Thot*, la transmission des actions vers les autres sites est tributaire de la granularité des événements fournis par l'*ECF* de *Thot*. Par

exemple, la version actuelle de l'*ECF* ne fournit pas d'événements pour l'insertion de chaque caractère. L'*ECF* génère l'événement de type Post, relatif à l'insertion d'une chaîne, à la fin de l'opération d'insertion. La fin de l'insertion est détectée par l'éditeur quand l'utilisateur pointe avec son curseur un autre élément du document. Dans ce cas, la transmission de l'action vers les autres sites ne peut être effectuée qu'à ce moment.

- l'interactivité du navigateur *CoopGrail* était comparable à celle enregistrée pour le mode d'utilisation privé de *Grail*. Toutefois, les temps de réponse absolus sont plutôt élevés du fait que les commandes de l'application sont interprétées,
- le passage à un mode d'utilisation privé du navigateur s'avère très utile. Au lieu de quitter complètement l'application et relancer *Grail*, le participant bascule dans un mode privé, bénéficie de toutes les URLs qu'il aura collectés lors de la coopération, et continue sa navigation en solitaire,
- l'absence d'autres applications partagée dans *CoopScan*, notamment l'absence d'un outil de communication intégré dans *CoopScan* (canaux audio/vidéo). Afin de remédier à ce manque, l'une des expérimentations de *CoopScan* intégrant l'éditeur *Thot* a été réalisée en utilisant l'outil d'audio conférence *IVS* [Turletti 94]. Cet outil n'est pas intégrée dans *CoopScan*, mais uniquement lancé en parallèle à l'application.

V.4 Conclusion

CoopScan est décrite à l'aide d'un modèle d'architecture générique mettant en œuvre des composants logiciels communiquant ; l'*agent session*, l'*agent local* et l'*agent distant*.

Ce modèle d'architecture répond aux objectifs que nous nous sommes fixés au début de ce travail, concernant la généricité de la plate-forme, et ce à différents niveaux. Ce chapitre approfondit particulièrement l'aspect construction et mise en œuvre.

L'architecture *CoopScan* a été implantée sur deux types de plates-formes différentes. Nous avons entrepris ce "portage" afin de bien vérifier la généricité de notre plate-forme de ce point de vue. En partant de la description *OLAN* de l'architecture, le travail de déploiement sur différentes plates-formes est essentiellement fondé sur l'instanciation des connecteurs entre les composants. Les composants sont instanciés par les entités actives fournies par la plate-forme (i.e. processus pour *Unix*, et objets pour la plate-forme *OODE*).

CoopScan permet également de valider l'étude des architectures logicielles pour collecticiels (étude menée dans le chapitre II). Nous y utilisons un schéma d'intégration d'applications à deux niveaux : au niveau du système de fenêtre et au niveau des mécanismes fournis dans les applications ouvertes (i.e. *API* et service de notifications).

Les différentes expérimentations de *CoopScan* nous laissent apprécier le gain, en termes de réduction de coût de développement, apportés par l'intégration d'applications existantes et la réutilisation de modules *coopératifs*.

Toutefois, à l'issue de ces expérimentations, *CoopScan* affiche certains manques, notamment en ce qui concerne les services garantissant la qualité de service d'un point de vue synchronisation entre les flots de données échangés. La plate-forme ne fournit pas de protocole de gestion de la concurrence permettant un accès libre aux applications.

Par ailleurs, nous continuons nos travaux en vue d'enrichir la bibliothèque de PDP par des protocoles de gestion de la concurrence et les services de gestion de groupes (SGG) par des protocoles adaptés à un plus large spectre d'applications, notamment les applications de téléconférence.

Chapitre V

Réalisations

V.1 Introduction	145
V.2 Réalisations	145
V.2.1 Déploiement de <i>CoopScan</i> sur une plate-forme Unix	146
V.2.1.1 L'instanciation des connecteurs <i>OLAN</i> sur une plate-forme Unix	146
V.2.1.2 L'agent session	148
V.2.1.3 L'éditeur partagé construit à partir de Thot	153
V.2.1.4 Le navigateur Coopératif <i>CoopGrail</i>	155
V.3 Bilan	159
V.4 Conclusion	160

Chapitre VI

Conclusion

Pour terminer ce mémoire, nous faisons ici le bilan de l'étude réalisée. Nous résumons les apports de ce travail et dégageons un certain nombre de perspectives de recherche pour la poursuite du développement d'architectures systèmes pour la construction et l'exécution d'applications coopératives.

VI.1 Conclusion

C'est en nous intéressant aux outils et aux langages pour le développement d'applications réparties, notamment les applications coopératives, que nous avons étudié pendant la durée de cette thèse les architectures logicielles et systèmes pour la construction et l'exécution de collecticiels synchrones. L'étude que nous avons réalisée couvre deux aspects :

- structurel concernant l'architecture logicielle des applications et l'architecture des plates-formes systèmes sous-jacentes pouvant supporter et favoriser leur déploiement et leur exécution,
- fonctionnel concernant les protocoles de contrôle fournis par les collecticiels afin de permettre la création et la participation à des tâches coopératives, le partage d'un ensemble de ressources communes (documents, images, ...), et l'interaction selon un mode temps-réel.

D'un point de vue conceptuel, notre étude diffère des travaux visant la modélisation de collecticiels par son caractère mixte dont l'objectif est de rattacher une architecture logicielle *générique* à un fonctionnement "à la carte" en fonction des besoins exprimés par les utilisateurs.

Nous avons commencé notre étude en faisant un rappel de la terminologie employée dans le domaine du collecticiel, en donnant la définition des concepts fondamentaux tels que le mode d'interaction synchrone d'un collecticiel, le mode de fonctionnement *WYSIWIS*, et les taxonomies les plus répandues, qui sont

communément utilisées pour répertorier les collecticiels. Ensuite, notre étude se scinde en deux parties complémentaires :

- *L'architecture* : nous avons étudié conjointement dans le chapitre II et le chapitre III les architectures logicielles des collecticiels ainsi que les environnements de mise en œuvre favorisant leur déploiement.

Pour construire des collecticiels, nous adoptons une approche fondée sur l'intégration d'applications existantes. Cette approche nous paraît la plus appropriée s'agissant d'applications qui, à la base, sont des applications présentant des caractéristiques *IHM*. La *flexibilité* d'utilisation et la large diffusion des outils utilisés pour coopérer constituent donc nos hypothèses de départ.

- *Les protocoles de contrôle* : il s'agit de fonctions coopératives fournies par un collecticiel en plus des fonctions communément fournies par une application mono-utilisateur. Nous avons groupé ces fonctions dans deux classes de protocoles : des protocoles de gestion de groupes, et des protocoles pour contrôler l'accès aux ressources partagées.

Nous nous sommes fixé le but de fournir un modèle d'architecture générique pour la construction de collecticiel. Cet objectif a été atteint à travers la conception et la mise en œuvre d'une plate-forme nommée *CoopScan*, pour la construction et l'exécution de collecticiels synchrone, et ce en fournissant une *généricité* recouvrant trois aspects :

1) la construction : nous avons proposé un modèle d'architecture générique fondée sur une structuration modulaire du collecticiel. Le modèle permet l'intégration d'applications s'exécutant sous le système de fenêtres X-Windows, et d'applications dites ouvertes fournissant une API et un service de notification.

2) le fonctionnement : il est régi par deux familles de protocoles dont les fonctions sont le contrôle d'accès aux données partagées et la gestion dynamique de groupe dont l'objectif est essentiellement de garantir l'adhésion du groupe. Pour les protocoles de contrôle d'accès, *CoopScan* fournit des politiques de gestion du droit de parole. Ces politiques sont fondées sur un mode d'accès exclusif.

Par ailleurs, *CoopScan* fournit des protocoles pour la connexion-déconnexion dynamiques de participants et pour la gestion des pannes. Pour la connexion, il s'agit de protocoles à deux phases, fondés sur le concept de "gel", le but étant de garantir une vue identique à tous les membres du groupe à l'issue de l'arrivée d'un nouveau participant.

Ces protocoles sont regroupés de façon modulaire dans deux familles de protocoles respectivement : *PDP*, *SGG*. Ces protocoles sont également dynamiquement instanciables pour plusieurs applications partagées du même collecticiel. En fait, dans *CoopScan*, il est possible d'appliquer des protocoles de gestion du droit de parole ou des protocoles de gestion dynamique de groupes indifféremment de l'application partagée,

3) *la plate-forme d'exécution* : nous utilisons pour la description de *CoopScan* un langage et un modèle à base de composants. Les différents modules de *CoopScan* y sont représentés par des composants logiciels communiquant à travers des connecteurs. Nous utilisons ce modèle pour mettre en valeur la généralité de *CoopScan* et son indépendance vis-à-vis du support d'exécution. Nous en avons donné la preuve pratique en déployant *CoopScan* sur deux plates-formes systèmes différentes : une plate-forme *Unix* et une plate-forme à objets répartis.

VI.2 Perspectives

Au chapitre des perspectives concernant le développement de collecticiels, certaines voies sont à achever alors que d'autres nous paraissent intéressantes à explorer. Nous détaillons ici ces travaux à court ou à moyen terme.

D'un point de vue expérimental, la plate-forme *CoopScan* est actuellement déployée sur deux types de plates-formes. Les applications qui y sont intégrées le sont conformément aux deux approches d'intégration (*bas vs haut*). Toutefois, un manque d'outils audio-vidéo synchronisés se fait ressentir dans des utilisations complètement réparties du collecticiel. L'enrichissement de la plate-forme par de nouvelles applications permettant l'établissement de communications audio-vidéo figure parmi nos objectifs à court terme.

Nous avons proposé dans *CoopScan* des protocoles de contrôle d'accès exclusif et des services de gestion de groupes offrant des protocoles de connexion dynamique exclusifs (exécutés par un site à la fois). Notre objectif est d'augmenter ces protocoles en fournissant des protocoles de gestion de la concurrence permettant un accès libre et des services de gestion de groupes dans lesquels la connexion de nouveaux membres puisse être décentralisée.

Par ailleurs, nous pensons que l'étude des architectures pour collecticiels synchrones représente un grand intérêt pour le développement des outils pour l'administration d'applications et de systèmes répartis. L'un de nos objectifs est d'enrichir le modèle *OLAN* par des fonctions spécifiques aux collecticiels

synchrones que l'administrateur ou le concepteur pourra choisir en fonction du réseau, des machines, et d'autres paramètres de configuration. L'apport que nous souhaitons fournir se situe à deux niveaux :

- la construction de composants ré-utilisables : il s'agit de fournir une librairie de composants "*applications partagées*" intégrant des applications de différentes sortes,

- la construction d'une librairie de connecteurs adaptés aux besoins des collecticiels synchrones. S'agissant d'applications dans lesquelles plusieurs types de média sont échangés, il est souhaitable de disposer de protocoles de communications qui soient configurables en fonction des classes de média véhiculés. Actuellement, dans *CoopScan* nous en recensons deux : les médias de contrôle (espace de coordination), et les médias de coopération (espace de coopération). Nous espérons en fournir plusieurs afin de satisfaire les besoins d'un large spectre de collecticiels synchrones.

Chapitre VI

Conclusion

VI.1 Conclusion	163
VI.2 Perspectives	165

Bibliographie

- [Bowers and Benford 91] J. M. Bowers and S. D. Benford, “Studies in Computer Supported Cooperative Work: Theory, Practice, and Design”, *Proceedings of the First European Conference on Computer Supported Cooperative Work*, North Holland 1991.
- [Coleman 92] D. Coleman, R. Shapiro, “Defining Groupware”, *Special Advertising Section to Network World*, Juin 1992.
- [Coleman 95] D. Coleman, *groupware: Technology and Applications*, Prentice Hall, 1995.
- [Dyson 92] E. Dyson, “A Framework for Groupware”, In [Coleman 92], pp. 10–20.
- [Ellis and al 91] C. A. Ellis, S. J. Gibbs, G. L. Rein, “Groupware: Some Issues and Experience”, *Communications of the ACM*, 34(1), pp. 38–58, Janvier 1991.
- [Ellis and Wainer 94] C. A. Ellis, J. Wainer, “A Conceptual Model of Groupware”, *CSCW’94 Conference on Computer Supported Cooperative Work*, pp. 79–88, ACM, Chapel–Hill, North Carolina, USA, 1994.
- [Greenberg 91] S. Greenberg, “Computer–Supported Cooperative Work and Groupware”, Academic Press.
- [Grudin 91] J. Grudin, “The convergence of Two Development Paradigms”, *Proceedings CHI’91*, ACM, pp. 91–97, 1991.
- [Hiltz 84] S. R. Hiltz, “Online Communities: A Case Study of the Office of the Future”, Ablex, 1984.
- [Hiltz and Turoff 78] S. R. Hiltz and M. Turoff, “The Network Nation: Human Communication via Computer”, *Addison–Wesley*, 1978.
- [Johnson–Lenz 80] P. Johnson–Lenz, T. Johnson–Lenz, “Groupware: The emerging Art of Orchestrating Collective Intelligence”, *Presented at the First Global Conference on the Future*, Toronto 1980.
- [Johnson–Lenz 81] P. Johnson–Lenz, T. Johnson–Lenz, “Consider the Groupware: Design and Group Process Impacts on Communication in the Electronic Medium”, In Hiltz S. R. and Kerr, E. B. (1981) *Studies of*

computer-Mediated Communication Systems: A Synthesis of the Findings. Final Report of a Workshop, Computerized Conferencing and Communications center, New Jersey Institute of Technology.

- [Johnson-Lenz 82] P. Johnson-Lenz, T. Johnson-Lenz, "Groupware: The process and Impacts of Design Choices", *In [Kerr and Hiltz 1982]*, pp. 45-55, 1982.
- [Johnson-Lenz 92] P. Johnson-Lenz, T. Johnson-Lenz, "Groupware in Computer-Mediated Culture: Some Keys to Using it Wisely", *in [Coleman 92]*, pp. 130-132, 1992.
- [Kerr and Hiltz 82] E. B. Kerr and S. R. Hiltz, "Computer-Mediated Communication Systems: Status and Evaluation", Academic Press.
- [Salber 95] Daniel Salber, *De l'interaction homme-machine individuelle aux systèmes multi-utilisateurs*, Thèse de troisième cycle, Université Joseph Fourier - Grenoble 1, Septembre 1995.
- [Sproull and Kiesler 91] L. Sproull and S. Kiesler, "Connections: New Ways of Working in the Networked Organization", The MIT Press 1991.
- [Abdel-Wahab 91] H. Abdel-Wahab, M. A. Feit, "XTV: A Framework for Sharing X Windows Clients in Remote Synchronous Collaboration", *Proceedings of IEEE Tricomm'91: Communications for Distributed Applications & Systems*, IEEE, Chapel-Hill NC, April 1991.
- [Wurtz 94] A. Wurtz, D. Scherer, T. Murer, A. Helbling, "Development of an Architecture for Integrated, Distributed Software Development Systems", *In Proceedings of SPP Information Conference on Secure Distributed Systems*, 1994.
- [Baecker 94] R. Baecker, G. Glass, A. Mitchell, I. Posner, "SASSE: The Collaborative Editor", *In proceedings of ACM Conference on Human Factors in Computing Systems*, Volume 2, pp. 459-460, 1994.
- [Bannon 91] L. J. Bannon, K. Schmidt, *Four Characters in Search of a Context*, *In Studies in Computer Supported Cooperative Work: Theory, Practice and Design*, Eds. J. M. Bowers and S. D. Benford, Elsevier, 1991.

- [Bignoli 89] C. Bignoli, C. Simone, *AI Techniques for Supporting Human To Human Communications in CHAOS*, In *Studies in Computer Supported Cooperative Work: Theory, Practice and Design*, Eds J. M. Bowers, S. D. Benford Elseiver, 1989.
- [Coleman 95] D. Coleman, *groupware: Technology and Applications*, Prentice Hall, 1995.
- [Crowley 90] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, R. Tomlinson, ‘‘MMconf: An Infrastructure for Building Shared Multimedia Applications’’, *CSCW’90*, pp. 329–342, Los Angeles, October 1990.
- [Decouchant 94] D. Decouchant, ‘‘R troaction de Groupe et Edition Coop rative de Documents Structur s’’, *IHM’94*, Lille, France, Septembre 1994.
- [Decouchant 95] D. Decouchant, V. Quint, M. Romero, ‘‘Structured Cooperative Authoring on the World–Wide Web’’, *World Wide Web Journal, 4th International Worl–Wide Web Conference Proceedings*, I. F. Cruz, J. Marks and K. Wittenburg, ed, pp. 657–666, O’Reilly & Associates, Boston, December 1995.
- [Durfee 89] E. H. Durfee, V. R. Lesser, D. D. Corkill, ‘‘Trends in Cooperative Distributed Problem Solving’’, *IEEE Transactions on Knowledge and Data Engineering*, 1(1), pp. 63–83, Mars 1989.
- [Ellis 91] C. A. Ellis, S. J. Gibbs, G. L. Rein, ‘‘Groupware: Some Issues and Experience’’, *Communications of the ACM*, 34(1), pp. 38–58, January 1991.
- [Ellis 94] C. A. Ellis, J. Wainer, ‘‘A Conceptual Model of Groupware’’, *CSCW’94 Conference on Computer Supported Cooperative Work*, pp. 79–88, ACM, Chapel–Hill, North Carolina, USA, 1994.
- [Greenberg 95] S. Greenberg, C. Gutwin, A. Cockburn, ‘‘Sharing fisheye views in relaxed–WYSIWIS groupware applications’’, *Proceedings of Graphics Interface, Distributed by Morgan kaufmann*, pp. 22–24, Toronto 1995.

- [Gutwin 95] C. Gutwin, S. Greenberg, “Support for Awareness in Real-time Desktop conferences”, *Proceedings of the 2nd New Zealand Computer Science Research Students’ Conference, University of Waikato, Also as Report 95/549/01 Dept of Computer Science, University of Calgary*, Avril 1995.
- [Hahn 91] U. Hahn, M. Jarke, “Teamwork Support in a Knowledge-Based Information System Environment”, *IEEE Transactions on Software Engineering*, 17(5), pp. 467–482, Mai 1991.
- [Jansen 95] B. Jansen, D. Severson, M. Spreitzer, *Inter-Language Unification (ILU)*, Xerox Corporation, March 1995.
- [Johnson 93] P.M. Johnson, D. Tjahjono, “Improving Software Quality through Computer Supported Collaborative review”, *In Proceedings of the third European Computer Supported Cooperative Work, Eds G. de Mechlis, C. Simone, K. Schmidt, Kluwer Academic Publishers*, pp. 61–76, Septembre 1993.
- [Johansen 84] R. Johansen, *Teleconferencing and Beyond: Communications in the Office of the Future*, MacGraw-Hill, New York 1984.
- [Karsenty 93] A. Karsenty, M. Beaudouin-Lafon, “An Algorithm for Distributed Groupware Applications”, *In Proceedings of the 13th International Conference on Distributed Computing Systems ICDCS’93*, pp. 195–202, IEEE, Pittsburgh, Mai 1993.
- [Lamport 78] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System”, *Communications of the ACM*, 13(2), pp. 125–133, July 1978.
- [Mallone 87] T. W. Mallone, K. R. Grant, F. A. Turbak, S. A. Brobst, M. D. Cohen, “Intelligent Information-Sharing Systems”, *Communications of the ACM*, 30(5), pp. 390–402, Mai 1987.
- [Mantei 91] M. Mantei, R. Baeker, A. Sellen, W. Buxton, T. Milligan, “Experience in the Use of a media Space”, *In ACM SIGCHI Conference on Human Factors in Computing Systems*, pp. 203–208, 1991.
- [Minör 93] S. Minör, B. Magnusson, “A Model for Semi-(a)Synchronous Collaborative Editing”, *ECSCW’93*, Milano, Italie 1993.

- [Ohmori 92] T. Ohmori, K. Maeno, S. Sakata, H. Fukuoka, K. Watabe, “Distributed Cooperative Control for Sharing Applications Based on Multiparty Multimedia Desktop Conferencing System: MERMAID”, *In Proceedings of IEEE International Conference on Parallel & Distributed Systems*, pp. 538, Juin 1992.
- [Roseman 96] M. Rosenam, S. Greenberg, “Building Real Time Groupware with GroupKit, a Groupware ToolKit”, *ACM Transactions on Computer Human Interaction*, Mars 1996.
- [Salber 95] D. Salber, J. Coutaz, D. Decouchant, M. Riveill, “De l’observabilité et de l’honnêteté : le cas du contrôle d’accès dans la communication Homme–Homme médiatisée”, *IHM’95*, Toulouse, Octobre 1995.
- [Smith 89] R. Smith, T. O’Shea, C. O’Malley, E. Scalon, J. Taylor, “Preliminary Experiences with a Distributed, Multi–Media, Problem Environment”, *Proceeding of 1st European Conference on Computer Supported Cooperative Work*, Gatwik, UK, Septembre 1989.
- [Stefik 87] M. Stefik, G. Foster, D. Bobrow, K. Khan, S. Lanning, L. Schuman, “Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings”, *Communication of the ACM*, 30(1), pp. 32, Janvier 1987.
- [Turletti 94] T. Turletti, *The Inria Video Conferencing System (IVS)*, ConneXions – The Interoperability Report, 8(10), pp. 20–24, INRIA, Sophia Antipolis, Octobre 1994.
- [Bellissard 95] L. Bellissard, S. Ben Atallah, A. Kerbrat, M. Riveill, “Component–based Programming and Application Management with OLAN”, *in Proceedings of Workshop on Object–Based Parallel and Distributed Computation*, LNCS Springer Verlag (1107), pp. 290–308, Tokyo Japan 1995.
- [Ben Atallah 95] S. Ben Atallah, R. Kanawati, R. Balter, “Architecture for Synchronous Groupware Development”, *HCI 95*, Japan, Juillet 1995.
- [Benford 92] S. Benford, H. Smith, A. Shepherd, A. Bullock, H. Howidi, “Information Sharing Approach to CSCW: The Grace project”, *Computer communications*, 15(8), pp. 933–946, Octobre 1992.

- [Chung 93] G. Chung, K. Jeffay, H. Abdel-Wahab, "Accommodating Latecomers in Shared Window Systems", *IEEE Computer*, Vol(36), pp. 72-74, Janvier 1993.
- [Cosquer 94] F. J. N. Cosquer and P. Verissimo, "Survey of Selected Groupware and Supporting Platforms", *Workshop on Basic Research On Advanced Distributed Computing: from Algorithms to Systems*, ESPRIT Basic Research Project 6360(1), pp. 1-31, Octobre 1994.
- [Crowley 90] T. Crowley, P. Milazzo, E. Baker, H. Forsdick, R. Tomlinson, "MMconf: An Infrastructure for Building Shared Multimedia Applications", *CSCW'90*, pp. 329-342, Los Angeles, Octobre 1990.
- [Ellis 89] C. A. Ellis, S. J. Gibbs, "Concurrency Control in Groupware Systems", *ACM SIGMOD*, pp. 399-405, ACM, Portland, Juin 1989.
- [Ellis 91] C. A. Ellis, S. J. Gibbs, G. L. Rein, "Groupware: Some Issues and Experience", *Communications of the ACM*, 34(1), pp. 38-58, Janvier 1991.
- [Gabre 96] M. F. Gabre, C.-W. Xu, "XShare, An Environment for Sharing X-Window Applications", *Proceedings of the 11th International Conference on Computer and Their Applications (ICCA)*, pp. 147-150, San Francisco CA, 1996.
- [Garfinkel 89] D. Garfinkel, P. Gust, M. Lemon, S. Lower, "The SharedX Multi-user Interface User's Guide", *HP Research Report No STL-TM-89-9*, Hewlett Packard, Palo Alto, 1989.
- [Glicksman 93] J. Glicksman, V. Kumar, "A SHARED collaborative environment for mechanical engineers", *In proceedings of Groupware'93*, pp. 335-347, Los Altos 1993.
- [Hill 92] R. H. Hill, "The Abstraction-link View Paradigm, Using Constraints to connect user interfaces applications", *Proc. of CSCW'92*, Octobre 1992.
- [Karsenty 93] A. Karsenty, M. Beaudouin-Lafon, "An Algorithm for Distributed Groupware Applications", *In Proceedings of the 13 th Intl. Conf. on Distributed Computing Systems ICDCS'93*, pp. 195-202, IEEE, Pittsburgh, Mai 1993.

- [Lamport 78] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System”, *Communications of the ACM*, 13(2), pp. 125–133, Juillet 1978.
- [Lauwers 90] J. Lauwers, T. Joseph, K. Lantz, A. Romanow, “Replicated Architectures for Shared Window Systems: A Critique”, *In the Proceedings of the Conference on Office Information Systems ACM*, Mars 1990.
- [MacCarthy 94] J. C. MacCarthy, A. F. Monk, “Channels, conversation, cooperation and relevance: all you wanted to know about communication but were afraid to ask”, *in Collaborative Computing*, 1(1), pp. 35–60, Mars 1994.
- [Moor 85] J. H. Moor, “What is Computer Ethics ?”, *in Metaphilosophy*, 16(4), pp. 226–275, Octobre 1985.
- [Mowbray 95] T. J. Mowbray, R. Zahavi, *The Essential CORBA: Systems Integration Using Distributed Objects*, Wiley/OMG, ISBN 0471106119, 1995.
- [Ohmori 92] T. Ohmori, K. Maeno, S. Sakata, H. Fukuoka, K. Watabe, “Distributed Cooperative Control for Sharing Applications Based on Multiparty Multimedia Desktop Conferencing System: MERMAID”, *In Proceedings of IEEE International Conference on Parallel & Distributed Systems*, pp. 538, 1992.
- [Patterson 90] J. Patterson, R. D. Hill, S. L. Rohall, W. S. Meeks, “RendezVous: an Architecture for Synchronous Multi-User Applications”, *CSCW'90*, pp. 317–328, Los Angeles, CA, Octobre 1990.
- [Patterson 91] J. Patterson, “Comparing the programming demands of single- and multi-user applications”, *in Proceedings of UIST'91*, ACM Press, pp. 87–95, Hilton Head, SC, Novembre 1991.
- [Powell 96] D. Powell, “Group Communication”, *Communications of the ACM*, 39(4), pp. 50–53, Avril 1996.
- [Quint 94] V. Quint, I. Vatton, “Making Structured Documents Active”, *Electronic Publishing*, 7(2), pp. 53–74, Juin 1994.

- [Roseman 92] M. Roseman, S. Gremberg, “Groupkit: A Groupware Toolkit for Building Real–Time Conferencing Applications”, *CSCW’92*, pp. 43–58, Novembre 1992.
- [Sarin 85] S.K. Sarin, I. Greif, “Computer–based Real–time Conferencing Systems”, *Computer*, 18(10), pp. 79–89, Octobre 1985.
- [Scheifler 86] R. W. Scheifler, J. Gettys, “The X Window System”, *Acm Transactions on Computer Graphics*, Mai 1986.
- [ShowMe] <http://www.sun.co.jp:8080/events/N+1/speedShowMe/>.
- [Shu 94] L. I. Shu, W. Flowers, “Teledesign: Groupware User Experiment in Three–Dimensional Computer Aided Design”, *Collaborative Computing*, 1(1), pp. 1–14, March 1994.
- [Stefik 87] M. Stefik, G. Foster, D. Bobrow, K. Khan, S. Lanning, L. Schuman, “Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings”, *Communication of the ACM*, 30(1), pp. 32, Janvier 1987.
- [Trevor 94] Trevor, J., Rodden, T. and Mariani, J., “The use of adapters to support cooperative sharing”, in *proceedings of CSCW’94*, ACM Press, pp. 219–230, Chappel Hill, Octobre 1994.
- [Balter 91] R. Balter, J.–P. Banâtre, S. Krakowiak, *Construction des systèmes d’exploitation répartis*, INRIA – Collection Didactique, Avril 1991.
- [Bellissard 96] L. Bellissard, S. Ben Atallah, F. Boyer, M. Riveill, “Distributed Application Configuration”, In *Proceedings of 16 th Intl. Conf. on Distributed Computing Systems ICDCS’96*, IEEE, Hong Kong, Mai 1996.
- [Bentley 95] R. Bentley, Th. Horstmann, K. Sikkell, J. Trevor, “Supporting collaborative information sharing with the World–Wide Web: The BSCW Shared Workspace System”, *Proceedings of the 4th International WWW Conference*, Boston Decembre 1995.
- [Berners–Lee 96] T. Berners–Lee, R. T. Fielding, H. F. Nielsen, “HyperText Transfer Protocol — HTTP/1.0”, *Internet draft* (<http://www..w3.org/pub/WWW/Protocols/HTTP/1.0/spec.html>), Février 1996.

- [Dilley 95] J. Dilley, “OODCE: A C++ Framework for the OSF Distributed Computing Environment”, *In Proceedings of the Winter Usenix Conference*, USENIX Association, Janvier 1995.
- [Frivold 94] R. Frivold, R. Lang, M. Fong, “Extending WWW for synchronous collaboration”, *In Electronic Proceedings of Second WWW Conference: Mosaic and the Web*, Chicago Octobre 1994.
- [Patterson 91] J. Patterson, “Comparing the programming demands of single– and multi–user applications”, *in Proceedings of UIST’91*, ACM Press, pp. 87–95, Hilton Head, SC Novembre 1991.
- [Rodden 92] T. Rodden, J. Mariani, G. Blair, “Supporting cooperative applications”, *In ComputerSupported Cooperative Work, Kluwer Academic Publishers*, 1(1), pp. 41–67, 1992.
- [Trevor 94] Trevor, J., Rodden, T. and Mariani, J., “The use of adapters to support cooperative sharing”, *in proceedings of CSCW’94*, ACM Press, pp. 219–230, Chappel Hill, Octobre 1994.
- [Ubique] Ubique, [<http://www.ubique.com>].
- [Worlds] Worlds, [<http://www.acsl.cs.uiuc.edu/kaplan/worlds.html>].
- [Lotus Notes] Lotus, [<http://www.lotus.com/groupware>].
- [InterNotes] InterNotes Web Publisher, [<http://www.internotes.com>].
- [Amir 93] Y. Amir, L. E. Moser, P. M. Melliar–Smith, D. A. Agarwal, P. Ciarfella, “Fast Message Ordering and Membership Using a Logical Token–passing Ring”, *In Proceedings of the 13th IEEE International Conference on Distributed Computing Systems (IC–DCS)*, pp. 551–560, Mai 1993.
- [Attiya 91] H. Attiya, J. Welch, “Sequential consistency vs. linearizability ”, *Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures*, pp. 304–325, Juillet 1991.
- [Babaoglu 94a] O. Babaoglu, R. Davoli, Luigi–A. Giachini, M. Gray, “RELACS: A communication infrastructure for constructing reliable applications in large–scale distributed system”, *BROADCAST Project deliverable report, Department of Computer science, University of Newcastle*, 1994.

- [Babaoglu 94b] O. Babaoglu, A. Schiper, “On Group Communication in Large Scale Distributed Systems”, *In proceeding of the ACM SGOPS European Workshop*, Dagstuhl Septembre 1994.
- [Balter 91] R. Balter, J.–P. Banâtre, S. Krakowiak, *Construction des systèmes d’exploitation répartis*, INRIA – Collection Didactique, Avril 1991.
- [Benford 92] S. Benford, H. Smith, A. Shepherd, A. Bullock, H. Howidi, “Information Sharing Approach to CSCW: The Grace project”, *Computer communications*, 15(8), pp. 933–946, Octobre 1992.
- [Bernstein 87] P. A. Bernstein, V. Hadzilacos and N. Goodman, “Concurrency control and Recovery in Database Systems”, *Addison–Wesley*, 1987.
- [Birman 87] K. Birman and T. Joseph, “Exploiting Virtual Synchrony in Distributed Systems”, *In Proceedings of the 11th Symposium on Operating System Principles*, Novembre 1987.
- [Birman 94] K. P. Birman and R. Van Renesse, “Reliable Distributed Computing with the Isis Toolkit”, *IEEE Computer Society Press, Los Alamitos, CA*, 1994.
- [Birman 96] K. Birman and T. Joseph, “Reliable Communication in the Presence of Failures”, *ACM Transactions on Computer Systems*, 5(1), pp. 47–76, Février 1987.
- [Chandra 92] T. D. Chandra, S. Toueg, *Unreliable Failure Detectors for Asynchronous Systems*, (14853), Department of Computer Science, Upson hall, Cornell University, Ithaca, New York, 1992.
- [Chandra 95] T. D. Chandra, V. hadzilacos, S. Toueg, B. Charron–ost , *On impossibility of Group Membership*, (TR95–1548), Department of Compuetr Science, Cornell University, 1995.
- [Chang 84] U. Chang, N. F. Maxembuk, “Reliable Broadcast Protocols”, *ACM Transactions on Computer Systems*, 2(3), pp. 251–273, Aout 1984.
- [Cheriton 85] D. R. Cheriton and W. Zwaenepoel, “Distributed Process Groups in the V Kernel”, *ACM Transactions on Computer Sustems*, 3(2), pp. 77–107, Mai 1985.

- [Coan 90] B. A. Coan and G. Thomas, “Agreeing on a Leader in Real–Time”, *In Proceedings of of the 11th Real–Time System Symposium*, pp. 166–172, Décembre 1990.
- [Cosquer 94] F. J. N. Cosquer and P. Veríssimo, “Survey of Selected Groupware and Supporting Platforms”, *Workshop on Basic Research On Advanced Distributed Computing: from Algorithms to Systems*, ESPRIT Basic Research Project 6360(1), pp. 1–31, Octobre 1994.
- [Ellis 89] C. A. Ellis, S. J. Gibbs, “Concurrency Control in Groupware Systems”, *ACM SIGMOD*, pp. 399–405, ACM, Portland, Oregon, Juin 1989.
- [Ezhilchelvan 95] P. D. Ezhilchelvan, R. A. Macêdo, S. K. Shrivastava, “Newtop: A Fault–tolerant Group Communication Protocol”, *In Proceedings of the 15th International Conference on Distributed Computing Systems*, Vancouver Canada, Juin 1995.
- [Fernandez 92] J. C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez, J. Sifakis, “A toolbox for verification of Lotos programs”, *In Lori A. Clarke, editor, Proceedings of the 14th International Conference on Software Engineering ICSE’14*, ACM, pp. 246–259, New–York, Mai 1992.
- [Fischer 85] M. J. Fischer, N. A. Lynch, M. Paterson, “Impossibility of Distributed Consensus with One Faulty Process”, *In Journal of the ACM*, 32(2), pp. 374–382, Avril 1985.
- [Jahanian 93] F. Jahanian, S. Fakhouri, R. Rajkumar, “Processor Group Membership Protocols: Specification, Design and Implementation”, *In Proceedings of the 12th IEEE Symposium on Reliable Distributed Systems*, pp. 2–11, Princeton, October 1993.
- [Kaashoek 91] M. F. Kaashoek, A. S. Tanenbaum, “Group Communication in the AMOEBA Distributed System”, *Proceedings of IEEE: 11th International Conference on Distributed Computing Systems*, pp. 222–230, Mai 1991.
- [Kerbrat 95] A. Kerbrat, S. Ben Atallah, “Formal Specification of a Framework for Synchronous Groupware Development”, *In Proceedings of Forte’95*, pp. 303–310, Montreal, October 1995.

- [Lamport 78] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System”, *Communications of the ACM*, 13(2), pp. 125–133, Juillet 1978.
- [Lamport 79] L. Lamport, “How to Make a Multiprocessor that Correctly Executes Multiprocess Programs”, *IEEE Transactions on Computers*, C-28(9), pp. 690–691, September 1979.
- [Malki 95] D. Malki, K. Birman, A. Riccardi, A. Schiper, *Uniform Actions in Asynchronous Distributed Systems*, (TR 94-1447), Department of Computer Science, Cornell University, 1995.
- [Manna 89] Z. Manna, A. Pnueli, “The Anchored Version of the Temporal Framework”, *Lecture Notes in Computer Science*, Vol 354, 1989.
- [Mishra 91] S. Mishra, L. L. Paterson, R. D. Schlichting, “A Membership Protocol Based on Partial Order”, *In Proceedings of the IEEE International Working Conference on Dependable Computing For Critical Applications*, pp. 137–145, Février 1991.
- [Mistra 89] J. Mistra, “Axioms for Memory Access in Asynchronous Hardware Systems”, *ACM Transactions on Programming Languages and Systems*, 8(1), pp. 142–153, Janvier 1989.
- [Melliar-Smith 94] P. M. Melliar-Smith, L. Moser, V. Agrawala, “Processor Membership in Asynchronous Distributed Systems”, *IEEE Transactions on Parallel and Distributed Systems*, 5(5), pp. 459–473, May 1994.
- [Moser 96] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, C. A. Lingley-Papadopoulos, “Totem: A Fault-Tolerant Multicast Group Communication System”, *Communication of the ACM*, 39(4), pp. 54–63, Avril 1996.
- [Powell 96] D. Powell, “Group Communication”, *Communications of the ACM*, 39(4), pp. 50–53, Avril 1996.
- [van Renesse 96] R. van Renesse, K. P. Birman and S. Maffeis, “Horus: A Flexible Group Communication System”, *Communications of the ACM*, 39(4), pp. 76–83, Avril 1996.

- [Ricciardi 91] A. Ricciardi, K. P. Birman, “Unsing Process Group to implement Failure Detection in Synchronous Environment ”, *In Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pp. 141–152, 1991.
- [Verissimo 92] P. Verissimo, W. Vogels, L. Rodrigues, “Group Orientation: A Paradigm for Modern Distributed Systems”, *In Proceedings of the ACM SIGOPS 1992 Workshop*, France, 1992.
- [Weihl 89] W. E. Weihl, “Local Atomicity Proprieties: Modular Concurrency Control for Abstract Data Types”, *ACM Transactions on Programming Languages and Systems*, 11(2), pp. 249–282, Avril 1989.
- [Comer 92] D. Comer, *TCP/IP, Architecture, protocoles, applications*, InterEditions, 1992.
- [van Rossum 96] G. van Rossum, *Grail : The Browser for The Rest of Us (DRAFT)*, Corporation for National Research Initiatives (CNRI), Reston, Virginia, (URL: <http://monty.cnri.reston.va.us/grail>), Mai 1996.
- [Salber 95] D. Salber, J. Coutaz, D. Decouchant, M. Riveill, “De l’observabilité et de l’honnêteté : le cas du contrôle d’accès dans le cas de la communciation Homme–Homme médiatisées”, *IHM’95*, Toulouse, Octobre 1995.
- [Quint 94] V. Quint, I. Vatton, “Making Structured Documents Active”, *Electronic Publishing*, 7(2), pp. 53–74, June 1994.
- [Welch 95] B. B. Welch, *Practical Programming in Tcl and Tk*, Prentice–Hall, 1995.
- [Turletti 94] T. Turletti, *The Inria Video Conferencing System (IVS)*, *ConneXions – The Interoperability Report*, 8(10),pp. 20–24, INRIA, Sophia Antipolis, Octobre 1994.