Conception et réalisation logicielles pour les collecticiels centrées sur l'activité de groupe : le modèle et la plate-forme Clover

Yann Laurillau

THÈSE
présentée par

Yann Laurillau

pour obtenir le titre de
DOCTEUR de L'UNIVERSITE JOSEPH-FOURIER-GRENOBLE I
(*arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992*)
Spécialité : **Informatique**

Conception et réalisation logicielles pour les collecticiels centrées sur l'activité de groupe : *le modèle et la plate-forme Clover*

Date de soutenance : 10 Septembre 2002

Composition du Jury :

Président :	Mr. Jean Caelen
Directeur de thèse :	Mlle. Laurence Nigay
Rapporteurs :	Mr. Michel Beaudouin-Lafon Mr. Michel Riveill
Examinateurs :	Mme. Joëlle Coutaz Mr. Nicholas Graham Mr. Daniel Salber

Thèse préparée au sein du laboratoire de
Communication Langagière et Interaction Personne-Système
Fédération IMAG
Université Joseph Fourier - Grenoble I

à mes parents,
à ma femme,
à mes amis ...

Remerciements

Ecrire une thèse est un travail long et souvent difficile, aussi pour ceux qui m'ont aidé et accompagné ces dernières années. Finalement, ce mémoire de thèse est une œuvre collective puisque de nombreuses personnes y ont participé, de près ou de loin, avec leur soutien scientifique, moral ou affectif.

Je remercie Laurence Nigay qui m'a encadré tout au long de ces trois années et qui m'a apporté une grande richesse et rigueur scientifique. Merci pour ta confiance, ta disponibilité et ton attention permanente sur l'évolution de mes travaux. Merci aussi pour ta complicité et ta bonne humeur qui ont joué un rôle important, surtout pendant les périodes de crises thésardiques. Enfin, je te remercie car c'est grâce à toi, par le biais de ce fameux projet de fin d'école d'ingénieur, que j'ai véritablement découvert le goût de la recheche scientifique.

Je remercie Joëlle Coutaz, responsable de l'équipe IIHM, qui anime cette équipe avec passion scientifique et avec beaucoup d'affection pour ceux qui y travaillent. Dans cette ambiance, la recherche devient un plaisir. Je te remercie pour m'avoir accordé du temps, chose précieuse lorsque l'on regarde ton agenda, pour me guider dans ma démarche scientifique. Merci aussi pour avoir accepté d'être lecteur de cette thèse et membre de mon jury.

Je remercie les membres du jury qui m'ont fait l'honneur d'évaluer et de juger la qualité de mon travail : Michel Beaudouin-Lafon, Jean Caelen, Nick Graham, Michel Riveill et Daniel Salber. Une pensée particulière pour Nick qui n'a pas pu se déplacer et qui était présent "virtuellement" grâce à la magie du téléphone (sic).

Je remercie tous les membres de l'équipe IIHM pour la bonne humeur qui contribue à cette ambiance chaleureuse : les nombreuses pauses café, les fêtes rue Ampère, chez Joëlle ou au resto, les journées base-ball (quand il ne pleut pas), le rituel barbecue sur les terrains de l'île d'amour, les soirées cinés sur grand écran (toujours rue Ampère), etc. Merci à Nick et à Leon lors de leur passage dans l'équipe.

Je remercie tous mes amis qu'ils soient à Grenoble, à Paris, au Mans, à Glasgow ou ailleurs.

Merci à Pascal, Olivier et Magali et Julie, Fanf, Fred, Camille et Sébastien, Dave, Sylvain B., Mathias et Blandine, Manu, Ninie, Christèle et Bertrand, Gaëlle, Nils, Kris, Fredouille, Eric et Céline, Phiphi, Chaouki, à tous les nouveaux et les anciens de l'équipe IIHM, et tout ceux que je n'ai pas cité et qui se reconnaitront. Un merci un peu particulier au CIES de Grenoble et à Claude Gaubert.

Merci à mes parents, Michel et Marie-Claude, à mes beaux-parents, Jean-Pierre et Michèle.

Et un grand merci à Anne-Lise ...

Table des matières

Table des figures

Chapitre I *Introduction*

Nos travaux s'inscrivent dans le domaine du Travail Coopératif Assisté par Ordinateur (TCAO) et sont consacrés à la conception et à la réalisation logicielles des collecticiels, c'est-à-dire des systèmes interactifs multi-utilisateurs. Le domaine du TCAO constitue un sous-domaine de celui de l'Interaction Homme-Machine (IHM), ce dernier étant dédié aux systèmes interactifs en général. le domaine du TCAO, souvent considéré comme la "petite sœur" de l'IHM, a émergé en 1984 au cours d'un groupe de travail organisé par Irène Greif et Paul Cashman, auteurs notamment du terme *CSCW* (*Computer Supported Cooperative Work*). La première conférence internationale dédiée à ce domaine a vu le jour en 1986 tandis que la première conférence Européenne s'est déroulée en 1989.

Le domaine du TCAO, tout comme le domaine de l'IHM, se situe à la croisée de plusieurs disciplines : l'informatique (comme le génie logiciel ou les technologies de la communication) et les sciences humaines (comme la psychologie cognitive, la sociologie ou l'ethnographie). Ainsi, la démarche générale de l'ingénierie des systèmes interactifs consiste à définir et de concevoir des outils à la fois pour comprendre et analyser l'interaction à l'aide de théories, méthodes et modèles issus des sciences humaines, pour ensuite envisager des solutions logicielles adaptées. La mise en commun de savoirs pluridisciplinaires est donc une exigence du domaine de l'IHM et par conséquent du domaine du TCAO.

1. Sujet

L'émergence des collecticiels est depuis les années 80 lente et progressive, mais connaît depuis peu un essor important avec l'avènement des réseaux informatiques et de la téléphonie mobile. Selon une étude récente [IDC 2002], les jeux en réseau à travers internet deviennent un des marchés les plus importants pour l'industrie des jeux. En effet, on estime en 2000 à 25 millions le nombre de joueurs en réseau dans le monde et ce nombre devrait dépasser les 40 millions d'ici 2004. Le *World Wide Web* n'est pas en reste puisqu'il participe à cette démocratisation progressive de l'accès aux réseaux informatiques.

Ainsi les collecticiels ne sont plus confinés aux laboratoires de recherche et strictement réservés aux informaticiens. Ils sont de plus en plus acceptés et accaparés par le grand public : les jeux en réseau, les forums de discussion (*chat* et *newsgroup*), les échanges de courriers électroniques (*email*), les échanges de fichiers (musicaux ou autres), les communautés virtuelles mais réelles comme les communautés de développement logiciel Open Source, la vidéoconférence sont des exemples de collecticiels quotidiennement utilisés aujourd'hui.

Néanmoins, cette multiplicité des collecticiels ne s'est pas accompagnée d'outils de mise en œuvre des collecticiels, que ce soient des outils de conception ou de réalisation logicielle. Or la complexité de mise en œuvre d'un collecticiel est accrue si nous la comparons à celle d'un système interactif mono-utilisateur (IHM).

D'une part, cette complexité est due à la variété des aspects à prendre en compte lors de la conception. Ainsi des éléments de conception sont d'ordre social puisqu'un collecticiel met en œuvre la collaboration au sein d'un groupe d'individus. D'autres éléments sont d'ordre éthique : par exemple la protection de la vie privée peut être un élément important de conception d'un collecticiel. Ces éléments, qui n'interviennent pas dans la conception d'un système mono-utilisateur, participent à la complexité de conception d'un collecticiel.

D'autre part, lors de la réalisation logicielle, des problèmes propres aux collecticiels impliquent des solutions logicielles adaptées. Par exemple, le logiciel doit réguler les actions concurrentes de plusieurs utilisateurs ou encore faire face aux aléas de l'infrastructure communicante (les réseaux informatiques). Il convient de constater que la réalisation logicielle d'un collecticiel reste aujourd'hui une tâche complexe et non systématique. Tandis qu'un développement ad hoc d'un système est acceptable pour un prototype jetable, il est admis que la conception logicielle d'un système complexe comme un collecticiel ne peut reposer que sur le savoir artisanal du développeur. Parmi les outils existants pour la réalisation logicielle, nous distinguons deux classes :

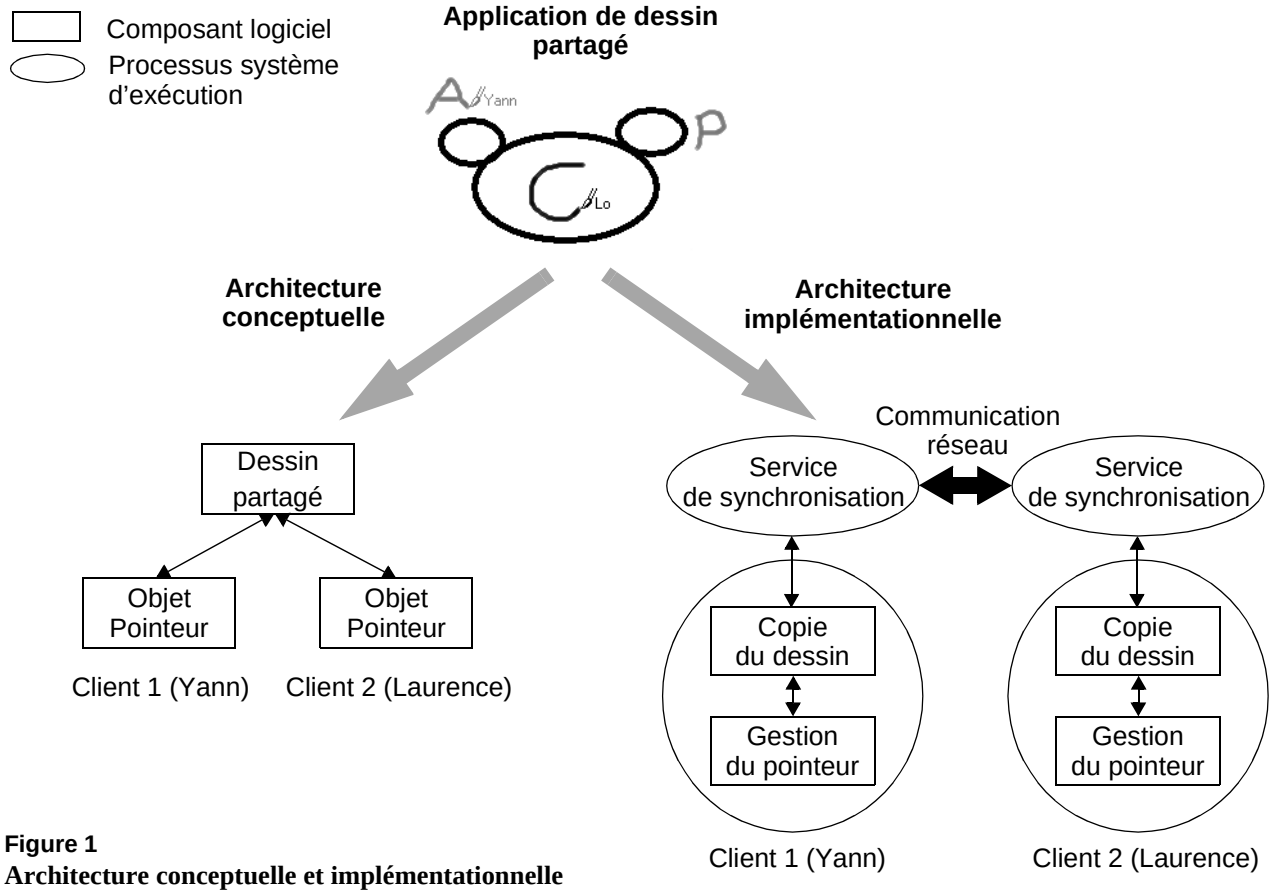


Figure 1
Architecture conceptuelle et implémentationnelle

1. les outils pour la conception logicielle que sont les modèles d'architecture logicielle,
2. les outils pour le développement logiciel que sont les boîtes à outils et les plates-formes de développement.

Outils de conception logicielle

Les modèles d'architecture définissent des structures modulaires normalisées permettant d'organiser les éléments logiciels du collecticiel. En appliquant un modèle d'architecture, le concepteur élabore une architecture conceptuelle du système, puis la traduit ensuite en une architecture implémentationnelle dépendante des outils de développement utilisés. Une architecture conceptuelle définit la structure modulaire : l'ensemble des modules et leurs relations ainsi que la couverture fonctionnelle de chaque module. A partir de cette structure modulaire, il convient alors d'assigner aux modules des processus (structure de coordination) puis d'allouer les processus aux processeurs (structure physique). La structure de coordination et la structure physique complètent la structure modulaire de l'architecture conceptuelle pour constituer ensemble l'architecture implémentationnelle. Nous illustrons la différence entre une architecture conceptuelle et une architecture implémentationnelle en considérant un exemple simple de dessin partagé. L'application de dessin partagé met en œuvre plusieurs pointeurs, un par utilisateur, permettant à plusieurs utilisateurs de dessiner ensemble dans

une zone partagée. L'architecture conceptuelle de cette application, comme le montre la Figure 1, est constituée d'un composant logiciel partagé correspondant à la zone de dessin commune. Ce composant communique avec plusieurs instances d'un composant gérant un pointeur propre à chaque utilisateur. L'architecture implémentationnelle est différente puisqu'elle décrit, entre autres, l'allocation des composants aux processus. Ainsi, une architecture implémentationnelle de l'éditeur de dessin partagé, présentée à la Figure 1, comprend un processus par utilisateur, ce dernier gérant une copie du dessin et un pointeur. Les copies sont gérées par un service de synchronisation, chaque instance de ce service communiquant à travers le réseau.

Les outils de développement, exposés au paragraphe suivant, n'éliminent pas la nécessité d'une architecture logicielle conceptuelle. L'architecture logicielle est un artefact incontournable tant que les outils impliquent une phase de programmation. Sans une architecture logicielle, le collecticiel est plus difficile à développer, à modifier, à étendre et à maintenir.

Outils de développement

Les outils de développement comme les boîtes à outils et les infrastructures reposent sur des bibliothèques de solutions logicielles satisfaisant une classe de problèmes. Le but visé est de réduire la charge de programmation par réutilisation de logiciel mais aussi de permettre de raisonner et de programmer à un haut niveau d'abstraction. Par exemple, les boîtes à outils classiques en IHM proposent des palettes d'objets graphiques prêts à l'emploi comme une barre de défilement. De même, pour le développement de collecticiels, certaines boîtes à outils proposent une barre de défilement collaborative permettant à plusieurs utilisateurs de la manipuler.

2. *Objectifs*

Face à la complexité de mise en œuvre d'un collecticiel, les objectifs de notre étude sont de définir et de concevoir des outils de conception et de réalisation logicielles pour collecticiels. Nous adoptons un regard nouveau sur les outils existants cités ci-dessus, en étudiant comment de tels outils dédiés au développement du logiciel peuvent intégrer des aspects de la conception comme ceux relevant de l'ergonomie ou de l'activité de groupe. Ainsi nous visons des outils qui puissent aider ou accompagner la transition entre la conception du collecticiel guidée par des aspects sociaux, éthiques et ergonomiques et la conception logicielle soumise à des contraintes fortes exercées par les exigences techniques. Pour résumer, notre étude vise à répondre à la question suivante :

> Dans quelle mesure les outils disponibles pour la conception et la réalisation logicielles de collecticiels, en l'occurrence les modèles d'architecture, les boîtes à outils et les plates-formes, aident-ils l'architecte et l'informaticien à proposer des solutions logicielles conformes aux spécifications, satisfaisant ainsi les requis sociaux, éthiques et ergonomiques ?

Nos objectifs se situent donc à la croisée des pratiques des sciences humaines et du génie logiciel. L'établissement de liens entre les aspects de l'IHM (ergonomie, sociologie, ethnographie, etc) et l'architecture logicielle constituent un point d'ancrage des sciences humaines dans le processus de développement logiciel. Nous soulignons l'importance de ce point d'ancrage en le situant dans le processus de mise en œuvre d'un collecticiel : un modèle d'architecture logicielle constitue un outil dédié à la phase de spécification globale du logiciel, phase de transition dans le processus de mise en œuvre, de l'espace Interface Homme-Machine (IHM) vers l'espace logiciel. En effet, il est classique de répartir les étapes du processus de mise en œuvre d'un système interactif en deux espaces : l'espace IHM et l'espace logiciel. Le premier se caractérise par la priorité qu'il faudrait accorder aux aspects ergonomiques, sociaux et éthiques, le second par l'accent mis sur les techniques d'implémentation logicielle. L'espace IHM inclut en particulier l'analyse des besoins et la conception de l'interface consignée dans le document de Spécifications Externes. A cette étape de conception, interviennent l'ergonomie, la sociologie ou encore l'ethnographie pour la définition et la spécification de l'interface utilisateur. L'espace logiciel laisse la place aux compétences informatiques avec les conceptions globales et détaillées, le codage et les tests unitaires et d'intégration. Aussi, avec la conception globale du logiciel reposant sur un modèle d'architecture conceptuelle, nous quittons l'espace de conception proprement dit de l'Interface Homme-Machine pour pénétrer dans l'espace logiciel réservé aux informaticiens. Cette situation particulière dans le processus de mise en œuvre se traduit par

une tension permanente entre d'une part la satisfaction des requis de l'utilisateur, des spécifications externes de l'IHM et d'autre part les contraintes techniques imposées par les outils de mise en œuvre, la satisfaction des requis de qualité logicielle.

Vis-à-vis de nos objectifs, nous identifions deux étapes de travail :

- **Analyser les outils existants de conception** (espace IHM) **et de développement logiciel** (espace Logiciel) pour les collecticiels. Cette analyse a pour objectif d'identifier des éléments de conception de l'interaction (espace IHM) et de réalisation logicielle (espace logiciel) en vue d'en étudier leurs liens : *le centre d'intérêt est ici la passerelle entre les espaces IHM et Logiciel.*

- **Proposer et réaliser des outils de conception et de réalisation logicielles** (espace Logiciel) pour les collecticiels qui offrent une approche complémentaire aux outils existants en *intégrant des éléments de conception de l'interaction* (espace IHM)

La démarche de travail, selon les deux étapes ci-dessus, repose sur **l'établissement d'une grille d'analyse** traduisant des éléments de conception de l'interaction (espace IHM). Il convient que la grille d'analyse organise dans un espace cohérent les éléments liés à l'activité de groupe, c'est-à-dire les actions individuelles ou collectives dans un contexte d'exécution. Cette grille définit alors un cadre d'étude et de comparaison des outils de réalisation logicielle existants et guide la définition et la conception de nouveaux outils de réalisation logicielle.

3. Structure du mémoire

La structure en chapitres de ce mémoire reflète notre démarche de recherche : le Chapitre II cerne précisément le contexte de l'étude en mettant l'accent sur l'importance des modèles d'architecture logicielle et des outils de développement pour les collecticiels. De plus, ce chapitre présente notre grille d'analyse des outils. Cette grille est utilisée dans les chapitres suivants, ces derniers étant organisés selon les deux classes d'outils existants : les modèles d'architecture pour les collecticiels (Chapitre II et Chapitre IV) et les outils de développement pour les collecticiels (Chapitre V et Chapitre VI). Pour chaque classe d'outils, nous dressons un état de l'art (Chapitre III et Chapitre V) organisé selon la grille d'analyse du Chapitre II, puis nous présentons notre contribution (Chapitre IV et Chapitre VI) que nous motivons en nous appuyant sur notre grille. Le Chapitre VII présente alors deux exemples de collecticiels développés avec nos outils.

Dans le Chapitre II, nous situons le rôle et la portée des outils existants de conception (espace IHM) et de développement logiciel (espace Logiciel) des collecticiels en les situant dans un cadre fédérateur. Pour cela, le chapitre est structuré selon un cycle de vie du logiciel. L'objectif est, outre la définition précise du rôle des outils existants, la mise en évidence des tensions entre les phases de conception de l'interface et de mise en œuvre logicielle. A l'issue de ce chapitre, nous détaillons notre démarche de recherche en définissant une grille d'analyse utilisée ensuite tout au long de cette étude, notamment pour analyser différents modèles d'architecture dans le Chapitre III.

Les Chapitre III et Chapitre IV sont consacrés aux modèles d'architecture pour collecticiels. Dans le Chapitre III, nous définissons le concept d'architecture logicielle en distinguant l'architecture conceptuelle de l'architecture d'implémentation. Puis nous étudions et analysons les modèles d'architecture existants selon les dimensions identifiées par notre grille d'analyse. A partir des observations et des conclusions tirées de cette étude, nous présentons dans le Chapitre IV notre modèle d'architecture pour la conception logicielle de collecticiels, le modèle d'architecture Clover.

Les Chapitre V et Chapitre VI sont dédiés aux plates-formes et infrastructures pour le développement des collecticiels. Nous appliquons la même approche et donc la même structuration que lors de l'étude des modèles d'architecture. Aussi, dans le Chapitre V, nous étudions plusieurs plates-formes logicielles pour le développement des collecticiels selon notre grille d'analyse. A partir des conclusions issues de cette étude, nous présentons dans le Chapitre VI notre plate-forme de développement pour les collecticiels, la plate-forme Clover.

Le Chapitre VII complète la présentation de notre modèle d'architecture Clover (Chapitre IV) et de notre plate-forme Clover (Chapitre IV) en illustrant leurs utilisations dans la mise en œuvre de deux collecticiels. Les deux systèmes développés sont le système CoVitesse, un système de navigation collaborative sur le *WWW*, et un système de tableau blanc partagé.

Au Chapitre VIII, nous concluons cette étude par une synthèse de notre contribution selon les deux facettes suivantes : conception logicielle et réalisation logicielle. Nous élargissons ensuite le cadre d'étude, par l'esquisse de perspectives tant conceptuelles que techniques.

Une annexe est incluse à la fin du mémoire. Celle-ci présente la conception de notre modèle de navigation collaborative par l'application du modèle Denver. Ce modèle de navigation comprend quatre types de navigation collaborative explicites dans notre système CoVitesse.

Chapitre II *Domaine du Travail Coopératif Assisté par Ordinateur*

1. Introduction

Le domaine du Travail Coopératif Assisté par Ordinateur (TCAO) a pour thème d'étude les **collecticiel** (ou **synergiciel**). De nombreuses définitions ont été proposées pour caractériser un collecticiel dont nous citons la plus courante, celle de C. Ellis :

> "*Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment*." [Ellis 1991]
>
> "*Les collecticiels sont des systèmes informatiques qui assistent un groupe de personnes engagées dans une tâche commune (ou but commun) et qui fournissent une interface à un environnement partagé.*" (traduction de A. Karsenty [Karsenty 1994])

Le domaine de la TCAO, lié au domaine de l'IHM, doit adapter et étendre les méthodes et outils développés en IHM car ceux-ci ont été forgés par rapport à une activité mono-utilisateur et en fonction d'un seul utilisateur. Ce constat est particulièrement saillant en ce qui concerne les facteurs humains (sciences humaines). Tandis que le domaine de l'IHM s'est longtemps appuyé sur des théories de psychologie cognitive et comportementale, le domaine du TCAO, dès ses débuts, s'est tourné vers les sciences sociales et en particulier l'ethnographie pour analyser les aspects sociaux au sein d'un groupe en situation de travail. Nous constatons encore aujourd'hui un gouffre entre les études liées aux facteurs humains en IHM et en TCAO. Dans [Bannon 1998], L. Bannon souligne cette différence et prône pour une plus grande socialité des études en IHM en prônant une "ergonomie sociale".

Dans cette étude, nous nous intéressons plus particulièrement aux outils conceptuels et aux outils de développement pour la mise en œuvre logicielle des collecticiels. Dans le domaine du TCAO, ces outils ont une position essentielle du fait de la complexité de mise en œuvre des collecticiels. Comme nous l'avons évoqué dans l'introduction générale, cette complexité est accrue comparée au développement d'un système interactif mono-utilisateur en IHM classique. D'une part, le collecticiel s'adresse à un groupe d'utilisateurs : celui-ci doit être capable de reproduire un espace social et un climat social favorable à l'activité de groupe. A cela viennent se greffer des problèmes d'ordre éthique comme le respect de la personne et de sa vie privée. D'autre part, le développeur d'un collecticiel doit faire face à des difficultés techniques supplémentaires liées à la gestion des accès concurrents et aux problèmes induits par la communication à travers un réseau informatique.

L'objectif de ce chapitre est de montrer le rôle et l'apport des outils de conception et de développement logiciel des collecticiels pour répondre à cette complexité et montrer dans quelle mesure ces outils aident le concepteur puis le développeur dans la réalisation d'un collecticiel. Ainsi, pour positionner ces outils dans un contexte de réalisation d'un collecticiel, ce chapitre est organisé de la façon suivante : la première partie présente trois taxonomies pour caractériser les collecticiels selon trois points de vue et ainsi mesurer la complexité de mise en œuvre d'un collecticiel. Dans la seconde partie du chapitre, nous présentons un ensemble de méthodes et outils dédiés à la conception et à la réalisation de collecticiels. A des fins analytiques et pour souligner leurs portées, nous les situons au sein du cycle de vie du logiciel. Enfin, la dernière partie rappelle, à la lumière de ce chapitre, les objectifs de cette étude et détaille la démarche de travail retenue pour traiter notre sujet.

2. Taxonomies des collecticiels

Dans cette partie, nous présentons trois taxonomies aux caractéristiques complémentaires afin de définir plus finement ce qu'est un collecticiel et de cerner l'étendue des possibilités. La multiplicité des possibilités peut se voir comme un indicateur du dynamisme de ce domaine, avec, sa compagne immédiate, la complexité de réalisation logicielle.

Les deux premières taxonomies présentées, l'une selon les types d'application et l'autre Espace-Temps, sont incontournables dans la littérature du domaine. La dernière taxonomie choisie repose quant à elle sur un modèle du travail coopératif. Les trois taxonomies exposées, *notre première contribution sera de choisir et de proposer un ensemble de termes issus de ces taxonomies qui couvrent raisonnablement le domaine de réflexion*.

2.1. TYPES D'APPLICATION

La première taxonomie consiste en une classification par domaines d'application. Aussi cette taxonomie doit souvent être révisée pour prendre en compte les nouveaux types d'application. Elle traduit donc une image du domaine à un instant donné. Pour élaborer une liste des domaines d'application des collecticiels, nous avons combiné plusieurs taxonomies existantes [Ellis 1991] [Dix 1993] [Karsenty 1994] [UsabilityFirst 2002]. La variété des domaines d'application souligne le dynamisme de cet axe de recherche. Nous obtenons quatre catégories de collecticiels qui sont :

1. les applications dédiées à la communication homme-homme médiatisée (CHHM, ou *CMC* pour *Computer-Mediated Communication*) où nous regroupons les messageries électroniques, les forums de discussion, les systèmes de vidéoconférence et les *mediaspace*,
2. les applications d'édition où nous classons les éditeurs de texte et les tableaux blancs partagés,
3. les applications pour la coordination où nous rassemblons les systèmes *workflow*, les systèmes d'aide à la décision et les calendriers partagés,
4. les applications de jeux en réseau.

Nous présentons dans la suite les différents types de collecticiels dans l'ordre des quatre catégories ci-dessus.

- **Les messageries électroniques** (*email*) sont actuellement les collecticiels les plus répandus et les plus utilisés. Le trafic généré par ces systèmes est devenu tel que, notamment avec l'intrusion du courrier électronique publicitaire (*junk mail* ou *spam*), les outils de

messagerie se sont enrichis de fonctionnalités "intelligentes" pour trier les courriers, pour détruire les courriers non désirables ou pour envoyer des réponses automatiquement. Il existe de nombreuses applications de gestion de courriers électroniques tels que l'utilitaire `mail` sous Unix ou l'application Eudora [Qualcomm 2002]. La grande tendance dans ce domaine est celle de la messagerie instantanée (*Instant Messaging*) avec, par exemple, les services de messagerie SMS (*Short Messaging Services*) dans le monde de la téléphonie mobile.

- **Les forums de discussion** (*chat* et *newsgroup*) : les deux principales classes de forum de discussion diffèrent par leur mode d'utilisation, synchrone ou asynchrone. La première classe regroupe les forums en ligne du type IRC (*Internet Relay Chat*), des applications très répandues qui reposent sur le concept de canal à thème (canal de discussion). La discussion est ici synchrone à l'opposé de la seconde classe qui rassemble les listes de diffusion (*mailing list*) et les *newsgroups* (système *USENET*) pour des discussions asynchrones sur un thème donné. Contrairement au courrier électronique, ce type de collecticiel est destiné à communiquer avec un grand nombre de personnes.

- **Les systèmes de vidéoconférence** permettent à des personnes physiquement distantes de se réunir et communiquer par l'intermédiaire d'un support audio et vidéo. Il s'agit d'un forum de discussion offrant une communication reposant sur des données audio et vidéo à l'opposé des forums de discussion du point précédent qui se basent sur des échanges textuels. La grande difficulté du déploiement de ce type d'application est liée en grande partie à la nécessité de disposer d'une bande passante capable de diffuser et recevoir des données audio et vidéo avec une qualité acceptable. Le système PictureTel est un des systèmes le plus connu [PictureTel 2002].

- **Les *mediaspace*** [Coutaz 1999] [Dourish 1992] [Finn 1997] [Mackay 1999] sont des collecticiels mettant en œuvre une liaison vidéo au sein d'une équipe dans le but de favoriser la communication informelle et d'entretenir une conscience de groupe forte entre membres distants : entre deux étages, deux bâtiments, deux villes, etc. L'objectif visé est différent des systèmes de vidéoconférence bien que les deux types d'application reposent sur des flux vidéo. En effet, contrairement à la vidéoconférence qui met en relation des individus sur une courte période et de manière planifiée, la connexion vidéo d'un *mediaspace* est permanente et l'interaction est opportuniste. Aussi un *mediaspace* soulève le problème de la protection de la vie privée de par

la présence de caméras fonctionnant en permanence dans les bureaux ou lieux communs d'un organisme (cafétéria, salle de réunion, etc.).

- **L'édition conjointe** (*shared editing*) [Prakash 1999] : les éditeurs partagés sont des systèmes dédiés à l'édition collaborative de documents avec gestion des différentes versions. Ces outils sont complexes à réaliser, en particulier pour la gestion des tâches concurrentes comme le "défaire" et "refaire" (*undo* et *redo*) [Choudary 1995] ou la fusion de différentes versions [Dourish 1996b]. Les éditeurs de texte partagés comme le système StorySpace [Eastgate 2002] ou les éditeurs de dessins partagés comme les tableaux blancs partagés [Ishii 1994] (*shared whiteboard*) sont des exemples de collecticiels permettant l'édition conjointe.

- **Les systèmes *workflow*** [Flores 1988] [Bowers 1995] [Ellis 1999] sont des systèmes dédiés à la gestion de processus (industriels, commerciaux, administratifs, etc) et à la coordination des différents intervenants au cours d'un processus. Un processus s'articule sur la réalisation de documents industriels et le système *workflow* a la charge de veiller à la bonne circulation des documents entre les différents intervenants aux moments clés du processus.

- **Les systèmes d'aide à la décision** (*GDSS, Group Decision Support Systems*) sont conçus pour faciliter la prise de décisions grâce à l'apport de nombreux outils : brainstorming, votes, pondération des décisions, génération et annotation des idées, etc. Ces systèmes encouragent tous les participants à s'engager dans la prise de décision, par exemple en permettant de conserver l'anonymat ou en garantissant que chaque participant puisse prendre au moins une fois la parole [UsabilityFirst 2002].

- **Les calendriers partagés** (*group calendars*) [Palen 2002] sont des systèmes qui offrent des services de planification de tâches, de gestion de projets et de coordination de membres d'une équipe de travail. Contrairement aux systèmes *workflow*, la planification n'est pas centrée sur l'acheminement d'un document ou d'un quelconque support de travail. Les fonctionnalités usuelles incluent la détection d'incompatibilités dans la planification d'une tâche ou la détermination de plages horaires communes aux membres d'un groupe. Par exemple,

Figure 1
Starcraft [Blizzard 2002]
Starcraft est un exemple de jeu de stratégie fondé sur la compétition. Les joueurs doivent élaborer une tactique de développement de troupes pour détruire l'adversaire et pour se protéger de l'invasion ennemie.

Lotus/Organizer [Lotus 2002] est un outil de planification collaborative.

- **Les jeux en réseau** sont certainement les collecticiels qui connaissent, avec les systèmes de messagerie, l'essor le plus fulgurant. Parmi les jeux les plus connus, citons le jeu Quake [Id Software 2002] ou Starcraft [Blizzard 2002] présenté à la Figure 1. Ces jeux misent sur la coopération et la compétition entre les joueurs. Ce type d'application est comparable à une forme d'éditeur partagé reposant, dans la majorité des cas, sur un mode de communication textuelle.

En synthèse, nous constatons une grande variété des domaines d'application des collecticiels. Comme expliqué en introduction de ce paragraphe, nous proposons de regrouper les neuf domaines d'application ci-dessus présentés, en quatre catégories : communication (messagerie, forum, vidéoconférence, *mediaspace*), production (éditeur partagé, tableau blanc partagé), coordination (*workflow*, calendrier partagé, prise de décision) et jeux. Certes cette catégorisation traduit l'ensemble des collecticiels existants à ce jour. A ne pas en douter, il conviendra de revisiter notre taxonomie dans quelques années pour intégrer de nouveaux types de collecticiels.

Relevons enfin le cas de collecticiels qui embrassent plusieurs domaines d'application de plusieurs catégories. L'objectif affiché de tels collecticiels est alors l'intégration de plusieurs outils complémentaires pour collaborer comme des éditeurs partagés, des tableaux blancs partagés et des systèmes d'aide à la décision au sein d'un même environnement. Ce dernier peut être réel ou virtuel.

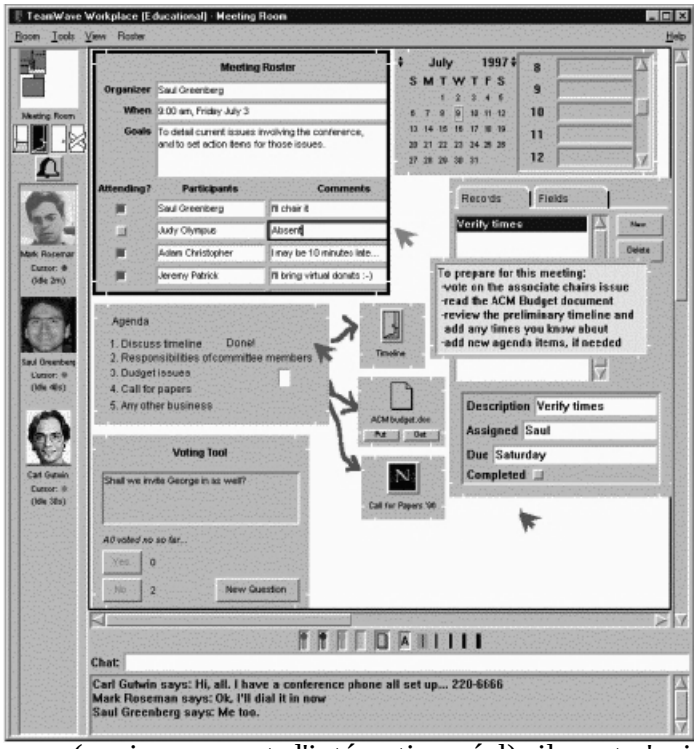


Figure 2
Bureau intégré TeamRooms [Roseman 1996b]
Il s'agit d'un bureau complet mettant à disposition un ensemble d'outils tels que les annotations, le système de vote, un tableau blanc, un calendrier partagé, une fenêtre de discussion, etc. Ce système rend observable les activités de tous les participants par le biais de cartes d'identité (photos), de pointeurs ou de vue radar (en haut à gauche de la copie d'écran).

- Dans le premier cas (environnement d'intégration réel), il peut s'agir d'une pièce spécialement aménagée pour faciliter le face-à-face et l'interaction avec les documents électroniques comme, la *Facilitator Room* du MIT MediaLab [MIT Medialab 2002].

- Dans le second cas, où les outils de collaboration sont intégrés dans un environnement virtuel, le système s'appuie sur la métaphore de la pièce ou du bureau partagé, comme le système TeamRooms [Roseman 1996b] de la Figure 2. Grâce à un espace partagé entre les utilisateurs, le système rend perceptible la présence de tous les utilisateurs, les activités de chacun, l'historique des activités et les communications entre utilisateurs.

Par l'exposé de cette taxonomie, nous nous sommes intéressés aux domaines d'application d'un collecticiel. Nous considérons maintenant deux autres caractéristiques complémentaires pour caractériser un collecticiel : les aspects temporel et spatial de la collaboration.

2.2. CLASSIFICATION ESPACE-TEMPS

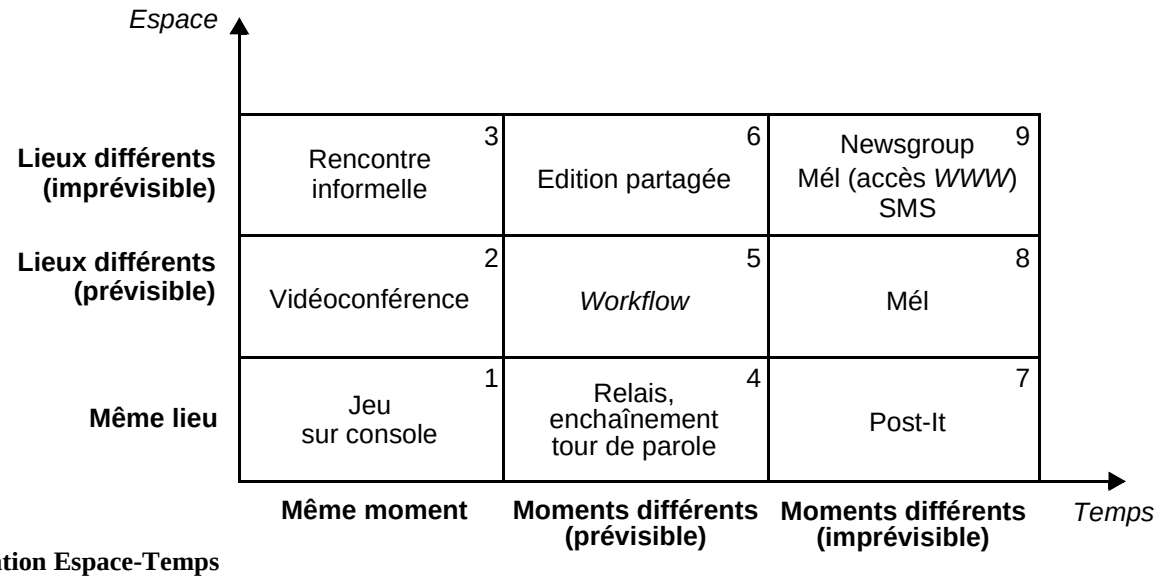


Figure 3
Classification Espace-Temps

La classification Espace-Temps repose sur deux caractéristiques, à savoir où et quand une action est exécutée par un des utilisateurs par rapport aux autres utilisateurs. Il s'agit de la classification la plus largement adoptée dans le domaine du TCAO, nommée Espace-Temps ou matrice Espace-Temps. La classification présentée à la Figure 3 est celle de J. Grudin [Grudin 1994], une version étendue de la matrice Espace-Temps de C. Ellis [Ellis 1991].

La classification Espace-Temps s'organise selon deux axes caractérisant l'usage du collecticiel : le premier axe *Espace* considère la distance spatiale entre les utilisateurs (*Même lieu* et *Lieux différents*) et le deuxième axe *Temps* considère la distance temporelle entre les utilisateurs (*Même moment* et *Moments différents*). Pour ce dernier axe, les termes couramment utilisés sont **synchrone** et **asynchrone**. Néanmoins, il n'est pas toujours possible d'attester du caractère prévisible du moment d'interaction, ni du lieu dans lequel se déroule l'interaction puisque l'interaction peut se dérouler au cours du temps dans des lieux différents. Ainsi, l'extension de la matrice Espace-Temps [Ellis 1991] proposée par J. Grudin [Grudin 1994] tient compte du caractère imprévisible de l'interaction pour les deux axes considérés. La matrice résultante, comme le montre la Figure 3, considère trois cas pour chaque axe. Nous illustrons les neuf types de collecticiels obtenus :

1 Les consoles de jeux ou les bornes d'arcade multi-joueurs sont des formes de collecticiel synchrone en face-à-face. Ces consoles sont constituées de deux manettes (ou plus) et chaque joueur pilote un personnage de jeu ou un objet animé (voiture, raquette, etc).

2 Les systèmes de vidéoconférence sont des collecticiels synchrones et l'interaction entre les utilisateurs a lieu en temps réel (même moment). Les participants sont répartis géographiquement dans des lieux différents mais prévisibles.

3 La rencontre informelle, comme celle au sein d'un forum de discussion, se réalise de manière imprévisible comme une rencontre fortuite dans la rue. Ce type d'interaction est de nature synchrone. Par exemple, les *mediaspace* [Coutaz 1999] [Dourish 1992] ont pour but de favoriser les rencontres informelles.

4 La prise de parole au cours d'une présentation est un cas où l'enchaînement des actions a été planifié. Il s'agit en l'occurrence de prendre le relais les uns à la suite des autres pour continuer la présentation. Contrairement aux systèmes *workflow* qui relèvent de la situation suivante, les actions réalisées se déroulent dans un même lieu.

5 Les systèmes *workflow* ont pour objectif la coordination des activités et des intervenants au cours d'un processus industriel. Ces systèmes permettent ainsi la planification des interventions de chacun sur les documents industriels échangés au cours d'un processus. Dans cette situation, les participants sont répartis géographiquement mais localisables (le bureau sur le lieu de travail).

6 Dans le cas de l'édition partagée asynchrone, il n'est pas nécessaire de connaître la localisation exacte des différents auteurs du document. Par contre, pour pouvoir transmettre le document entre les auteurs, ces derniers doivent planifier leurs échanges.

7 Le post-it, qu'il soit réel (papier) ou virtuel, est un support de la collaboration très largement adopté. La transmission du post-it est totalement imprévisible et se déroule, au sens large, dans un même lieu. Par extension, nous pouvons supposer que le support d'un post-it, comme un livre, est toujours le lieu de l'interaction bien qu'il soit possible de le déplacer.

8 Le courrier électronique dans sa forme usuelle est comparable au courrier postal : les destinataires sont répartis géographiquement, mais sont localisables (lieu de travail, lieu de résidence, etc). Ainsi, lorsque l'on envoie un message électronique, l'envoyeur peut localiser le destinataire. Par contre, le moment de réception est imprévisible.

9 Les listes de diffusion (ainsi que les nouvelles formes de courrier électronique avec accès par le *WWW*) ne permettent pas la localisation des participants. De plus, il n'est pas possible de prévoir à quel moment les messages seront lus.

En synthèse, cette classification considère deux caractéristiques, le temps et l'espace, pour définir 9 classes de collecticiels, que nous avons illustrées.

Il convient de noter que cette classification Espace-Temps ne permet pas de différencier les collecticiels où les utilisateurs sont mobiles. En effet, cette classification ne tient pas compte du passage d'un lieu à un autre de manière continue, ce qui est désormais possible avec l'informatique mobile.

Enfin il est intéressant de constater que les collecticiels existants classés par domaine d'application dans le paragraphe précédent considèrent presque tous que leurs utilisateurs sont distants (lieux différents, cas prévisible et imprévisible). Certes, il semble assez immédiat de conclure que c'est dans le cas d'un groupe d'utilisateurs distants que l'informatique peut le plus contribuer à favoriser la collaboration. Il convient néanmoins de noter aussi que la réalisation de collecticiels, où les utilisateurs sont distants, soulève des enjeux techniques que la communauté de recherche a souhaité traiter. Nous pouvons faire la comparaison entre le domaine de la TCAO et l'interaction multimodale. En effet, en multimodalité, nombreuses sont les recherches sur l'usage synergique des modalités [Oviatt 1999] car il s'agit du cas sans doute le plus difficile à réaliser et non du cas le plus utile. De manière similaire, nombreux sont les travaux en TCAO sur les collecticiels où les utilisateurs sont distants : ce constat n'implique pas que les collecticiels pour des utilisateurs regroupés en un seul lieu ne soient pas utiles.

2.3. TAXONOMIE DU TRAVAIL COOPÉRATIF

La taxonomie des collecticiels selon A. Dix [Dix 1993] repose sur un modèle du travail coopératif. Ce modèle, comme le montre la Figure 4, identifie deux entités impliquées dans le travail coopératif : les participants (P) et les artéfacts du travail (A), c'est-à-dire les entités manipulées permettant aux participants d'interagir entre eux et avec le système. Par exemple, ces artéfacts peuvent être les outils mis à disposition comme un outil pinceau utilisé pour dessiner dans un tableau partagé.

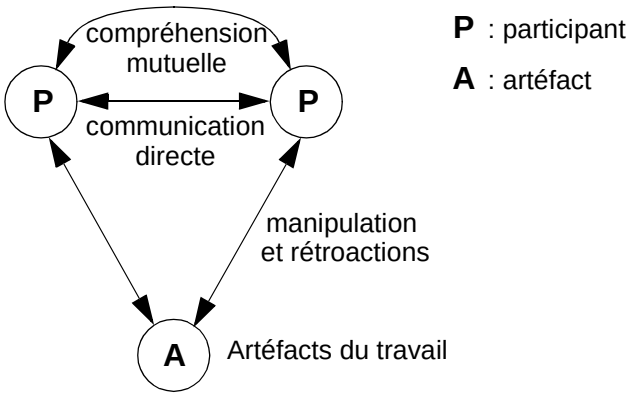


Figure 4
Taxonomie du travail coopératif [Dix 1993]

Ce modèle identifie les relations entre ces entités, c'est-à-dire les participants et les artéfacts du travail, qui caractérisent le travail coopératif. Il existe trois types de relation :

- la communication directe entre les participants au cours d'une activité (orale, écrite, gestuelle, etc),

- l'établissement d'une compréhension mutuelle pour mener des actions conjointes et,

- la manipulation des artéfacts.

Le but de la communication directe est d'échanger des informations mais aussi d'établir une compréhension mutuelle des actions qui vont être menées conjointement. Cette notion de compréhension mutuelle est essentielle pour le succès d'une action conjointe. La manipulation d'artéfacts évoque la manipulation d'objets physiques que l'on déplace ou modifie sur un bureau réel comme déplacer un livre, écrire sur une feuille, utiliser son téléphone, etc. La rétroaction est la perception par les autres utilisateurs du résultat de ces actions. Par exemple, s'apercevoir qu'un livre a été déplacé ou que quelque chose a été écrit sur la feuille.

A partir de ce modèle, A. Dix a identifié trois catégories [Dix 1993] de collecticiels (taxonomie du travail coopératif) relatives aux trois relations mises en évidence par ce modèle du travail coopératif :

- **Les systèmes de communication Homme-Homme médiatisée** (CHHM) désignent les systèmes dédiés à la communication directe entre les participants. Cette catégorie de collecticiels regroupe, entre autres, le courrier électronique, les forums de discussion, la vidéoconférence et les *mediaspace*. Nous retrouvons ici notre catégorie de la première taxonomie.

- **Les systèmes de réunions et d'aide à la décision** sont des collecticiels dont le but est de favoriser et d'aider à la compréhension mutuelle pour faciliter le déroulement de réunions ou la prise de décisions entre différents participants. Cette catégorie regroupe donc les systèmes d'aide à la décision (GDSS) et les systèmes de réunion virtuelle (bureau partagé) et réelle (tableau blanc réel et partagé).

- **Les systèmes d'espaces partagés** (*shared workspaces*) recouvrent les collecticiels mettant en œuvre des espaces partagés dans lesquels les participants peuvent manipuler des artéfacts. Il s'agit par exemple des éditeurs partagés ou des calendriers partagés.

En conclusion, les trois classes identifiées par cette taxonomie, basée sur un modèle du travail coopératif, pour caractériser un collecticiel (communication directe, compréhension mutuelle et manipulation d'artéfacts partagés), sont complémentaires aux dimensions espace et temps de la taxonomie précédente. En effet, considérons la communication directe : cette dernière peut concerner les neuf catégories de la taxonomie Espace-Temps, que ce soit en face-à-face (cas 1), par le biais de post-it (cas 7) ou par les notes prises dans le cas d'une édition partagée (cas 6). De plus, nous constatons que les neuf domaines d'application de la première taxonomie peuvent être rangés dans les trois classes identifiées ici.

2.4. SYNTHÈSE

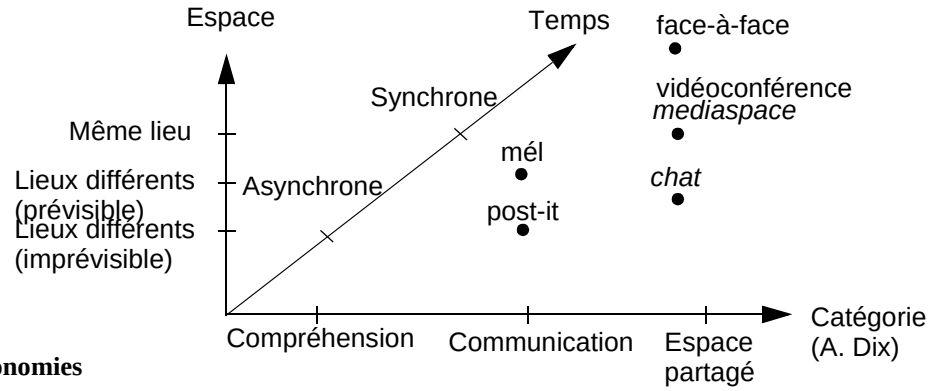


Figure 5
Schéma de synthèse des taxonomies

En synthèse, nous avons présenté trois taxonomies pour souligner l'étendue des possibilités dans le domaine du TCAO. Au cours de leur présentation successive, nous avons souligné leur complémentarité. A la Figure 5 de synthèse, nous combinons les axes de classification des deux dernières taxonomies. En effet nous partons des neuf cas de la taxonomie Espace-Temps et nous affinons chacune des cases de la matrice par les trois types de la taxonomie de A. Dix. La première taxonomie, reposant sur les domaines d'application, est par définition instable et doit être enrichie au fur et à mesure qu'un nouveau collecticiel traite d'un domaine d'application non encore répertorié. De plus, nous pouvons situer les neuf domaines d'application cités au sein de ces trois axes. A titre illustratif, nous situons à la Figure 5, les types de collecticiels de la première taxonomie que nous avions rangés sous la catégorie Communication dans le paragraphe 2.1.

En étudiant les taxonomies existantes des collecticiels, nous avons souligné leur diversité en terme de domaines d'application et d'usage. Cette diversité laisse entrevoir la difficulté de mise au point de méthodes et d'outils pour la conception et la réalisation de tels systèmes. Méthodes et outils pour la mise en œuvre des collecticiels font l'objet de la partie suivante.

3. Outils pour la mise en œuvre d'un collecticiel

Notre étude porte sur une catégorie d'outils, les outils de conception et de réalisation logicielles des collecticiels. Aussi dans les chapitres suivants, nous dressons un état de l'art ainsi qu'une analyse critique des outils existants de cette catégorie. Dans cette partie, nous souhaitons définir cette catégorie d'outils en cernant sa portée par rapport aux autres outils existants. Pour cela nous présentons un ensemble de méthodes et d'outils au sein d'un cadre fédérateur. L'objectif principal est de situer des outils existants les uns par rapport aux autres.

A des fins analytiques, nous avons choisi de présenter les méthodes et outils existants selon le cycle de vie du logiciel en V. Nous ne prenons pas position sur le fait que ce cycle est adapté ou non à la mise en œuvre d'un collecticiel. Nous nous servons de ce cycle de vie comme un canevas structurant. En particulier ce cycle de vie établit une distinction nette entre la conception ergonomique d'un collecticiel, l'espace IHM, et la conception et réalisation logicielles, l'espace logiciel, notre sujet d'étude.

Le paragraphe suivant présente les principales étapes du cycle en V et définit ainsi l'organisation de cette partie.

3.1. CYCLE DE VIE LOGICIEL

Les étapes du cycle de vie en V [McDermid 1984] sont représentées à la Figure 6. Nous avons adapté le cycle de vie au développement logiciel de systèmes interactifs. Il existe d'autres cycles de vie comme le cycle en cascade [Royce 1970] ou en spirale [Boehm 1981]. Notre propos n'est pas de comparer les différents cycles de vie mais d'en extraire les étapes principales afin de définir un canevas fédérateur pour situer ensuite les méthodes et outils dédiés aux collecticiels.

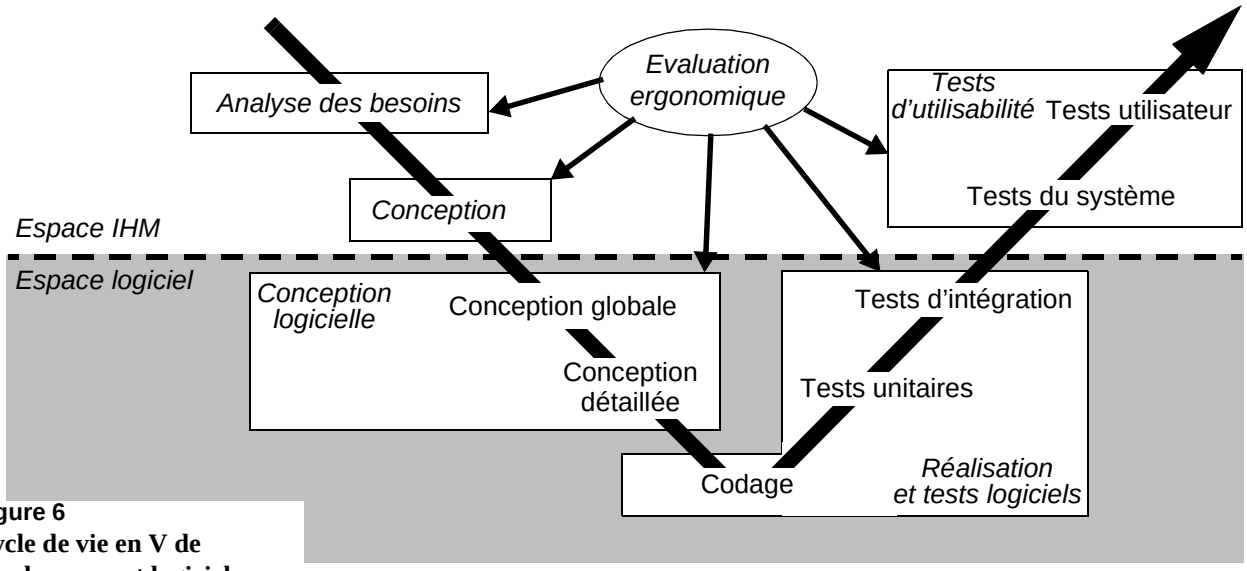


Figure 6
Cycle de vie en V de développement logiciel

Les étapes de ce cycle peuvent être organisées selon deux catégories : les étapes relevant de l'espace IHM, c'est-à-dire les étapes de conception ergonomique du système et d'élaboration d'un modèle d'interaction répondant à des requis identifiés à l'issue de l'étape d'analyse des besoins, et les étapes relevant de l'espace logiciel, c'est-à-dire les étapes de conception et de développement logiciels. L'évaluation ergonomique, comme le montre la Figure 6, intervient à toutes les étapes du cycle. Le but de cette phase est d'identifier les incohérences et les erreurs d'ergonomie pour pouvoir les corriger le plus tôt possible.

L'objectif de l'étape d'analyse des besoins est de déterminer les besoins afin de produire un logiciel qui réponde aux attentes des utilisateurs [Assali 1998]. Les résultats de cette étape sont consignés dans un cahier des charges. Cette étape consiste donc à cerner le domaine d'application et le rôle attendu du système tout en prenant en compte les contraintes de développement et d'utilisation. Le paragraphe 3.2 présente cette étape à travers trois outils pour l'analyse des besoins : l'étude ethnographique, le modèle Denver et la méthode GTA pour l'analyse de la tâche.

L'étape suivante est alors la conception du système. C'est au cours de cette étape que l'on identifie les requis fonctionnels (spécifications fonctionnelles) et les requis utilisateurs (spécifications externes). Le document de spécifications externes issu de cette étape reflète le système tel qu'il sera utilisé et perçu par l'utilisateur. Le paragraphe 3.3 traite d'abord des outils pour les spécifications fonctionnelles en présentant un modèle, le modèle du trèfle pour identifier les fonctionnalités d'un collecticiel. Les spécifications externes sont ensuite abordées par l'exposé d'un ensemble de propriétés ergonomiques spécifiques aux collecticiels.

L'étape de conception logicielle est une étape préliminaire à la phase de codage. A l'aide d'outils pour la conception logicielle, présentés au paragraphe 3.4, cette étape consiste à construire l'architecture logicielle décrivant la structure du logiciel à développer ainsi que son comportement à l'exécution.

L'étape suivante de codage dont la finalité est de produire un logiciel exécutable fait l'objet du paragraphe 3.5. Nous y présentons les outils de développement.

Enfin, le paragraphe 3.6 traite des outils pour l'évaluation ergonomique. Cette phase intervient à toutes les étapes du cycle et a pour but d'identifier et de corriger le plus tôt possible les erreurs de conception pouvant nuire à l'utilisabilité et à l'utilité du futur système.

3.2. OUTILS POUR L'ANALYSE DES BESOINS

Au cours de la phase d'analyse des besoins, le concepteur identifie les concepts du domaine d'application et élabore un modèle de tâches décrivant l'interaction avec le futur système. Pour déterminer ces concepts et analyser la tâche, le concepteur dispose d'outils hérités, en grande partie, des sciences sociales et en particulier de l'ethnographie.

Notamment, dans la première section, intitulée "Etude ethnographique", nous traitons de l'étude ethnographique qui consiste à étudier le comportement et les pratiques d'un groupe d'individus, par exemple, à partir d'observations sur le terrain ou à partir de questionnaires.

Dans la seconde section, intitulée "Modèle Denver", nous présentons le modèle Denver en tant que moyen complémentaire à l'étude ethnographique pour caractériser les besoins à partir des résultats issus de l'étude. En effet, bien que ce modèle ne soit pas complètement abouti, celui-ci offre une base de travail intéressante pour l'analyse des besoins. Il définit et organise un ensemble de critères permettant l'analyse du travail collaboratif ; en particulier, il offre des axes pertinents pour définir les situations et protocoles d'interaction

Enfin, dans la troisième section, intitulée "GTA et analyse de la tâche", nous présentons la méthode GTA qui est une méthode d'analyse et de description de la tâche qui exploite les résultats d'une étude ethnographique afin d'élaborer un arbre de tâches.

Etude ethnographique

Les travaux entrepris dans le domaine du TCAO marquent une rupture avec les méthodes en IHM largement influencées par la psychologie cognitive et du comportement et se placent dans une optique résolument sociale de par l'influence de disciplines telles que la sociologie ou l'ethnographie. Ces dernières ont une place importante dans les travaux sur les systèmes collaboratifs. Selon L. Bannon [Bannon 1998], le défaut principal de l'approche cognitive liée aux facteurs humains en IHM est de considérer l'utilisateur isolément et de modéliser son comportement indépendamment du contexte d'utilisation, de sa façon de travailler. Les travaux issus du monde du TCAO tendent donc à remettre en cause l'influence majeure de la psychologie cognitive en IHM et prônent pour une approche plus "sociale" qui prendrait en compte le contexte et les aspects sociaux de l'interaction.

Plus précisément, selon son origine étymologique, l'ethnographie est une branche de la sociologie qui consiste à recueillir une description du fonctionnement d'un groupe d'individus dans leur environnement par rapport à leurs usages, leurs relations, leur histoire, leur culture, etc. Une étude ethnographique permet ainsi de comprendre les pratiques et les activités. Le travail principal de l'ethnographe consiste d'abord à analyser l'activité dont il doit noter tous les phénomènes les plus significatifs, puis à les analyser. La deuxième phase consiste alors à collecter les

informations sur les activités de chaque individu du groupe. L'étude ethnographique se doit de décrire : le cadre de l'interaction (la localisation, le statut, etc.), les règles régissant l'organisation du groupe (rendre compte de la hiérarchie, de la répartition des rôles, des protocoles de communication, etc.), les rencontres informelles, les événements inattendus qui modifient la nature de l'interaction au sein de ce groupe [Van Der Veer 1996]. Enfin, selon Anderson [Anderson 1994], "l'étude ethnographique est une forme de reportage bien plus qu'une collecte de données" et l'ethnographe interprète ce qu'il observe. Il s'agit de capter l'aspect social du travail de groupe à travers ses pratiques, sa culture, son histoire, etc. L'étude ethnographique est donc un support essentiel pour la conception d'un collecticiel car elle permet, par l'étude des tâches accomplies, du comportement des participants à la tâche, et de l'environnement, de comprendre le fonctionnement complexe de l'action réalisée. Contrairement à l'étude d'un seul utilisateur, l'étude du fonctionnement d'un groupe est plus complexe car elle doit intégrer le comportement de chaque utilisateur ainsi que l'interaction entre utilisateurs. Cela est d'autant plus difficile lorsque le nombre d'utilisateurs varie au cours du temps et qu'il n'y a, à priori, aucune limitation de ce nombre.

Il existe d'autres approches qui tendent à intégrer les apports de l'ethnographie dans la réalisation de collecticiels comme les travaux fondateurs de L. Suchman [Suchman 1987] à travers la notion d'action située (une action doit être considérée par rapport à son contexte d'exécution), ou comme les travaux issus de l'école scandinave à travers la notion de conception participative (*participatory design*) [Kyng 1997]. Cette dernière approche vise à impliquer les utilisateurs tout au long du cycle de développement d'un collecticiel.

Modèle Denver

Le modèle Denver (*Denver model*) [Salvador 1996], résultat d'un groupe de travail (*workshop*) à la conférence *ACM CHI* (*Computer Human-Interaction*) en 1995, est une ébauche de méthode d'analyse des besoins pour les applications où la notion de groupe est un facteur dominant.

Ce modèle offre un cadre pour identifier la situation d'interaction et le protocole social d'interaction. Les résultats de l'étude ethnographique peuvent être transcrits en terme de situation d'interaction et de protocole d'interaction. Cette transcription a l'avantage d'organiser les résultats et de caractériser les différentes formes de situation et protocoles d'interaction au sein d'un groupe. Ce modèle est bien utilisé pour la phase d'analyse et aide le concepteur pour décrire les besoins.

La situation d'interaction et le protocole social d'interaction sont définis de la façon suivante :

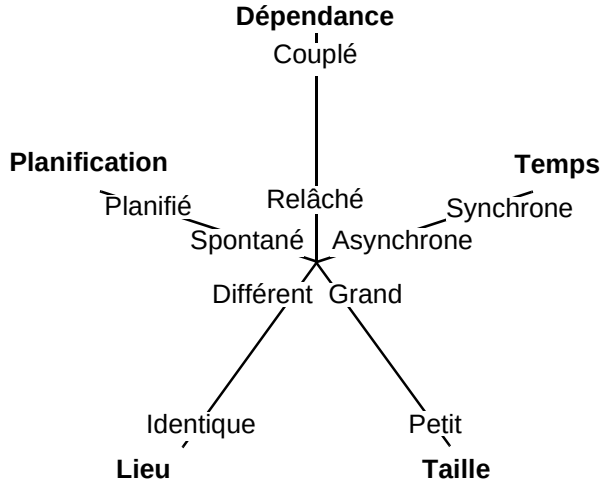


Figure 7
Situation d'interaction

- **Situation d'interaction** (*Interactive situation)*
  Les situations d'interactions sont définies par les relations entre le temps, l'espace et le couplage de l'interaction (dépendance entre les participants). Les participants peuvent définir un petit ou un grand groupe, être proches ou loin, avoir des interactions spontanées ou au contraire prévues, être dépendants d'un autre participant dans la progression de leurs travaux ou ne pas l'être. Enfin ils peuvent être en interaction synchrone (temps réel) ou asynchrone. Nous représentons ces cinq axes sur le diagramme de la Figure 7.

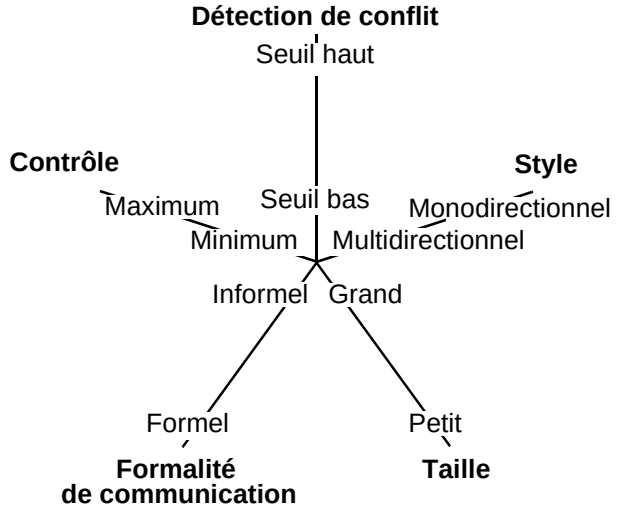


Figure 8
Protocole social d'interaction

- **Protocole social d'interaction** (*Interactive social protocol*)
  De tels modèles de protocoles sont analogues aux axes des situations d'interaction ci-dessus. Un protocole social se réfère aux séquences possibles d'échanges de signaux et d'information qui déterminent et identifient les discussions. Ceci inclut le concept d'hétérogénéité qui lui se réfère à la malléabilité des structures et des fonctions d'un groupe. Nous retrouvons ici cinq axes : la taille du groupe, les formalismes de communication, le contrôle des outils, la détection et la

prise en compte des idées échangées (*contention detection* et *resolution*). La Figure 8 présente les axes qui caractérisent un protocole.

Toutefois, nous avons adapté ce modèle pour spécifier et concevoir le système CoVitesse [CoVitesse 2002] [Laurillau 2000] [Laurillau 1999b], système de navigation collaborative sur le *WWW*, dont une description de la mise en œuvre logicielle est présentée au Chapitre VII. C'est à l'aide de ce modèle, suite à une synthèse d'études ethnographiques, que nous avons mis en évidence et caractérisé quatre types de navigation collaborative qui sont offerts par le système CoVitesse. L'utilisation complète de ce modèle pour la conception du système CoVitesse est détaillée dans [Laurillau 1999a] et dans l'annexe de ce manuscrit.

GTA et analyse de la tâche

Jetant un pont entre les pratiques en IHM et l'ethnographie, la méthode *GTA* [Van Der Veer 1996] (*Groupware Task Analysis*) est un outil pour l'analyse de la tâche. A l'issue de cette étape, une structure hiérarchique des tâches est souvent obtenue [Coutaz 1990] [Shepherd 1989]. Plusieurs modèles de tâches sont construits au cours de la conception. GTA identifie trois modèles de tâches :

- *Modèle de tâches 1* : un système est motivé par la nécessité d'informatiser des pratiques de travail. Pour que le système soit conforme à ces méthodes de travail, il est nécessaire de capter et décrire les différentes tâches exécutées en situation réelle. Ceci fait l'objet de ce premier modèle de tâches qui capture les résultats de l'étude ethnographique. Ce premier modèle de tâches constitue un support à l'expression de besoins du système, définis au cours de la phase suivante.

- *Modèle de tâches 2* : cette seconde étape consiste à déterminer quels sont les besoins du système en se basant sur le premier modèle de tâches et aboutit à un second modèle de tâches. Il s'agit du point de vue système de la tâche, c'est-à-dire l'ensemble des tâches que l'utilisateur pourra effectuer avec le système. En passant à l'étape suivante, c'est-à-dire l'élaboration du modèle de la machine virtuelle, nous quittons l'étape d'analyse des besoins pour passer à l'étape de spécification du système. En ce sens, nous raisonnons à un niveau d'abstraction moins élevé.

- *Modèle de la machine virtuelle d'un utilisateur* (*user's virtual machine*) : cette dernière étape est l'élaboration d'un modèle de tâches "système" embarqué par le collecticiel selon un point de vue technologique. Il s'agit d'une description complète des tâches systèmes décomposées en actions physiques, c'est-à-dire la description

des manipulations à réaliser avec l'interface pour exécuter une action. En ce sens, GTA couvre aussi en partie la phase de conception de l'espace IHM dans le cycle de vie en V. Ce modèle est un élément de spécification du système car ce modèle fait partie intégrante des spécifications externes et fonctionnelles. Ce point est abordé dans le paragraphe suivant.

Les formalismes pour décrire les arbres de tâches sont, par exemple, des formalismes orientés objet comme MAD [Scapin 1989] (Méthode Analytique de Description). De plus, pour élaborer ces modèles de tâches, GTA repose sur trois concepts :

- les participants : acteurs d'une tâche dans un rôle donné au sein d'une organisation,
- le travail : identifié par une tâche et structuré par un ensemble de sous-tâches et d'actions exécutées selon les règles sociales imposées,
- une stratégie : les tâches à réaliser en fonction d'un rôle donné,
- une situation de travail : identifiée par les objets manipulés et l'environnement de travail.

Ayant définis les besoins, nous quittons l'étape d'analyse des besoins pour passer à l'étape de conception et de spécification du système traitée dans le paragraphe suivant.

3.3. OUTILS POUR LES SPÉCIFICATIONS FONCTIONNELLES ET EXTERNES

A partir de l'analyse de besoins, il convient alors de concevoir le modèle de l'interaction. Les spécifications externes d'un collecticiel s'expriment en termes de requis fonctionnels et requis utilisateurs.

Les requis fonctionnels cernent les fonctionnalités offertes par le collecticiel. Le modèle du trèfle, présenté dans la première section intitulée "Spécifications fonctionnelles et modèle du trèfle", propose un cadre pour organiser et définir les fonctionnalités d'un collecticiel.

Les requis utilisateurs concernent l'utilisabilité du collecticiel et donc l'interface du collecticiel. Communément, l'utilisabilité se décline en propriétés ergonomiques [Coutaz 2001]. Les propriétés ergonomiques propres aux collecticiels sont présentées dans la seconde section intitulée "Spécifications externes et propriétés ergonomiques".

Spécifications fonctionnelles et modèle du trèfle

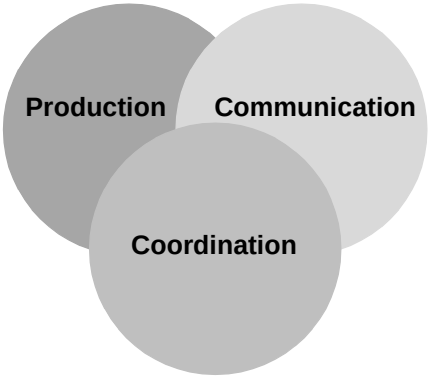


Figure 9
Modèle du trèfle [Salber 1995]
Ce modèle organise les fonctionnalités d'un collecticiel selon trois espaces : production, communication et coordination.

Le modèle du trèfle [Salber 1995], inspiré du modèle conceptuel d'un collecticiel proposé par C. Ellis [Ellis 1994], fournit un cadre conceptuel utile pour déterminer les requis fonctionnels et mener une analyse fonctionnelle. En effet, selon ce modèle présenté à la Figure 9, un collecticiel couvre trois espaces fonctionnels :

- **L'espace de production** concerne l'ensemble des fonctionnalités de production d'objets partagés tels que des documents communs et la gestion des accès à ces données partagées. Par exemple, les éditeurs partagés, définis dans le paragraphe 2.1, sont dédiés à la production.

Figure 10
InTouch [Brave 1998]
Le système InTouch est un jeu à retour d'effort collaboratif constitué de rouleaux fixés sur un socle. Dès qu'un rouleau subit une rotation, la force résultante est appliquée aux rouleaux correspondants des autres socles distants.

- **L'espace de communication** correspond aux fonctionnalités permettant l'échange d'information entre les acteurs du collecticiel. Cet échange est de la communication homme-homme médiatisée [Salber 1995] (CHHM). Il existe différents modes de communication comme, par exemple, l'audio (téléphone), la vidéo (*mediaspace*), le textuel (messageries), la gestuelle (langue des signes) ou l'haptique (communication à travers un système à retour d'effort comme, par exemple, le système InTouch [Brave 1998] présenté à la Figure 10). Les systèmes de vidéoconférence et les *mediaspace*, définis dans le paragraphe 2.1, sont dédiés à la communication homme-homme médiatisée.

- **L'espace de coordination** correspond aux fonctionnalités dédiées à l'assignation de tâches et de rôles aux différents acteurs d'une activité collaborative. Ces fonctionnalités ont pour but de coordonner les acteurs afin de réaliser une œuvre commune. Cette coordination peut s'exprimer en terme de planification de tâches. Par exemple, les systèmes *workflow*, définis dans le paragraphe 2.1, sont des systèmes dédiés à la planification de transferts de documents entre différents intervenants au cours d'un processus industriel.

Les trois espaces fonctionnels d'un collecticiel étant définis, il convient néanmoins de noter que certaines fonctionnalités peuvent être à l'intersection de plusieurs espaces comme le souligne la Figure 9 (intersection entre les trois espaces). Ainsi, la distinction selon les trois espaces n'est pas stricte car il est tout à fait possible qu'une activité de coordination ait lieu suite à une activité de communication. Par exemple, la prise de rendez-vous par téléphone est une activité de coordination basée sur une activité de communication. Ce point est étudié plus en détail dans les chapitres suivants.

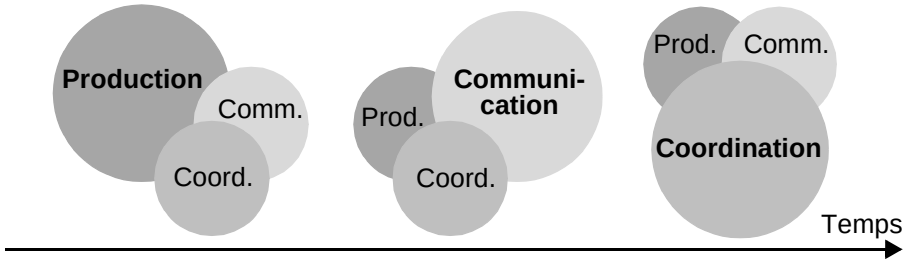


Figure 11
Evolution dans le temps du rôle fonctionnel d'un collecticiel

De plus, le rôle fonctionnel global d'un collecticiel, comme le montre la Figure 11, peut évoluer au cours du temps : cette évolution peut être modélisée grâce au modèle du trèfle. Par exemple, lors de l'utilisation d'un système dédié à la production, il est possible, à un moment donné, que l'activité de groupe soit centrée sur la communication en vue de se coordonner afin de redéfinir l'activité de production. Cette approche permet de prendre en compte la variabilité dans les activités de groupe au cours du temps.

Comme nous l'avons évoqué, le modèle du trèfle est un modèle de conception utile pour déterminer et organiser les fonctionnalités d'un collecticiel selon la production, la communication et la coordination. Par exemple, F. Tarpin propose une méthode pour spécifier les fonctionnalités d'un collecticiel en fonction d'un ensemble de questions thématiques, en s'appuyant sur les trois espaces fonctionnels du modèle du trèfle [Tarpin 1997] :

- **Production** : structuration et gestion des données, par identification des données partagées et de leurs structures, par modélisation du support

du travail et des opérations de production et par identification des caractéristiques de partage.

- **Coordination** : organisation du processus de travail par identification des tâches et modélisation du processus de travail et par identification des rôles et des modes de coopération (couplage).

- **Communication** : choix des modes de communication et des protocoles de conversation (rôles et attribution de responsabilités).

Spécifications externes et propriétés ergonomiques

Le document de spécifications externes décrit le système tel qu'il sera perçu et utilisé par l'utilisateur. Ces spécifications détaillent le modèle d'interaction proposé à l'utilisateur pour qu'il puisse interagir avec le système.

Comme toute application logicielle, la production d'un collecticiel doit répondre à des exigences et à un niveau de qualité requis. En génie logiciel, il existe de nombreux facteurs [McCall 1977] pour quantifier la qualité du logiciel comme par exemple la modifiabilité du logiciel. Dans le domaine de l'IHM, les efforts se concentrent plus particulièrement sur l'utilisabilité d'un système interactif qui est un facteur de qualité [McCall 1977]. Ce facteur définit la qualité d'utilisation d'un système interactif et sert de guide pour la conception et l'évaluation. Nous détaillons ce dernier point dans le paragraphe consacré à l'évaluation ergonomique. Une approche en IHM [IFIP 1996] est de décliner ce facteur selon trois facteurs qualité : la facilité d'apprentissage (facilité d'apprentissage pour un utilisateur novice à manipuler le système), la souplesse (capacité du système à proposer un éventail de choix à l'utilisateur pour exécuter une action) et la robustesse de l'interaction (capacité du système à faciliter l'accomplissement d'une action). McCall affine les facteurs en critères ; ici les trois facteurs sont affinés en un ensemble de propriétés ergonomiques. Par exemple, l'adaptativité est une propriété ergonomique qui est liée au facteur de souplesse et traduit la capacité du système à s'adapter à l'utilisateur sans une intervention explicite de sa part [IFIP 1996]. L'ensemble de ces propriétés guide le concepteur dans son activité de conception et notamment pour les spécifications externes du système.

Dans le cas des collecticiels, de nouvelles propriétés ergonomiques sont spécifiques à l'activité de groupe. Nous présentons les propriétés ergonomiques les plus étudiées issues de [Salber 1995] [UsabilityFirst 2002] :

- **Conscience de groupe et rétroaction de groupe** (*group awareness* et *feedthrough*) : lors d'une activité en groupe, par un exemple un sport tel que le basket, les joueurs ont toujours un oeil sur ce que font les

autres membres du groupe pour agir et réagir en fonction des tâches réalisées (et de leurs résultats). Le groupe est efficace, en se basant encore sur la métaphore sportive, que s'il existe un esprit d'équipe qui s'appuie sur la conscience de groupe. Il s'agit d'informer un utilisateur de l'activité en cours des autres utilisateurs. Le terme employé dans la littérature est la **conscience de groupe**. Cette source d'information n'est pas essentielle à la réalisation de la tâche en cours mais elle y contribue dans le sens où il s'agit d'informations relatives à l'état d'avancement de l'activité des autres utilisateurs. Par exemple, les *mediaspace*, définis dans le paragraphe 2.1, sont une classe de collecticiels où la notion de conscience de groupe est primordiale. En effet, le but principal est de maintenir une conscience de groupe forte entre des individus dispersés géographiquement. Néanmoins, les *mediaspace* soulèvent de nombreux problèmes liés à la protection de l'espace privé (propriété abordée au point suivant) de par la présence d'une caméra dans les bureaux et espaces communs.

La propriété de conscience de groupe traduit donc la capacité du système à rendre observable des informations sur l'activité de groupe indépendamment de la tâche en cours.

Une propriété liée à la conscience de groupe est celle de **rétroaction de groupe** qui traduit la capacité du système à rendre observable des informations sur l'activité de groupe pertinente pour la réalisation de la tâche en cours. Cette propriété est liée au *WYSIWYS* abordé dans la suite.

- **Protection de la vie privée** (*privacy*) : la propriété de protection de la vie privée est relative à la protection des informations privées [UsabilityFirst 2002]. De nombreuses propriétés sont en relation avec celle-ci, à savoir la propriété d'observabilité et de publication, et la propriété de réciprocité et d'identification. La conception d'un collecticiel impose un bon dosage entre la propriété de protection de la vie privée et la propriété de conscience de groupe.

- **Observabilité et publication** : la propriété d'observabilité n'est pas nouvelle puisqu'elle est considérée pour les systèmes interactifs mono-utilisateurs [IFIP 1996] (page 84). Pour les systèmes interactifs, l'observabilité traduit la capacité du système à rendre observable son état interne et à offrir des moyens à l'utilisateur pour qu'il puisse le consulter. Dans le cas des collecticiels, cette notion d'observabilité s'applique aussi aux informations concernant les autres utilisateurs et leurs activités. D. Salber [Salber 1995] désigne cette propriété par la notion d'observabilité publiée. Des informations observables de

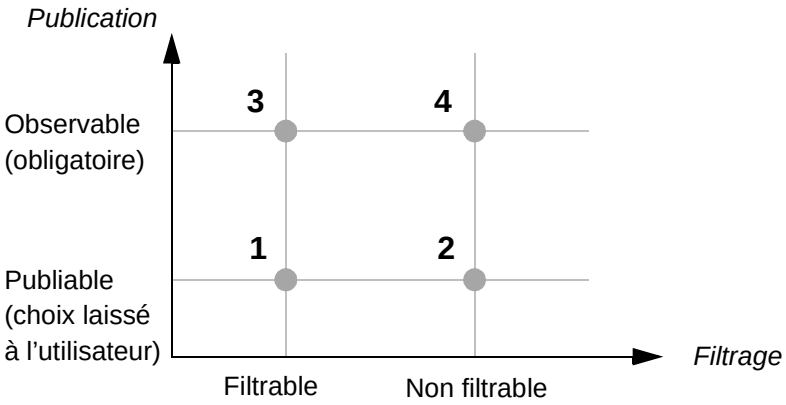


Figure 12
Quatre niveaux d'observabilité [Laurillau 1999a]

l'utilisateur, nous considérons trois formes d'observabilité [Laurillau 1999a] :

* ***Observable*** : ce qui est souhaitable qu'un utilisateur percoive,
* ***Publiable*** : ce qu'un utilisateur décide de rendre observable de lui-même,
* ***Filtrable*** : la forme de ce qui est observable. Les informations ainsi filtrées donnent de l'utilisateur l'apparence qu'il a choisie.

La différence entre un filtrage et une information publiée se situe dans la véracité des informations : quand une information est publiée, celle-ci est à priori exacte, quand celle-ci est filtrée, elle est modifiée, elle est modelée à sa propre façon.

Nous définissons [Laurillau 1999a] quatre niveaux d'observabilité, présentés à la Figure 12 : les quatre niveaux sont définis selon deux axes orthogonaux, *Filtrage* et *Publication*. Pour chaque entité, le concepteur doit décider quel niveau d'observabilité est nécessaire pour que l'utilisateur puisse réaliser sa tâche tout en préservant son espace privé et celui des autres. Ainsi, un moyen possible de protéger son espace privé est de doter le système de filtres des informations paramétrables par l'utilisateur. Par exemple, dans le *mediaspace* Comedi [Coutaz 1997], un ensemble de filtres de l'image vidéo est proposé pour rendre compte de l'activité dans un bureau tout en préservant l'intimité des utilisateurs. Parmi les filtres proposés, l'utilisateur peut choisir de masquer son image à l'aide d'un "store vénitien". Ce filtre tout en masquant les personnes permet néanmoins de deviner l'activité dans le bureau, comme le nombre de personnes. Dans le système CoVitesse, présenté dans le Chapitre VII, le système propose un ensemble de filtres pour ne rendre observable qu'une partie des pages web visitées. Un de ces filtres permet de masquer toutes les pages visitées contenant un mot particulier, ce dernier définissant le filtre.

- **Réciprocité** (*reciprocity*) **et identification** : les propriétés de réciprocité et d'identification sont des cas particuliers de l'observabilité. **La propriété de réciprocité** exprime la capacité du système à fournir des moyens pour que des utilisateurs puissent s'observer mutuellement. Ainsi, si un utilisateur obtient des informations sur un autre utilisateur, ce dernier doit pouvoir obtenir le même type d'information sur le premier. Par exemple, si un utilisateur peut observer un autre utilisateur à travers une vidéo, ce dernier doit aussi pouvoir l'observer. **La propriété d'identification** traduit la capacité du système à offrir des moyens pour qu'un utilisateur puisse identifier les auteurs d'une action ou d'une décision.

- ***WYSIWIS*** (*What You See Is What I See*) **et couplage de l'interaction** (*coupling*) : *WYSIWIS* signifie en français "ce que tu vois est ce que je vois". Dans la suite du manuscrit, nous utilisons systématiquement l'abréviation *WYSIWIS*.
  La **propriété de *WYWISIS*** [Stefik 1987] se traduit en principe par une vue identique entre plusieurs utilisateurs ; dès qu'un utilisateur apporte une modification à la vue courante (par exemple, le déplacement d'une barre de défilement). Cette notion ne se limite pas à l'interface car cette propriété impose que toutes modifications, que ce soit au niveau de l'interface ou au niveau fonctionnel, soient diffusées à tous les autres utilisateurs.
  La modification apportée à la vue courante est répercutée dans les autres vues avec un délai plus ou moins important suivant le **degré de couplage** (fortement couplé ou faiblement couplé) [Dewan 1995]. Il existe deux modes *WYSIWIS* qui ont un impact direct sur la nature du couplage de l'interaction : le mode *WYSIWIS* strict et le mode relâché. Dans le premier cas, cela signifie que l'interaction est fortement couplée et que tous les utilisateurs disposent nécessairement d'une vue unique. En l'occurrence, l'espace privé est très limité. Par exemple, certains collecticiels ne disposent que d'un seul pointeur de souris soumis à une politique de partage. Dans le second mode, l'interaction est plus souple et les utilisateurs disposent de leur propre vue et de leur propre espace privé.
  Cette propriété exprime une tension entre la propriété de protection de la vie privée et la conscience de groupe. En effet, un mode *WYSIWIS* strict aura tendance à favoriser la conscience de groupe (par exemple il n'y a qu'un seul pointeur et tous les utilisateurs sont informés de la manipulation du pointeur) tandis que le mode *WYSIWIS* relâché aura tendance à favoriser la protection de l'espace privé (par exemple l'utilisateur dispose de son propre pointeur et travaille dans son espace privé mais n'est pas nécessairement informé de l'activité des autres).

- **Viscosité** : la viscosité [Green 1990] est un phénomène social qui exprime l'incidence d'une action, c'est-à-dire les effets secondaires d'une action, sur l'activité des autres utilisateurs. Ces effets secondaires peuvent se traduire par une surcharge de travail pour les autres utilisateurs, c'est-à-dire par l'apparition de tâches supplémentaires. Il s'agit d'une forme particulière de la propriété de couplage de l'interaction. Par exemple, dans un éditeur de dessin partagé, si un utilisateur décide de déplacer la zone de dessin commune avec une barre défilement collaborative, les autres utilisateurs sont obligés de s'interrompre et de recentrer leur outil de dessin pour se replacer au bon endroit et reprendre leurs activités.

3.4. OUTILS POUR LA CONCEPTION LOGICIELLE

Tandis que dans le paragraphe précédent nous avons proposé un ensemble d'outils pour la conception ergonomique du collecticiel, nous nous intéressons à la conception logicielle en faisant l'hypothèse que le collecticiel est conçu et spécifié et qu'il convient maintenant de le développer.

Le développement du logiciel se déroule en deux grandes étapes, comme indiqué à la Figure 6 : la phase de conception logicielle, qui se décompose elle-même en deux sous étapes de conception globale et détaillée, et la phase de réalisation logicielle (codage et tests unitaires). Cette approche est commune au développement de tout système interactif mais est encore plus complexe dans le cas des collecticiels. L'étape de conception globale est donc une étape charnière puisqu'elle a la charge de faire le lien entre l'espace IHM et l'espace logiciel. C'est au cours de cette étape qu'il convient de construire une architecture logicielle. Il existe deux types d'architecture logicielle : l'architecture conceptuelle et l'architecture implémentationnelle. La première décrit une organisation des fonctionnalités en modules et décrit les protocoles de communication entre ces modules. La seconde est l'implémentation (i.e. la version programmée) de l'architecture conceptuelle qui est dépendante des outils de codage. Ce dernier point est abordé dans le paragraphe suivant.

Pour concevoir l'architecture conceptuelle du système, il convient d'utiliser un modèle d'architecture : l'usage commun assimile une architecture conceptuelle à un ensemble organisé de composants dont les interactions sont médiatisées par des entités spécialisées ou connecteurs. Le processus de conception d'une architecture conceptuelle recouvre les activités suivantes : définition de la décomposition fonctionnelle du système, identification de son organisation structurelle, allocation des fonctions à la structure et définition de la coordination entre les entités de la structure. Le résultat de ces quatre activités constitue la structure modulaire ou architecture conceptuelle.

Les modèles d'architecture pour collecticiels sont principalement conceptuels comme Clock [Graham 1996], PAC* [Calvary 1997] ou le modèle de Dewan [Dewan 1999]. Ces modèles définissent des structures modulaires normalisées. Par exemple, le modèle de Dewan préconise un empilement de couches partagées et répliquées communiquant entre elles par échanges d'événements.

Comme nous l'avons évoqué, avec l'architecture logicielle conceptuelle nous quittons l'espace de conception proprement dit du système interactif pour pénétrer dans l'espace logiciel réservé aux informaticiens. La transition de l'espace IHM à l'espace logiciel est toujours une étape délicate puisqu'il s'agit de transcrire les concepts identifiés au cours de l'analyse des besoins et de la conception de l'interface sous forme d'une organisation de fichiers, d'algorithmes et de structures de données. Cette situation charnière [Nigay 1997] se traduit par une tension permanente entre d'une part les contraintes techniques imposées par les outils de mise en œuvre et d'autre part la satisfaction des requis de l'utilisateur, des spécifications externes de l'IHM et des spécifications fonctionnelles du système. Aussi, il est important que les modèles d'architecture conceptuelle véhiculent, outre des propriétés logicielles, des propriétés ergonomiques et intègrent des éléments de conception du système interactif.

La notion d'architecture logicielle, conceptuelle et implémentationnelle, et de modèle d'architecture pour les collecticiels fait l'objet du chapitre suivant.

Après avoir conçu l'architecture logicielle conceptuelle dans le cycle de vie, il convient ensuite de concevoir l'architecture implémentationnelle puis de développer le système. Nous étudions au paragraphe suivant les outils disponibles pour cette phase de codage.

3.5. OUTILS POUR LA RÉALISATION LOGICIELLE

L'étape de la réalisation logicielle du collecticiel consiste à traduire l'architecture conceptuelle, issue de l'application d'un modèle, en une architecture implémentationnelle puis de développer le système.

Actuellement, le codage des applications logicielles ne peut se faire sans l'utilisation d'outils de développement dans le but de simplifier la tâche du développeur et de lui éviter de tout réinventer. Cependant, les outils ne font pas tout car ils offrent essentiellement des services élémentaires et la majeure partie du travail est à la charge du développeur. Ces outils sont constitués d'un capital de fonctionnalités que le développeur peut utiliser et assembler pour produire le système. Dans le cas des collecticiels, les outils de développement deviennent indispensables étant donné la complexité de mise en œuvre de ces systèmes (accès concurrents, perte d'information sur le réseau, etc). Dans ce paragraphe, nous présentons deux types d'outils : les services logiciels, et les boîtes à outils et

infrastructures logicielles. Les services logiciels sont des regroupements de fonctionnalités capables de choisir l'algorithme le plus adapté à une situation donnée, réduisant ainsi la charge de développement. Par exemple, dans le cas des collecticiels, un service de verrouillage des données doit être capable de choisir entre un algorithme de verrouillage optimiste ou pessimiste en fonction de la situation. Les boîtes à outils et infrastructures logicielles reposent sur un ensemble de services permettant ainsi le développement d'applications.

La première section présente un ensemble de services logiciels spécifiques aux collecticiels. La seconde section traite ensuite des boîtes à outils et des infrastructures logicielles qui reposent sur un ensemble de services.

Services logiciels pour les collecticiels

Le développement logiciel d'un collecticiel repose sur des services logiciels. De nombreux services généraux ont été identifiés pour mettre en œuvre la collaboration. Bien sûr, un collecticiel n'exploite pas nécessairement tous les services. Nous listons ci-dessous les services logiciels les plus fréquemment traités dans la littérature du domaine [Dewan 2001] [UsabilityFirst 2002] :

- **Gestion de sessions** (*session management*) : le gestionnaire de sessions est un service qui maintient à jour la liste des sessions et des différents participants. Une session est une période pendant laquelle les utilisateurs participent à l'activité de groupe. L'accès à une session en cours en régie par des droits d'accès. C'est à ce niveau que le gestionnaire met en place une politique de gestion des participants rejoignant une session en cours (*latecomers*) en donnant la possibilité de rejouer les actions réalisées dans le passé.

- **Contrôle de la concurrence** (*concurrency control*) : ce service a pour rôle de veiller à ce que des actions concurrentes puissent accéder aux objets partagés et que l'état du système reste cohérent. Une technique courante consiste à verrouiller (*locking*) les objets partagés. Notamment, il existe de nombreux mécanismes tels que le verrouillage optimiste ou pessimiste [Prakash 1999].

- **Contrôle d'accès** (*acces control*) : ce service gère les droits d'utilisation des outils et le droit d'exécution de certaines fonctionnalités. Les droits d'accès sont définis en fonction des rôles assignés aux différents participants. Le mécanisme de "contrôle d'utilisation" (*floor control*) a la charge de répartir les plages d'utilisation d'un outil. Par exemple, il peut s'agir du droit d'utilisation d'un tableau blanc partagé par une seule personne pendant un certain laps de temps. Ce mécanisme de "contrôle d'utilisation" existe sous

plusieurs formes comme, par exemple, le mécanisme de prise de parole (*turn-taking*).

- **Comparaison et fusion de données** (*diffing* et *merging*) : ces mécanismes, souvent associés aux éditeurs partagés asynchrones, permettent de comparer et de fusionner différentes versions d'un même document. Il existe de nombreux algorithmes de fusion dont l'algorithme de fusion divergente avec garantie de consistance de P. Dourish [Dourish 1996a] : cet algorithme autorise plusieurs versions d'un document à un instant donné, tout en garantissant que la fusion ultérieure des différentes versions sera cohérente.

- **Défaire-refaire** (*undo-redo*) : ce service, classique dans les systèmes interactifs mono-utilisateurs, est complexe à mettre en œuvre dans le cas des collecticiels. En effet, ce service doit veiller à ce que les actions concurrentes de défaire-refaire n'entraînent pas un état incohérent ou non désiré du système. Par exemple, dans le cas d'un éditeur de dessin partagé, un utilisateur décide d'annuler la dernière action, c'est-à-dire le dessin d'un cercle. Au même moment, un autre utilisateur décide de dessiner un rectangle. Ce service doit alors veiller à ce que le cercle soit supprimé au lieu du rectangle, sinon l'état obtenu est incompatible avec l'état désiré. Ce service doit respecter exactement dans quel ordre les actions ont été réalisées. Par exemple, différents modèles et algorithmes pour mettre en œuvre ce service sont décrits dans [Choudary 1995].

- **Synchronisation des présentations** (*coupling*) : ce service a la charge de synchroniser toutes les instances de la présentation du collecticiel en propageant les événements graphiques. Cette dépendance entre les différentes vues est plus ou moins forte selon le degré de couplage (*WYSIWIS* strict ou relâché). Par exemple, si un utilisateur, dans le cas d'un collecticiel disposant d'un unique pointeur de souris partagé (*WYSIWIS* strict), déplace sa souris, alors le pointeur doit se déplacer automatiquement au niveau de toutes les autres vues. De nombreux travaux portent sur l'étude du couplage des interfaces [Dewan 1995].

- **Notification** : le service de notification [Patterson 1996] a la charge de propager les changements d'état du système et d'informer les différents clients de ces changements d'état. Ce service est similaire au précédent mais celui-ci relève du Noyau Fonctionnel. Ce service participe au maintien de la conscience de groupe. D. Ramduny et A. Dix [Ramduny 1998] ont notamment étudié les apports de ce service et les différentes configurations pour le mettre en œuvre.

Boîtes à outils et infrastructures logicielles

Les outils de développement sont les boîtes à outils pour les collecticiels (*groupware toolkit*) et les infrastructures, deux types d'outils qui s'appuient sur des services comme ceux exposés dans la section précédente. Nous définissons une infrastructure comme une boîte à outils pour les collecticiels imposant un schéma d'exécution statique ou dynamique. Les outils de développement de collecticiels sont présentés et analysés dans le Chapitre V. Citons ici, à titre d'exemple, la boîte à outils GroupKit [Roseman 1992] qui est certainement l'outil de développement de collecticiels le plus connu dans le monde du TCAO. Cette boîte à outils met à disposition du développeur de nombreuses fonctionnalités comme, par exemple, le pointeur de souris collaboratif. Ce pointeur est un objet graphique qui répond au mouvement de la souris d'un utilisateur. Ainsi dès que l'utilisateur bouge sa souris, ce pointeur est déplacé dans toutes les vues. Il s'agit ici d'un exemple de service, en l'occurrence un service de couplage de l'interaction, que la boîte à outils intègre : ce service évite donc au développeur de reconcevoir cette fonctionnalité qui aurait nécessité de programmer la gestion des différentes vues du pointeur et de programmer l'association entre le déplacement de la souris et le déplacement du pointeur dans toutes les vues.

Pour conclure sur les outils de réalisation logicielle de collecticiels, il est important de souligner le fait que ces outils, comme GroupKit [Roseman 1992] ou COCA [Li 1999], n'éliminent pas la nécessité d'une architecture conceptuelle élaborée lors de l'étape précédente de conception logicielle. L'architecture conceptuelle est un artefact incontournable tant que les outils impliquent une phase de programmation. Sans une architecture conceptuelle, le collecticiel est plus difficile à développer, à modifier, à étendre et à maintenir. Une grande difficulté est de disposer d'outils adaptés qui offrent un haut niveau d'abstraction, pour réduire à la fois le coût de développement et pour faciliter le passage de l'architecture conceptuelle à la réalisation logicielle. L'architecture conceptuelle intégrant des concepts issus de l'espace IHM (Figure 6) en terme de composants logiciels, les outils doivent donc permettre d'exprimer ces concepts liés à l'interaction à un coût réduit.

3.6. OUTILS POUR L'ÉVALUATION ERGONOMIQUE

L'évaluation ergonomique intervient à toutes les étapes du cycle de vie logicielle, quel que soit le type de système interactif (mono-utilisateur ou collecticiel). L'évaluation ergonomique permet de détecter les problèmes d'utilisabilité du collecticiel à priori lors des phases de conception ou à posteriori, lorsque le système interactif a été développé. Dans ce dernier cas, les résultats de cette évaluation expérimentale peuvent donner lieu à une nouvelle itération du cycle de conception.

Il existe donc deux types de techniques pour l'évaluation, les techniques prédictives (à priori) et les techniques expérimentales (à posteriori). Le premier type de techniques repose sur des modèles théoriques tels que le modèle *ICS* [Barnard 1987] (*Interactive Cognitive Subsystems*, modèle cognitif de l'utilisateur) ou des modèles de performance comme, par exemple, le modèle *GOMS* [Card 1983] (*Goals, Operators, Methods and Selection Rules*), ainsi que l'utilisation d'heuristiques comme celles proposées par J. Nielsen [Nielsen 1990]. Ces heuristiques peuvent être liées à des propriétés ergonomiques, comme celles que nous avons exposées au paragraphe 3.3. Le second type de techniques repose sur une évaluation expérimentale du système à partir de prototypes, de maquettes ou du produit final, par observation des utilisateurs en situation d'utilisation. Notons enfin la technique du magicien d'Oz [Dahlbäck 1993] qui permet d'obtenir des résultats expérimentaux alors que le système n'est pas dévelo ppé : il s'agit donc d'une technique prédictive qui consiste à simuler le comportement du système, à l'insu de l'utilisateur observé, par un compère humain. A l'issue d'une séance, les observations sont souvent complétées par un questionnaire.

L'évaluation d'un collecticiel est nettement plus complexe que l'évaluation d'un système interactif mono-utilisateur. En effet il est difficile et coûteux de mener une évaluation car cela nécessite de disposer d'un grand nombre d'utilisateurs et de former des groupes homogènes. Or de nombreux critères doivent être pris en compte pour caractériser un groupe : la taille, la représentativité sociale, le niveau de compétence dans le domaine de la tâche, le niveau de compétence d'utilisation des outils informatiques, etc. Les techniques d'évaluation des systèmes mono-utilisateur doivent donc être adaptées afin de traiter ces nouveaux paramètres.

Or les travaux dans ce domaine sont pauvres. Dans la littérature, nous avons relevé une technique basée sur des heuristiques [Backer 2001]. Visant à adapter aux collecticiels la technique d'évaluation reposant sur des heuristiques de J. Nielsen [Nielsen 1990], K. Backer propose un ensemble de huit heuristiques issues de la théorie de la mécanique de la collaboration [Gutwin 2000]. Cette dernière identifie six actions de base qui régissent la mécanique de collaboration dans une situation d'espace partagé : communication, coordination, planification, consultation (*monitoring*), assistance et protection. Les huit heuristiques identifiées à partir de ces actions de base sont alors les suivantes

1. Un collecticiel doit fournir des moyens pour rendre observable l'échange verbal d'informations.
2. Un collecticiel doit fournir des moyens pour rendre observable l'échange par la gestuelle d'informations.

3 Un collecticiel doit fournir des moyens pour rendre observable l'échange d'informations induit par le comportement (position du corps, des mains, des yeux, expression du visage, etc.).

4 Un collecticiel doit fournir des moyens pour rendre observable l'échange d'informations induit par la manipulation d'artéfacts partagés (manipulation d'objets partagés et retour d'information sur les changements d'états de ces objets).

5 Un collecticiel doit fournir des moyens pour protéger l'espace de travail, l'espace privé, les objets partagés, etc.

6 Un collecticiel doit permettre plusieurs modes d'interaction (fortement couplé ou faiblement couplé) et la possibilité de modifier dynamiquement le niveau de couplage.

7 Un collecticiel doit mettre en œuvre des moyens pour autoriser les participants à coordonner leurs actions.

8 Un collecticiel doit permettre aux utilisateurs de découvrir qui sont tous les participants et doit faciliter le contact entre ces participants.

Les heuristiques sont employées pour diagnostiquer des problèmes potentiels d'utilisabilité liés à une interface. La méthode est très simple et consiste à faire inspecter l'interface par des évaluateurs selon les heuristiques. L'évaluateur commente l'interface pour chaque heuristique. Couramment, cette inspection nécessite entre trois et cinq évaluateurs maximum qui ont la capacité de détecter entre 75 et 80% des problèmes d'utilisabilité [Gutwin 2000]. Cette approche a l'avantage d'offrir une méthode d'évaluation à faible coût qui est très populaire dans les milieux industriels. Celle-ci peut être appliquée par des évaluateurs qui ne sont pas experts, car les heuristiques sont souvent bien documentées.

3.7. CONCLUSION

Dans cette partie du chapitre, nous avons présenté un ensemble de méthodes, modèles et outils pour la conception et la réalisation d'un collecticiel. Pour les présenter et bien cerner leurs portées, notre contribution a été de les situer par rapport aux étapes du cycle de vie du logiciel en V. Néanmoins cette partie ne constitue pas un état de l'art des outils existants. Notre objectif était ici de présenter un panorama des outils, organisé dans un canevas fédérateur, afin de mieux situer notre étude dont nous rappelons les objectifs dans la partie suivante.

4. Conclusion

4.1. MISE EN ŒUVRE DES COLLECTICIELS : OBJECTIFS DE L'ÉTUDE

Dans ce chapitre, nous avons d'abord souligné l'ampleur de l'espace des possibilités pour les collecticiels. Nous avons ensuite rappelé, par la présentation d'outils et de méthodes de mise en œuvre des collecticiels que le processus est organisé selon deux principales classes d'étapes de conception : les étapes de conception IHM (espace IHM) et les étapes de conception logicielle (espace Logiciel).

Les sciences humaines nourrissent essentiellement l'espace IHM grâce à l'apport d'outils et de méthodes issus de l'ethnologie ou de la sociologie (GTA), de modèles de l'utilisateur de psychologie cognitive et de modèles pour l'analyse fonctionnelle (modèle du trèfle). L'objectif affiché de ces outils, méthodes et modèles est d'aider à obtenir une spécification du collecticiel répondant aux besoins, identifiés au cours de l'analyse des besoins, en termes de spécifications fonctionnelles et de spécifications externes vérifiant des propriétés ergonomiques.

Les étapes suivantes consistent à exprimer en termes logiciels les spécifications : ce sont les étapes de l'espace Logiciel. C'est à l'aide des outils de conception logicielle, tels que les modèles d'architecture, et des outils de développement, tels que les boîtes à outils, que le développeur conçoit et code le logiciel du collecticiel.

Néanmoins, la transition entre l'espace IHM et l'espace Logiciel est complexe et délicate car il s'agit de traduire des spécifications sous forme de structures de données et d'algorithmes exécutés par des processus. Avec les étapes d'analyse des besoins et de conception, nous quittons l'espace de conception proprement dite de l'IHM pour pénétrer dans l'espace Logiciel réservé aux informaticiens. Le passage d'un espace à l'autre n'est pas sans difficulté [Nigay 1997] : le concepteur et le développeur ont l'obligation de satisfaire les spécifications fonctionnelles et externes de l'IHM tout en tenant compte des contraintes techniques imposées par les outils de mise en œuvre. Existe-t-il ainsi des outils logiciels pour la conception et le développement logiciels de collecticiels adaptés pour accompagner cette transition et permettre, entre autres, de satisfaire les requis ergonomiques tout en facilitant la réalisation logicielle ?

Dans ce contexte, l'objectif de cette étude est :

- d'analyser les outils logiciels existants au regard de cette transition de l'espace IHM vers l'espace Logiciel,
- de définir des outils de conception et de réalisation logicielles qui soient complémentaires à ceux existants et adaptés pour accompagner cette transition.

Nous nous sommes donc intéressés aux modèles d'architecture logicielle ainsi qu'aux boîtes à outils et infrastructures pour le développement des collecticiels. Ainsi le Chapitre III dresse un état de l'art des modèles d'architecture logicielle pour collecticiels tandis que le Chapitre IV présente notre contribution, le modèle d'architecture Clover pour faciliter la transition entre l'espace IHM et l'espace Logiciel. Symétriquement, le Chapitre V propose un état de l'art des boîtes à outils et infrastructures et le Chapitre VI notre contribution, la plate-forme Clover. Le Chapitre VII illustre l'utilisation de la plate-forme Clover en détaillant la mise en œuvre du système CoVitesse, un logiciel de navigation collaborative sur le *WWW*.

4.2. DÉMARCHE DE TRAVAIL

La démarche de travail adoptée dans cette étude se place du point de vue de l'activité de groupe. La position ainsi retenue apporte une dimension novatrice et complémentaire par rapport aux travaux existants. Comme nous l'avons évoqué dans le paragraphe précédent, l'objectif de cette étude porte sur la transition entre l'espace IHM et l'espace Logiciel. Par conséquent, il convient de se positionner par rapport à la phase de spécification, c'est-à-dire par rapport aux paramètres des spécifications fonctionnelles et externes, en s'appuyant sur le modèle du trèfle et les propriétés ergonomiques relatives aux collecticiels (présentés au paragraphe 3.3). Aussi, nous proposons une grille d'analyse qui organise ces paramètres au sein d'un canevas intégrateur présenté à la Figure 13.

Cette grille est ensuite utilisée tout au long de ce manuscrit pour étudier les architectures conceptuelles et les outils de développement pour les collecticiels mais aussi pour situer nos contributions. Nous verrons qu'il y a plusieurs niveaux de compatibilité entre les éléments de la grille et les modèles d'architecture et outils de développement. Nous reprenons ici les trois niveaux de compatibilité identifiés dans [Nigay 2001]. Selon les modèles et outils, les éléments de la grille seront explicités par localisation dans un composant logiciel (faible compatibilité), seront liés à des motifs architecturaux (compatibilité moyenne) ou encore seront associés à des services généraux ou mécanismes architecturaux (forte compatibilité). Plusieurs travaux en IHM pour des systèmes mono-utilisateurs étudient cette compatibilité entre les concepts de l'IHM et la réalisation logicielle : outre l'approche de notre équipe IIHM [Nigay 2001] qui est dépendante du modèle d'architecture logicielle PAC-Amodeus [Nigay 2001], nous relevons les travaux récents de L. Bass et B. John [Bass 2001] qui visent à établir des couples, scénarios d'utilisabilité liés à des propriétés ergonomiques et mécanismes architecturaux, ces travaux étant indépendants d'un modèle d'architecture logicielle particulier.

La grille d'analyse, centrée sur l'activité de groupe, repose sur le modèle du trèfle et les propriétés ergonomiques, deux outils présentés au paragraphe 3.3 pour la conception ergonomique de collecticiels. Pour définir la structure de la grille, nous exploitons la notion de contexte supposé partagé des actions, sachant qu'une action peut être individuelle ou collective [Hoogstoel 1995].

Le contexte, selon L. Karsenty [Karsenty 1997], est l'ensemble des ressources disponibles et exploitées par un utilisateur pour interpréter et exécuter une action. Il existe deux types de contexte, le contexte individuel et le contexte partagé. Précisément, le contexte est **supposé** partagé puisqu'il représente une "*compréhension mutuelle des actions de chacun*" et l'articulation des actions "s*uppose un certain degré de contexte partagé (l'intersection entre les contextes individuels)*". Ce contexte est nécessairement "supposé" partagé puisque chacun dispose de sa propre interprétation du contexte partagé, ce qui peut induire dans certaines situations des incompréhensions.

Selon L. Karsenty [Karsenty 1997], le contexte joue un rôle important dans la mise en œuvre de l'activité de groupe. En effet, l'activité de groupe résulte d'une articulation d'actions dans un contexte donné et il apparaît que cette articulation repose essentiellement sur la communication et l'exploitation de données contextuelles.

Le contexte est constitué de six niveaux de ressources organisés en deux grandes familles :

- La première est liée à la connaissance et à la cognition : cette forme de contexte est relative à la culture, à la capacité de perception et de réflexion d'un individu, c'est-à-dire l'ensemble des données et des connaissances "internes" à un individu.

- La seconde est liée aux autres et à ce qui nous entoure : il s'agit de l'environnement externe (bruits, mouvements, etc), de la communication, de la structure organisationnelle du groupe et de la perception de l'activité en cours (i.e. activité mise en œuvre par les autres participants), c'est-à-dire l'ensemble des données et connaissances "externes" à un individu.

A partir de cette définition de l'activité de groupe qui résulte d'un ensemble d'actions dans un contexte donné [Karsenty 1997], nous définissons la structure de notre grille. En effet nous considérons quatre dimensions : les actions, l'observabilité des actions, les ressources du contexte supposé partagé et l'observabilité de ces ressources. Deux dimensions caractérisent les actions et leur contexte tandis que deux autres décrivent leurs observabilités respectives. Deux valeurs sont considérées sur chacune des quatre dimensions :

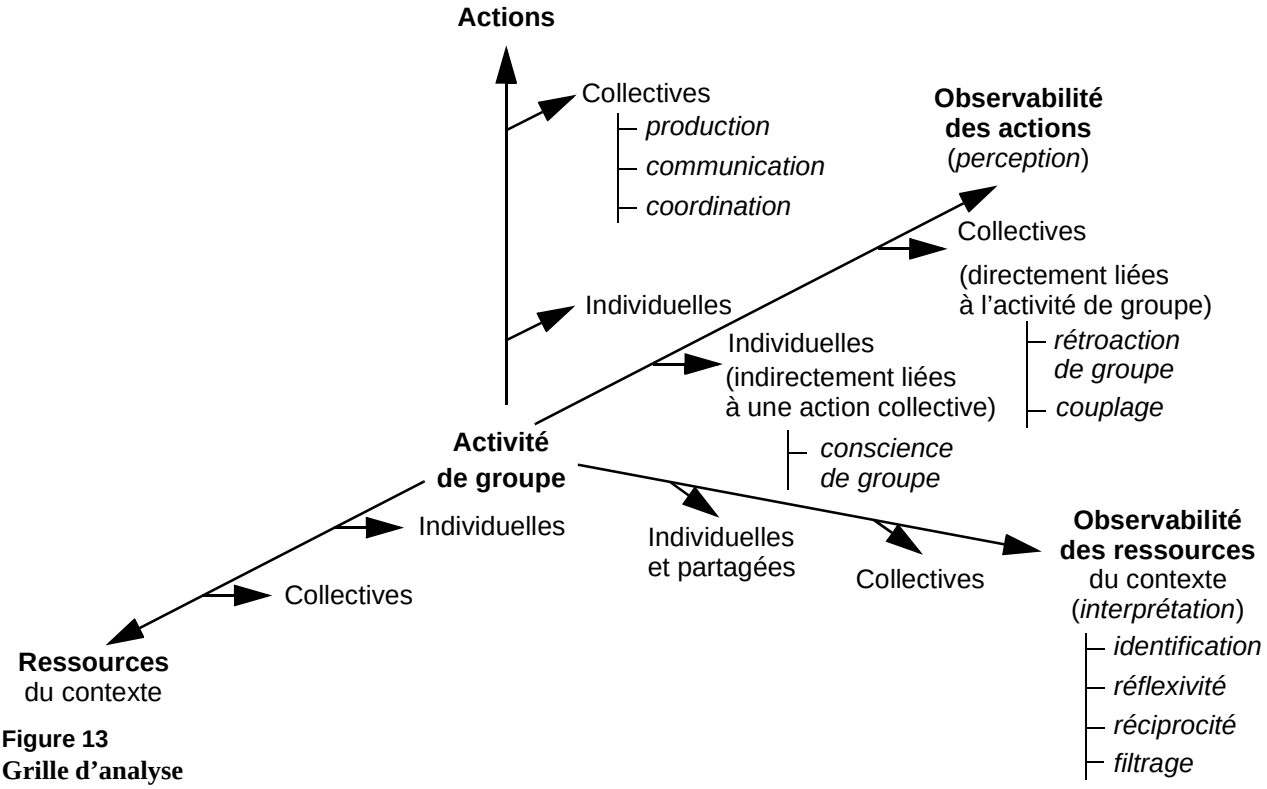


Figure 13
Grille d'analyse

- Individuelle ou propre à un utilisateur

- Collective ou commune à l'ensemble des utilisateurs

En détaillant les quatre dimensions, nous montrons que cette structure nous permet d'atteindre notre objectif annoncé : intégrer les aspects du modèle du trèfle (spécifications fonctionnelles) et les propriétés ergonomiques (spécifications externes).

- **Actions** : Deux valeurs sont considérées sur cette dimension intitulée Actions.

  * *Actions collectives* : un collecticiel doit fondamentalement mettre en œuvre et favoriser les actions collectives. Pour détailler les actions collectives, nous reprenons les trois classes du modèle du trèfle : actions de production, actions de communication et actions de coordination.

  * *Actions individuelle*s : les actions individuelles ne sont pas toujours possibles dans un collecticiel. Notons qu'un couplage fort de l'interaction (abordé dans la section intitulée "Spécifications externes et propriétés ergonomiques" du paragraphe 3.3) ne permet pas à un utilisateur d'effectuer une action individuelle. L'utilité d'autoriser des actions individuelles au sein d'un collecticiel dépend du domaine d'application. Néanmoins, de nombreuses études

montrent qu'une activité de groupe n'est efficace que si le collecticiel aménage des sessions ou des espaces dédiés aux actions individuelles : un travail de groupe est le résultat d'un entrelacement d'actions collectives et d'actions individuelles.

- **Observabilité des actions** : selon cette dimension, nous avons identifié deux cas d'observabilité correspondant aux deux types d'actions de la dimension Actions. Nous considérons l'observabilité des actions :
  * *Actions collectives* : une action collective qui participe directement à l'activité de groupe est caractérisée, par définition, par la propriété de rétroaction de groupe et par la propriété de couplage *WYSIWIS*.
  * *Actions individuelles* : une action individuelle qui participe indirectement à l'activité de groupe est caractérisée, par définition, par la propriété de conscience de groupe.

- **Ressources du contexte supposé partagé** : il existe par définition deux types de ressources constituant le contexte des actions : les ressources collectives et les ressources individuelles. Nous retrouvons ici la notion de contextes externe et interne décrite dans [Karsenty 1997].

- **Observabilité des ressources du contexte supposé partagé** : les ressources du contexte sont essentielles pour interpréter les actions. Cette interprétation repose donc sur les ressources disponibles, c'est-à-dire observables. Pour caractériser l'observabilité de ces ressources, nous nous appuyons sur les propriétés suivantes : identification, réflexivité, réciprocité et filtrage. Nous identifions deux types de ressources observables ;
  * *Ressources collectives* ; il s'agit des ressources propres à un groupe d'utilisateur ou communes à tous les utilisateurs : ces ressources collectives peuvent être rendues observables pour un groupe et ne pas l'être pour les autres.
  * *Ressources individuelles* : il s'agit de ressources propres à chaque utilisateur rendues observables en les partageant. Entre autres, le filtrage définit le niveau de partage des ressources individuelles. En effet, toutes les ressources individuelles ne sont pas systématiquement rendues observables et une partie peut rester entièrement privée.

Les dimensions présentées à la Figure 13 constituent un cadre conceptuel pour décrire l'activité de groupe en intégrant des éléments de conception ergonomique. Cependant, contrairement à ce que peut laisser supposer la Figure 13, les axes relatifs à l'observabilité ne sont pas totalement indépendants des axes relatifs aux actions et ressources. Par exemple,

l'observabilité d'une ressource individuelle ne peut être considérée que si cette ressource existe. Cette grille d'analyse sert de fondement à nos travaux dont les objectifs sont : concevoir et élaborer des outils pour la conception et la réalisation logicielles qui soient complémentaires aux outils existants et centrés sur l'activité de groupe.

Comme nous l'avons exposé, les phases de réalisation logicielle s'organisent en deux étapes : la conception logicielle selon un modèle d'architecture conceptuelle puis la réalisation logicielle grâce à un outil de développement. Le chapitre suivant aborde la première étape en présentant un état de l'art des architectures conceptuelles pour les collecticiels selon notre grille d'analyse.

Chapitre III *Modèles d'architecture logicielle pour les collecticiels*

1. Introduction

Le développement logiciel d'un collecticiel, comme nous l'avons expliqué dans le chapitre précédent, repose sur des spécifications définies au cours de l'étape de conception (espace IHM). L'étape charnière entre l'espace IHM et l'espace Logiciel est l'étape de conception logicielle, objet de ce chapitre. Lors de cette phase, l'architecture logicielle du collecticiel, en terme de composants, est construite. Cette étape constitue donc une première phase de traduction des spécifications de l'IHM en éléments logiciels, complétée ensuite par une seconde phase de codage à l'aide d'outils de développement. Cette étape de conception logicielle est soumise à une tension permanente entre d'une part la satisfaction des requis de l'utilisateur, des spécifications de l'IHM et d'autre part les contraintes techniques imposées par les outils de mise en œuvre et la satisfaction des requis de qualité logicielle.

Dans ce chapitre nous dressons un état de l'art des modèles d'architecture pour les collecticiels selon la démarche de travail retenue et définie dans le chapitre précédent. Aussi chaque modèle d'architecture est étudié selon notre grille d'analyse (Chapitre II), qui caractérise l'activité de groupe.

Le chapitre est organisé de la façon suivante : la première partie définit la notion d'architecture logicielle et de modèle d'architecture. Dans la seconde partie, nous présentons trois modèles d'architecture de référence pour les systèmes interactifs mono-utilisateurs à l'origine de nombreux modèles d'architecture pour les collecticiels : Arch, MVC et PAC-Amodeus. Dans la troisième partie, nous traduisons les éléments de notre grille d'analyse en des termes liés à l'architecture logicielle, afin d'étudier dans la quatrième partie les modèles existants. En conclusion, nous

présentons un bilan critique des modèles existants, bilan qui nous a conduit à définir le nouveau modèle Clover présenté au chapitre suivant.

2. *Architecture logicielle*

2.1. DÉFINITION

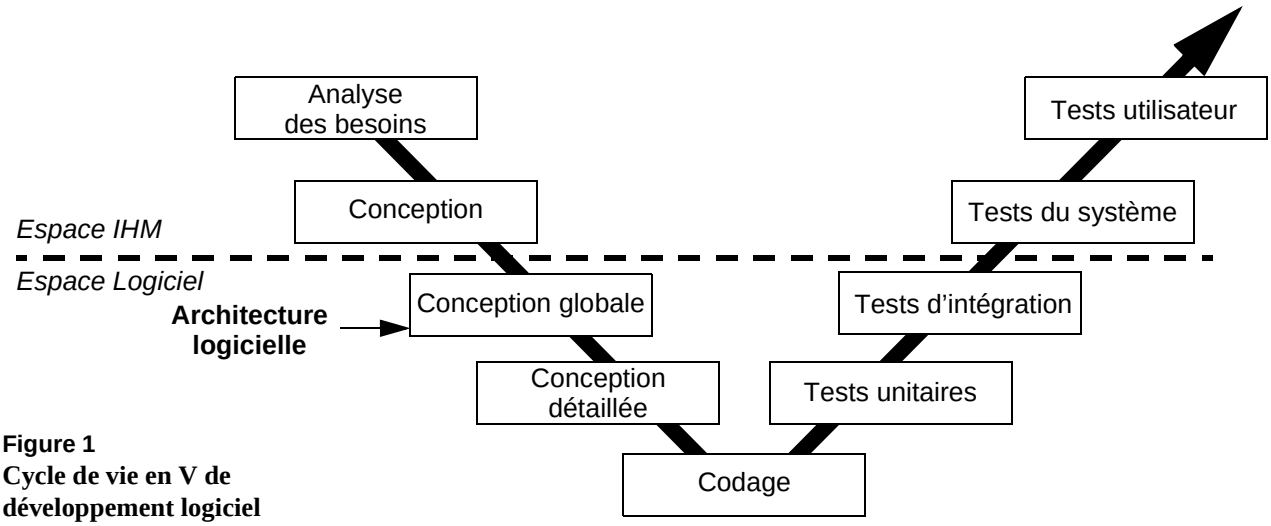


Figure 1
Cycle de vie en V de développement logiciel

L'architecture est l'art de la construction d'édifices selon un ensemble de règles établies [Robert 1992]. L'architecture d'un édifice, c'est aussi sa forme, sa structure. Dans le domaine de l'Interaction Homme-Machine, il s'agit de la façon dont les composants logiciels ou matériels sont organisés et assemblés pour concevoir un système interactif.

Un modèle d'architecture logicielle est utilisé lors de la phase de conception logicielle dans le cycle de vie du logiciel pour concevoir l'architecture du système. Dans un processus de développement, comme le montre la Figure 1 représentant les étapes d'un cycle en V, l'architecture d'un système interactif est le résultat de l'étape de conception globale. Cette étape est une phase critique puisqu'il s'agit d'une étape charnière entre l'espace de conception de l'Interface Homme-Machine (espace IHM) et l'espace de la réalisation logicielle (espace Logiciel). En effet, comme nous l'avons souligné dans le chapitre précédent, c'est au cours de la phase de spécification que l'ensemble des concepts et fonctionnalités est identifié. Cette phase de spécification résulte de l'élaboration de spécifications fonctionnelles et de spécifications externes, c'est-à-dire la spécification de l'interaction. Ainsi, la phase suivante, phase de transition entre les espaces IHM et Logiciel,

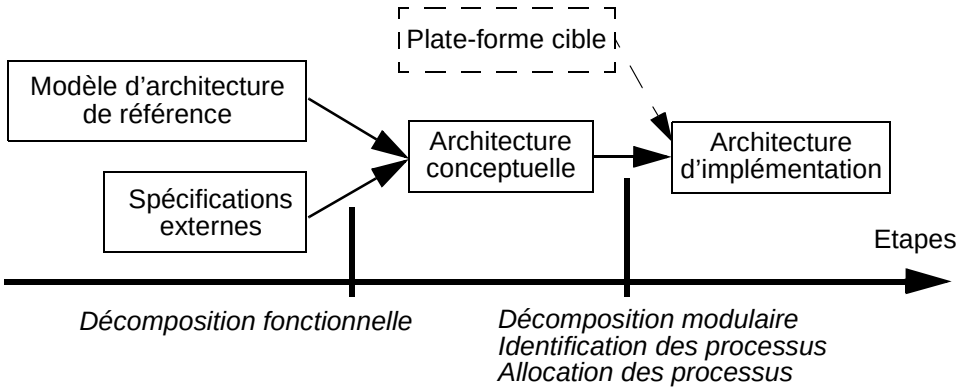


Figure 2
Etapes de production d'une architecture [Coutaz 2001]

consiste à traduire ces spécifications sous forme logicielle en appliquant un modèle d'architecture logicielle.

Une architecture logicielle est assimilée à un ensemble organisé de composants dont les interactions sont médiatisées par des entités spécialisées ou connecteurs. Le processus de conception d'une architecture logicielle, comme le montre la Figure 2, recouvre les deux principales activités suivantes [Bass 1998][Coutaz 2001] : définition de la décomposition fonctionnelle du système, choix d'une organisation structurelle (vue statique et dynamique). Le résultat de cette activité constitue la structure modulaire ou **architecture conceptuelle** :

- *Définition d'une* ***décomposition fonctionnelle*** : cette activité consiste à identifier les fonctionnalités du système regroupées sous forme d'entités logicielles ou de composants. Cette étape est complexe puisqu'il s'agit d'exprimer les requis utilisateurs et les spécifications externes de l'interface en termes logiciels.

- *Choix d'une* ***organisation structurelle*** : il s'agit de choisir ou de définir un schéma d'organisation des différents composants identifiés lors de la décomposition fonctionnelle, ainsi qu'un protocole d'échanges entre ces composants. Ce protocole est défini par l'ensemble des connecteurs et des règles définissant l'interaction entre les composants. L'organisation obtenue reflète une vue statique (agencement des composants) et dynamique (protocole d'échanges et migration fonctionnelle) du système. Les concepteurs peuvent s'inspirer de **modèles d'architecture de référence**. Ces modèles offrent une description standard d'une organisation de composants et de règles régissant les relations entre ces composants. En général, un modèle d'architecture permet de répondre à une classe de problèmes. Par exemple, le modèle d'architecture du compilateur permet l'identification de ses principaux composants : l'analyseur lexical, l'analyseur syntaxique, l'analyseur sémantique et le générateur de code.

La structure modulaire du logiciel étant établie, il convient de focaliser sur l'association entre entités structurelles et processus d'exécution du système, voire l'allocation des processus aux processeurs. Cette dernière activité prend tout son sens pour un collecticiel, système par définition réparti. La structure physique est alors conçue : elle complète la structure modulaire de l'étape précédente pour constituer ensemble l'**architecture implémentationnelle**. Cette architecture décrit l'organisation des fonctionnalités selon un ensemble de modules qui représentent les unités logicielles qui seront développées. De plus, cette architecture traduit l'aspect dynamique du logiciel en détaillant la répartition du logiciel selon les différents processus. De plus, comme le montre la Figure 2, le développement d'un collecticiel semble de plus en plus dépendant du choix de la plate-forme cible car les choix techniques ont un impact direct sur son développement (ressources disponibles en matière de réseau ou de capacités calculatoires, etc).

2.2. PROPRIÉTÉS

Tout projet de réalisation d'un système interactif est guidé par la nécessité de produire un système de qualité mesurable. Selon l'approche proposée dans [IFIP 1996], la qualité d'un système interactif se mesure en terme de propriétés externes et internes : les propriétés externes sont liées à l'utilisabilité du système, le point de vue est donc celui de l'utilisateur final du système. Les propriétés internes traduisent la qualité du logiciel. [McCall 1977] a identifié onze propriétés internes comme la modifiabilité, l'extensibilité ou la portabilité. Par exemple, la propriété d'extensibilité traduit la capacité d'un système à facilement intégrer de nouvelles fonctionnalités avec un coût de développement réduit. Pour les systèmes interactifs, certaines propriétés du logiciel comme la modifiabilité, sont très importantes. En effet, la réalisation d'un système interactif nécessite une conception itérative de l'interface centrée sur l'utilisateur : chaque itération du cycle permet de corriger des problèmes d'utilisabilité et ainsi d'améliorer la qualité de l'interface. La modifiabilité du code de l'interface est donc essentielle pour faciliter ces itérations avec un coût de développement le plus faible possible.

Une architecture n'est jamais intrinsèquement bonne ou mauvaise, mais sa qualité se juge à la lumière de propriétés identifiées par avance dans le cadre d'un plan d'assurance qualité. Au-delà des propriétés internes usuelles du Génie Logiciel (modifiabilité, extensibilité, portabilité, etc.), les collecticiels sont concernés par des propriétés externes centrées sur l'utilisateur et l'activité de groupe dont le respect peut avoir un impact sur les solutions architecturales. Au cours de ce chapitre, nous étudions les modèles d'architecture logicielle à la lumière de notre grille d'analyse. Ainsi, nous utilisons des propriétés externes incluses dans notre grille comme élément d'évaluation des architectures spécifiques aux collecticiels. On trouvera dans [IFIP 1996] une discussion systématique

des relations entre modèles d'architecture et propriétés externes pour la mise en œuvre de systèmes mono-utilisateur. Notre approche étend ces résultats [IFIP 1996] pour la mise en œuvre de collecticiels.

Ayant défini ce qu'est un modèle d'architecture et son rôle, la partie suivante présente trois modèles d'architecture pour la conception de systèmes interactifs auxquels nous faisons référence lors de la présentation des modèles d'architecture dédiés aux collecticiels.

3. Modèles d'architecture pour les systèmes interactifs

Cette partie présente trois modèles d'architecture pour les systèmes interactifs : *Arch*, *MCV* et *PAC-Amodeus*. Ces modèles sont à l'origine de nombreux modèles d'architecture pour les collecticiels présentés dans la partie 5.

Les trois modèles retenus illustrent respectivement trois styles d'architecture, c'est-à-dire une façon d'organiser et de regrouper les fonctionnalités en modules : modèle monolithique, modèle multi-agent, et modèle hybride combinant les deux premiers styles. Ces trois modèles préconisent une décomposition fonctionnelle qui véhicule une séparation du code du noyau fonctionnel (logique de l'application) de celui de l'interface. Ces modèles véhiculent de nombreuses propriétés dont la propriété de modifiabilité qui est, comme nous l'avons expliqué, cruciale en Interaction Homme-Machine.

Le premier paragraphe présente le modèle Arch (modèle monolithique), le second le modèle MVC (modèle multi-agent) et enfin le dernier est dédié au modèle hybride PAC-amodeus.

3.1. MODÈLE ARCH

Le modèle Arch [Bass 1992], une extension du modèle séminal Seeheim [Green 1985], est un modèle d'architecture qui offre une décomposition canonique des principaux composants d'un système interactif. Le principe véhiculé par le modèle, comme le montre la Figure 3, est de séparer l'interface utilisateur du Noyau Fonctionnel (logique de l'application). En pratique, le Noyau Fonctionnel ne doit avoir aucune connaissance des fonctionnalités relevant de l'interface utilisateur pour faciliter une conception itérative de l'interface, pour favoriser la réutilisation et pour favoriser la portabilité du logiciel. Les composants de ce modèle d'architecture sont :

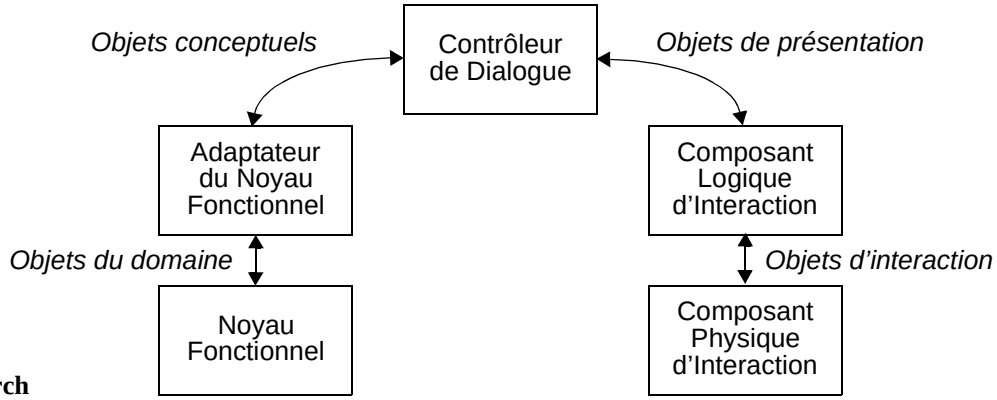


Figure 3
Modèle de référence Arch

- **Le Noyau Fonctionnel** (NF) implémente les fonctionnalités et les concepts du domaine indépendamment de leur présentation. Les structures de données manipulées par ce composant sont les objets du domaine.

- **L'Adaptateur du Noyau Fonctionnel** (ANF) joue un rôle de médiateur entre le Noyau Fonctionnel et le Contrôleur de Dialogue. Les données échangées avec le Noyau Fonctionnel sont les objets du domaine que le NF exporte vers l'utilisateur. Les données échangées avec le Contrôleur de Dialogue sont des objets conceptuels correspondant à une représentation mentale de l'utilisateur des objets du domaine.

- **Le Composant Physique d'Interaction** (CP) représente les interacteurs logiciels (*widget*) et matériels. Il s'agit en général d'une boîte à outils graphique (*User Interface Toolkit*) et des périphériques d'interaction.

- **Le Composant Logique d'Interaction** (CL) joue aussi un rôle de médiateur entre le Contrôleur de Dialogue et le Composant Physique d'Interaction. Ce composant, généralement assimilé à une boîte à outils graphique abstraite, permet l'indépendance vis-à-vis des boîtes à outils graphiques du niveau du Composant Physique. Par exemple, la boîte à outils graphique *AWT* (*Abstract Window Toolkit*) écrite en Java relève de ce niveau : elle est indépendante de la boîte à outils graphique sous-jacente et un même programme peut utiliser insensiblement la boîte à outils graphique X/Motif sous Unix ou la boîte à outils graphique MacToolbox sous MacOS.

- **Le Contrôleur de Dialogue** (CD) est la pierre angulaire de cette architecture puisque ce composant a la charge de gérer le dialogue, c'est-à-dire l'enchaînement des tâches. Ce composant manipule à la fois les objets conceptuels et les objets de présentation nécessaires à l'interaction. Le Contrôleur de Dialogue associe un ou plusieurs objets de présentation avec un ou plusieurs objets conceptuels correspondants et réciproquement.

Ce modèle fournit une décomposition fonctionnelle canonique à gros grain. En effet, le modèle Arch structure un système interactif selon cinq niveaux d'abstraction, distinguant ceux qui relèvent du domaine de l'application de ceux qui gèrent l'interface utilisateur. Néanmoins, ce modèle n'apporte aucune précision sur la structure de ses composants comme celle du Noyau fonctionnel et du Contrôleur du Dialogue.

D'autres modèles d'architecture, dits modèles multi-agents, préconisent une décomposition fonctionnelle plus fine, reposant sur des composants de taille significativement plus petite que ceux d'Arch. Dans le paragraphe suivant, nous présentons un tel modèle, le modèle multi-agent MVC.

3.2. MODÈLE MVC

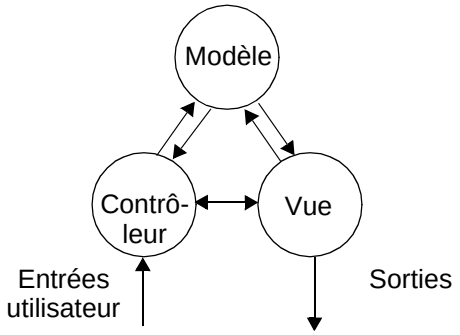


Figure 4
Agent MVC

Le modèle MVC a été introduit dans le langage Smalltalk [Krasner 1988]. Il s'agit d'un modèle multi-agent qui s'inscrit dans la même ligne de pensée que le modèle Arch, puisqu'il distingue la partie interface utilisateur du modèle de l'application, c'est-à-dire le Noyau Fonctionnel.

Comme le montre la Figure 4, un agent MVC est constitué de trois facettes :

- **Une facette modèle** (M) qui représente les concepts du domaine. Par analogie, cette facette correspond à l'agrégation du Noyau Fonctionnel, à l'Adaptateur du Noyau Fonctionnel et au Contrôleur de Dialogue du modèle Arch.

- **Une facette contrôleur** (C) qui interprète, au niveau de l'interface utilisateur, les entrées. Par comparaison avec le modèle Arch, il s'agit de la partie de gestion des entrées des Composants Logique et Physique d'interaction du modèle Arch.

- **Une facette vue** (V) qui offre une représentation en sortie au niveau de l'interface utilisateur (affichage, son, haptique, etc). Cette facette est le complémentaire de la facette contrôleur, puisque celle-ci gère les

sorties vers l'utilisateur et correspond, par analogie, à la gestion en sortie des Composants Logique et Physique du modèle Arch.

La vue et le contrôleur communiquent avec le modèle à l'aide d'événements et de traitants d'événements (*callback*). Par contre la communication entre la vue et le contrôleur est directe et est réalisée par le biais d'appels de fonction. Par exemple, une barre de défilement pourrait être implémentée sous la forme d'un agent dont le modèle gère la position du curseur ; dès que le curseur est déplacé dans le contrôle, le modèle reçoit un événement de modification pour mettre à jour la position du curseur et la vue reçoit l'ordre d'afficher le curseur à sa nouvelle position. Pour des raisons de performance, la vue et le contrôleur peuvent êtres fusionnés et développés sous la forme d'un seul composant. Par exemple, la boîte à outils graphique Swing de Java [Java 2002] repose sur une variante du modèle MVC : les parties vue et contrôleur des objets graphiques (boutons, menus, etc) sont fusionnées sous la forme d'une seule classe Java.

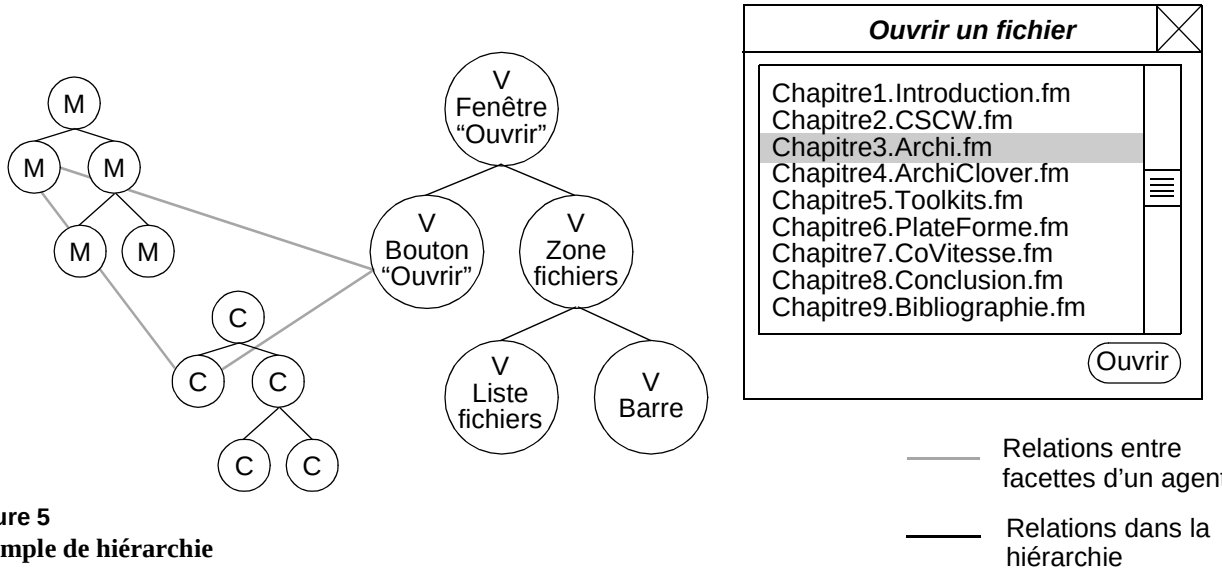


Figure 5
Exemple de hiérarchie

L'architecture d'un système interactif conçue à l'aide de ce modèle multi-agent résulte d'une combinaison d'agents MVC organisés, comme le montre la Figure 5, en une hiérarchie. Cette hiérarchie traduit l'organisation spatiale de l'interface graphique. En effet, une interface peut être vue comme une hiérarchie de composants. Prenons l'exemple de l'interface de la Figure 5 : il s'agit d'une boîte de dialogue standard pour ouvrir un fichier. Cette boîte est une fenêtre dont les fils sont une liste de fichiers et un bouton pour fermer la fenêtre. La zone listant les fichiers dans le répertoire courant est elle-même constituée de plusieurs fils : une barre de défilement et une zone de texte contenant les noms des fichiers. La hiérarchie résultante est constituée d'un agent racine, représentant la fenêtre, constitué de deux agents, un pour la zone de fichiers, l'autre pour

le bouton. L'agent de la zone de fichiers est lui-même constitué de deux fils, un pour la barre de défilement et un pour l'affichage des noms de fichiers.

Une extension de ce modèle d'architecture pour les collecticiels met en œuvre des agents MVC dont la facette modèle est partagée par plusieurs instances des deux facettes vue et contrôleur. Par exemple, le modèle Clock, présenté au paragraphe 5.4, est un modèle d'architecture pour les collecticiels qui repose sur le modèle MVC.

Comparé aux cinq niveaux fonctionnels du modèle Arch, ce modèle fournit une décomposition fonctionnelle plus grossière puisque la partie modèle d'un agent englobe à la fois le Noyau Fonctionnel, l'Adaptateur du Noyau Fonctionnel et le Contrôleur de Dialogue. Aussi on pourrait envisager un modèle multi-agent où chaque agent serait décomposé en cinq facettes correspondant aux cinq niveaux fonctionnels d'Arch.

Une autre façon de combiner la décomposition fonctionnelle d'Arch avec un modèle multi-agent, consiste à peupler certains d'agents logiciels composants d'Arch. Ceci fait l'objet du paragraphe suivant, qui présente le modèle hybride PAC-Amodeus : ce dernier combine la décomposition d'Arch avec celle du modèle multi-agent PAC.

3.3. MODÈLE PAC-AMODEUS

Le modèle PAC-Amodeus [Nigay 1994] est un modèle hybride puisqu'il repose sur une extension du modèle Arch selon une approche multi-agent. Ce modèle reprend les cinq niveaux fonctionnels du modèle Arch et structure le Contrôleur de Dialogue avec une hiérarchie d'agents PAC.

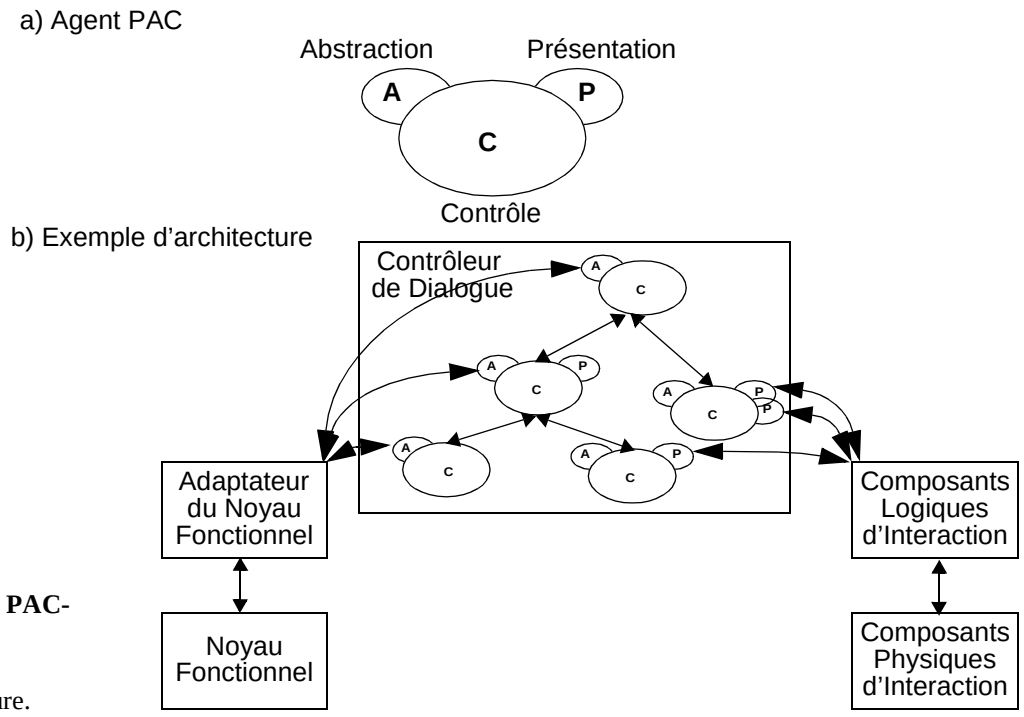


Figure 6
Modèle d'architecture PAC-Amodeus
a) Agent PAC,
b) Exemple d'architecture.

Un agent PAC, comme le montre la Figure 6 (a), est composé de trois facettes :

- **Une facette Abstraction** (A) qui gère les concepts du domaine et définit la compétence de l'agent indépendamment de la présentation,

- **Une facette Présentation** (P) qui définit l'interface utilisateur et interprète les entrées et sorties générées par l'utilisateur au cours de l'interaction,

- **Une facette Contrôle** (C) qui, d'une part, fait le lien entre les facettes Abstraction et Présentation et qui, d'autre part, assure et gère les relations avec les autres agents dans la hiérarchie (l'agent père et les agents fils). Les agents communiquent entre eux uniquement à travers cette facette Contrôle.

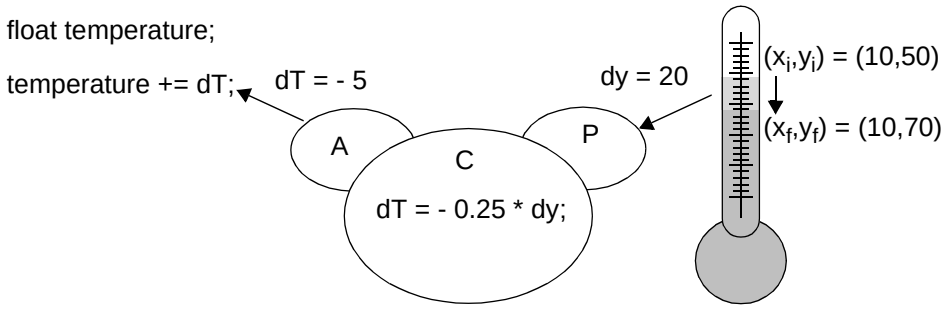


Figure 7
Exemple d'agent PAC

Pour illustrer le rôle de chaque facette d'un agent PAC, prenons le célèbre exemple du thermomètre présenté à la Figure 7. La facette Présentation a la charge de dessiner le thermomètre et d'interpréter les événements utilisateurs en entrée qui modifient la valeur de la température. La facette Abstraction constitue le modèle abstrait du thermomètre représentant la valeur courante en degrés par un nombre réel. La facette Contrôle réalise un lien entre la facette Abstraction et la facette Présentation et se charge de mettre en correspondance les phénomènes concrets avec les phénomènes abstraits. Ainsi, lorsque l'utilisateur modifie par manipulation directe la température, la facette Contrôle informe la facette Abstraction du changement et se charge de transmettre la nouvelle valeur après avoir effectué la correspondance entre la valeur liée au formalisme graphique et celle de sa représentation abstraite.

La propriété de modifiabilité est fortement favorisée par ce modèle d'architecture car il permet l'ajout et le remplacement de facettes à moindre coût grâce à sa structure très modulaire. Ainsi, en reprenant l'exemple du thermomètre, nous pouvons imaginer que la facette Présentation dessinant un thermomètre peut être facilement remplacée par une autre facette Présentation qui afficherait la valeur du thermomètre uniquement sous forme textuelle et permettait de saisir au clavier une nouvelle valeur après sélection de la zone de texte.

Si l'on compare le modèle multi-agent PAC [Coutaz 1984] au modèle Arch, la facette Abstraction correspond au Noyau Fonctionnel, la facette Contrôle englobe l'Adaptateur de Noyau Fonctionnel et le Contrôleur de Dialogue et la facette Présentation correspond aux Composants Logique et Physique d'Interaction. Comparé au modèle MVC, les facettes Abstraction et Contrôle correspondent au modèle (M) et la facette Présentation englobe la vue (V) et le contrôle (C).

Au sein du modèle hybride PAC-Amodeus, les agents PAC, comme le montre la Figure 6 (b), sont organisés selon une hiérarchie qui définit le Contrôleur de Dialogue. A l'opposé du modèle multi-agent PAC où la hiérarchie d'agents constitue l'ensemble du système interactif, dans le modèle PAC-Amodeus, les agents peuplent un seul composant, le Contrôleur de Dialogue. La facette Abstraction de chaque agent communique alors avec le Noyau Fonctionnel via l'Adaptateur du Noyau Fonctionnel et manipule des objets conceptuels. De même, la facette Présentation de chaque agent communique avec le Composant Physique d'Interaction via le Composant Logique d'Interaction. Dans PAC-Amodeus, la structuration en trois facettes d'un agent PAC n'est pas obligatoire. Ainsi un agent peut ne pas avoir de facette Présentation. Un agent peut aussi implémenter plusieurs facettes Présentation dans le cas de vues multiples. Un agent situé à un noeud de la hiérarchie n'implémentant que la facette Contrôle et/ou la facette Abstraction est un agent ciment : cet agent résulte d'une factorisation de compétences pour offrir un niveau supplémentaire d'abstraction et accroître la modularité du code. L'organisation de ces agents est régie par un ensemble de règles heuristiques décrites par L.Nigay dans [Nigay 1994].

Il existe aussi une extension de ce modèle pour les collecticiels : le modèle PAC* étudié dans la paragraphe 5.7.

3.4. SYNTHÈSE

Dans cette partie, nous avons présenté trois modèles d'architecture, de styles différents : monolithique, multi-agent et hybride. Le dernier modèle exposé, le modèle PAC-Amodeus, est un modèle hybride qui combine le modèle Arch avec une approche multi-agent. Cette combinaison se traduit par un Contrôleur de Dialogue peuplé par une hiérarchie d'agents PAC. Il est bien sûr tout à fait concevable d'utiliser des agents MVC pour peupler ce Contrôleur de Dialogue.

Lors de l'étude des modèles d'architecture logicielle pour collecticiels dans la partie 5, nous retrouvons les trois styles de modèles. Avant cette étude, il convient de traduire notre grille d'analyse du Chapitre II en termes d'architecture logicielle, pour ensuite pouvoir analyser et comparer de façon systématique les modèles d'architecture à la lumière de notre grille, dans la partie 5.

4. Grille d'analyse

Pour étudier les différents modèles d'architecture pour les collecticiels selon la démarche adoptée, il convient de traduire les éléments de notre grille d'analyse en termes d'architecture logicielle conceptuelle. Dans les paragraphes suivants, nous traitons successivement les quatre dimensions de la grille : actions au paragraphe 4.1, ressources du contexte des actions au paragraphe 4.2, et enfin observabilité des actions et des ressources du contexte au paragraphe 4.3.

4.1. ACTIONS INDIVIDUELLES ET COLLECTIVES

Au niveau architectural, les actions collectives et individuelles correspondent à des fonctions localisées dans des composants logiciels. Dans la grille d'analyse, les actions collectives sont organisées selon les trois espaces du modèle du trèfle : production, communication et coordination. Les fonctions du collecticiel doivent donc pouvoir se classer selon ces trois espaces.

Nous définissons trois niveaux de compatibilité entre cette dimension de la grille et un modèle d'architecture :

- *Compatibilité faible* : le modèle d'architecture préconise un composant logiciel qui contient les fonctions correspondant aux actions collectives et individuelles.

- *Compatibilité moyenne* : le modèle d'architecture préconise deux composants logiciels distincts, l'un contenant les fonctions correspondant aux actions individuelles, l'autre les fonctions liées aux actions collectives.

- *Compatibilité forte* : le modèle d'architecture préconise les deux composants du niveau de compatibilité précédent. De plus le modèle affine le composant dédié aux actions collectives en trois composants, selon les trois types de fonction suivants : production, communication et coordination.

4.2. RESSOURCES DU CONTEXTE

Selon la dimension intitulée "ressources du contexte", nous distinguons les ressources individuelles et les ressources collectives. Au sein d'une architecture logicielle, les ressources se traduisent en données maintenues par un composant logiciel.

Ainsi, nous considérons deux types de composants, les composants privés gérant les ressources individuelles et les composants publics gérant les ressources collectives. Il convient néanmoins de noter, et nous revenons sur ce point dans le dernier paragraphe de ce chapitre, que des ressources

individuelles peuvent être consultées par les autres utilisateurs en rendant accessibles des données gérées par un composant privé.

Un collecticiel doit nécessairement mettre en œuvre au moins un composant public : c'est le point d'ancrage de la collaboration entre les utilisateurs. En effet, en l'absence de composant public, il n'y a plus de collaboration possible puisque tous les composants sont alors privés et le contexte partagé est inexistant. L'étude menée dans la partie suivante confirme ce point.

Dans le reste du mémoire, nous employons systématiquement les termes de composants publics et privés, sauf dans le cas où l'auteur d'un modèle d'architecture utilise explicitement une autre terminologie. De plus, nous désignons par espace public l'ensemble des composants publics et par espace privé l'ensemble des composants privés "appartenant" à un seul utilisateur.

4.3. OBSERVABILITÉ DES ACTIONS ET DES RESSOURCES DU CONTEXTE

Selon la dimension "observabilité des actions", nous identifions deux formes d'observabilité : observabilité des actions individuelles (indirectement liées à l'activité du groupe) et celle des actions collectives (directement liées à l'activité du groupe). La première forme d'observabilité participe à la propriété de conscience de groupe, la deuxième aux propriétés de rétroaction de groupe et de couplage de l'interaction. L'observabilité des actions se traduit en terme architectural par un échange de données entre composants. Le composant source est celui qui contient la fonction associée à l'action et qui maintient les données traduisant cette action (état initial, état final et états intermédiaires de l'action). Le composant destinataire est le composant chargé de la présentation (interface en sortie), pour chaque utilisateur du collecticiel.

De même l' "observabilité des ressources du contexte" se traduit par un échange de données. S'il s'agit de ressources individuelles, alors le composant source est le composant de l'espace privé qui maintient ces ressources. Au contraire s'il s'agit de ressources collectives, alors le composant source appartient à l'espace public. Dans les deux cas, le composant destinataire est là aussi le composant responsable de la présentation (interface en sortie), pour chaque utilisateur du collecticiel.

Cependant, bien que l'observabilité des ressources du contexte et l'observabilité des actions présentent des similarités, il convient de marquer la différence qui existe entre ces deux notions. La réalisation d'une action s'opère à destination d'un composant et à partir de données issues des ressources. En ce sens, une action collective agit sur des ressources collectives et une action individuelle agit sur des ressources individuelles. Le résultat de cette action se traduit par une succession de changements d'état d'un ensemble de données, c'est-à-dire des

ressources. L'observabilité des actions et des ressources du contexte présentent donc des similarités puisque cela revient à rendre observables des ressources. Néanmoins, l'observabilité des actions ne peut pas être totalement réduite à une observabilité de ressources : l'observabilité d'une action consiste à rendre observable toute une chaîne de changements d'états ; l'observabilité des ressources est la consultation de données "figées".

4.4. SYNTHÈSE

En synthèse, nous listons l'ensemble des termes architecturaux liés à notre grille d'analyse :

1 **Actions** :

* *Action individuelle* : composant logiciel dédié aux fonctions mono-utilisateurs.
* *Action collective* : composant logiciel dédié aux fonctions multi-utilisateurs.
  - production : composant logiciel dédié à la production.
  - communication : composant logiciel dédié à la communication.
  - coordination : composant logiciel dédié à la coordination.

2 **Ressources du contexte** :

* *Ressources collectives* : composant logiciel public (ou partagé).
  L'espace public est composé de l'ensemble des composants publics.
* *Ressources individuelles* : composant logiciel privé.
  L'espace privé désigne l'ensemble des composants privés.

3 **Observabilité des actions et des ressources** :

* Echange de données entre composants logiciels.

Ayant traduit en termes architecturaux notre grille d'analyse, nous analysons dans la partie suivante des modèles d'architecture pour les collecticiels.

5. Modèles d'architecture pour les collecticiels

Dans cette partie, nous étudions sept modèles d'architecture pour les collecticiels. L'objectif de cette étude est d'analyser comment ces différents modèles traitent de la notion d'activité de groupe comme définie dans notre grille d'analyse. Les sept modèles étudiés, organisés selon leur style d'architecture (monolithique, multi-agent, hybride), sont : le modèle *Zipper* et le méta-modèle d'architecture de Dewan ; les trois modèles multi-agents ALV, Clock et AMF-C; puis les deux modèles hybrides CoPAC et PAC*.

Pour décrire chaque modèle, nous employons la terminologie adoptée par l'auteur que nous relions à la notre. Par contre, l'analyse de chaque modèle repose sur notre terminologie.

5.1. MODÈLE *ZIPPER*

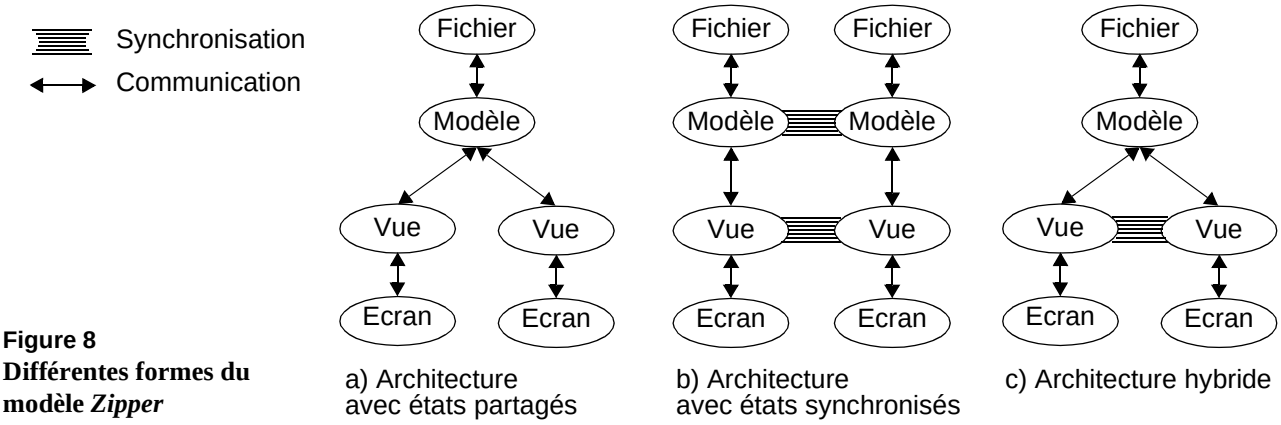


Figure 8
Différentes formes du modèle *Zipper*

Description

Le modèle *Zipper* (fermeture éclair) de Patterson [Patterson 1994] repose sur la notion d'états partagés. Ainsi, un collecticiel est décomposé, comme le montre la Figure 8, selon quatre niveaux d'états qui définissent des niveaux d'abstraction :

- **Etat de l'écran** (*Display*) qui correspond à l'état des périphériques d'entrée et de sortie (moniteur, souris, etc.),
- **Etat de la vue** (*View*) qui correspond à l'état de la présentation logique des données, c'est-à-dire l'état de l'interface utilisateur,
- **Etat du modèle** (*Model*) qui correspond au noyau fonctionnel et aux objets du domaine,
- **Etat du fichier** (*File*) qui correspond à la représentation persistante du modèle.

Ces états peuvent être instanciés selon trois modes : partagé, synchronisé ou répliqué. Selon notre terminologie, un état privé est similaire à un composant public, un état synchronisé est similaire à un composant partagé et copié à l'identique au niveau de tous les clients, et un état répliqué est similaire à un composant privé.

Dans le mode partagé, un état est instancié sous forme unique. Dans le mode répliqué, il existe autant d'instances d'un état qu'il y a d'utilisateurs. Le mode synchronisé est similaire au mode répliqué, mais un mécanisme de synchronisation est chargé de maintenir la cohérence entre toutes les copies (états identiques). La synchronisation d'états répliqués a été introduite pour rendre les collecticiels plus performants puisque les états synchronisés contiennent exactement la même information mais avec une mise à jour plus ou moins retardée en fonction du degré de couplage. Une architecture hybride est constituée à la fois d'états partagés, répliqués et synchronisés. L'exemple de la Figure 8 (a) représente une architecture constituée de deux états partagés (le fichier et le modèle) et de deux états répliqués. L'exemple de la Figure 8 (b) représente une architecture répartie avec synchronisation de deux états (le modèle et la vue). Enfin, l'exemple de la Figure 8 (c) représente une architecture hybride, une combinaison des deux exemples précédents, constituée de deux états partagés (le fichier et le modèle) et de deux états répliqués avec une vue synchronisée. La métaphore de la fermeture éclair qui a inspiré le nom du modèle, *Zipper*, fait écho au niveau de fermeture de l'architecture par des états partagés. Dans l'exemple de la Figure 8 (a), la fermeture est réalisée au niveau de l'état modèle. Au contraire, dans l'exemple de la Figure 8 (b), l'architecture est totalement ouverte.

Analyse

Comparé au modèle Arch, le modèle *Zipper* offre une décomposition fonctionnelle à plus gros grain ne considérant pas le niveau Contrôleur de Dialogue. Le modèle est assez flou sur ce point, puisque la gestion du dialogue est à la fois traitée par le modèle, la vue et la synchronisation potentielle des différentes vues.

Le modèle ne fait aucune distinction entre les fonctions mono-utilisateur et multi-utilisateurs. Ceci est dû au fait que le modèle raisonne principalement en termes d'états et non en termes de fonctions. A fortiori, le modèle ne fait aucune distinction entre les fonctions relevant de la production, de la communication ou de la coordination.

Le modèle distingue les états partagés des états répliqués : un état partagé est commun à tous les utilisateurs (espace public) et un état répliqué est propre à un seul utilisateur (espace privé). Cependant, le modèle oscille entre aspect conceptuel et aspect implémentationnel. En effet, selon l'auteur, un état partagé est nécessairement alloué à un processus centralisé et un état répliqué alloué à un processus réparti. Ceci est

confirmé par l'existence du mécanisme de synchronisation qui a la charge de maintenir cohérent un état partagé réparti sur plusieurs processus. Or la dimension "public-privé" du niveau conceptuel, qui correspond, selon la terminologie de l'auteur, à la dimension "partagé-répliqué", est orthogonale à la dimension "centralisé-réparti" du niveau implémentationnel. Ainsi, un composant public de l'architecture conceptuelle peut se traduire par un processus unique centralisé ou par un ensemble de processus répliqués synchronisés au niveau de l'architecture implémentationnelle.

Pour l'observabilité des actions et des ressources, si nous éliminons le mécanisme de synchronisation qui relève d'architecture implémentationnelle et non conceptuelle, nous constatons que le modèle n'autorise pas d'échanges entre deux états répliqués. Aussi l'observabilité implique un flot de données via un état partagé. Ainsi plus l'architecture est fermée selon ce modèle, qui n'autorise pas à ouvrir (états répliqués) à nouveau au-dessus d'un état partagé, plus le flot de données pour l'observabilité des actions et des ressources du contexte est court.

Le modèle suivant est le méta-modèle d'architecture de Dewan, une généralisation du modèle *Zipper*.

5.2. MÉTA-MODÈLE D'ARCHITECTURE DE DEWAN

Description

Le méta-modèle d'architecture pour les collecticiels de Dewan [Dewan 1999] est une généralisation du modèle Arch [Bass 1992] et du modèle de la fermeture éclair (*Zipper model*) [Patterson 1994]. Selon ce méta-modèle, un collecticiel est constitué d'un nombre variable de

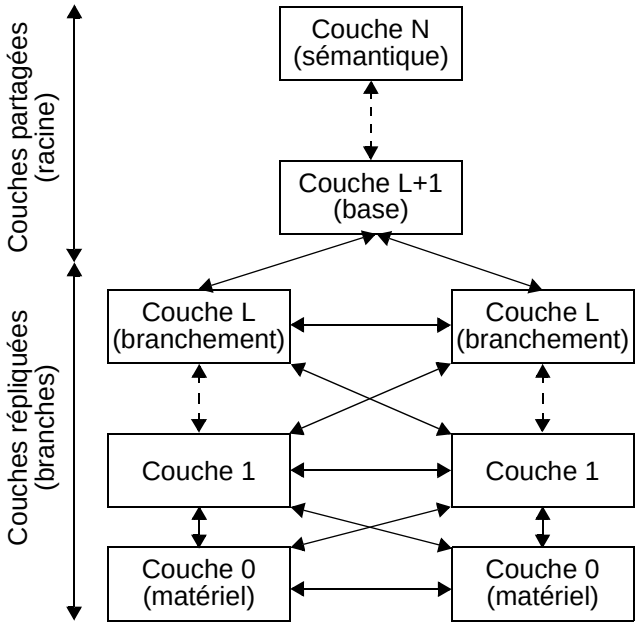


Figure 9
Méta-modèle d'architecture de Dewan

couches représentant plusieurs niveaux d'abstraction. Comme le montre la Figure 9, la couche la plus haute (niveau N) est de nature sémantique alors que la couche la plus basse (niveau 0) représente le matériel. Comparé au modèle Arch, la couche la plus haute correspond au Noyau Fonctionnel et la couche la plus basse correspond au Composant Physique d'Interaction.

L'architecture globale est constituée d'une racine et de plusieurs branches. La racine, comme le montre la Figure 9, est composée de couches partagées (niveaux L+1 à N). Les branches sont composées de couches répliquées (niveaux 0 à L), reliées à la racine au niveau de la couche L (point d'embranchement). De plus, les objets gérés par les branches sont privés. Réciproquement, les objets gérés par les couches partagées sont publics. Ainsi, selon notre terminologie, un composant partagé est un composant public et un composant répliqué est un composant privé. Les couches communiquent entre elles à l'aide de deux types d'événements : les événements d'interaction (échangés le long de l'empilement des couches) reflétant l'interaction de l'utilisateur avec le système et les événements de la collaboration (échangés entre les couches appartenant à deux branches différentes) reflétant l'interaction entre tous les utilisateurs.

Ce modèle distingue les couches collaboratives de celles qui implémentent uniquement des fonctionnalités mono-utilisateur. A travers cet étiquetage, Dewan propose un moyen pour identifier à quel niveau un collecticiel doit mettre en œuvre la collaboration, à savoir si la collaboration relève du Noyau Fonctionnel ou si elle relève plus de la Présentation. En ce sens, le degré de collaboration (*collaboration awareness degree*) indique quelle est la couche de plus haut niveau qui est collaborative, sachant que les couches situées à un niveau inférieur peuvent être aussi collaboratives. Ainsi, si le degré est faible, alors la collaboration est mise en œuvre au niveau de la Présentation. Si le degré est fort, alors la collaboration est mise en œuvre au niveau du Noyau Fonctionnel. Ce degré permet de déterminer les couches susceptibles de générer des actions concurrentes et ainsi de déterminer à quel niveau il est nécessaire d'implémenter des mécanismes de contrôle d'accès, de verrouillage des données ou de couplage des données.

Analyse

D'une part, ce modèle est une généralisation du modèle Arch. Cette généralisation se traduit par un nombre quelconque de couches et potentiellement supérieur aux cinq niveaux définis par le modèle Arch. Cependant cette généralisation rend le modèle moins précis que le modèle Arch puisqu'il ne détaille pas le rôle fonctionnel de chaque couche. En cela, ce modèle doit être vu comme un méta-modèle, à partir duquel plusieurs modèles d'architecture peuvent être dérivés. D'autre part, le

modèle hérite du modèle *Zipper* la notion de couches répliquées et partagées.

Ce méta-modèle, vis-à-vis de la première dimension de notre grille, distingue clairement les fonctions liées à des actions collectives de celles relevant d'actions individuelles. En effet, le méta-modèle définit un degré de collaboration qui identifie les couches répliquées implémentant des aspects collaboratifs et par conséquent les fonctions relevant d'actions collectives. Ainsi, les couches partagées ou répliquées non étiquetées collaboratives implémentent des fonctions mono-utilisateur. Néanmoins, le méta-modèle ne fait aucune distinction entre les fonctions relevant de la production, de la communication ou de la coordination.

Le méta-modèle explicite aussi les éléments de la seconde dimension de notre grille puisque les composants publics sont clairement identifiées à travers la notion de couches partagées et les composants publics à travers la notion de couches répliquées.

Le protocole d'échanges entre les couches est assuré par l'existence de deux types d'événements permettant aux différentes couches d'interagir, les événements d'interaction et les événements collaboratifs. Les événements collaboratifs permettent de définir des raccourcis pour le flot de données correspondant à l'observabilité des actions et des ressources du contexte. A l'opposé du modèle *Zipper*, qui impliquait un flot de données via un composant public, un échange direct de données entre le composant source et destinataire est possible pour réaliser l'observabilité d'actions et de ressources du contexte. Par exemple l'observabilité des actions individuelles ("conscience de groupe") peut être réalisée par un échange direct de données entre deux composants privés étiquetés collaboratifs. L'observabilité peut néanmoins être aussi réalisée par un flot de données via un composant public comme dans le modèle *Zipper*. Par exemple, le couplage de l'interaction (observabilité des actions) se traduit par le degré de fermeture de l'architecture : dans le cas d'un couplage fort, l'architecture est pratiquement refermée au niveau 0 ; dans le cas d'un couplage faible, l'architecture est pratiquement ouverte.

Néanmoins et hérité du modèle *Zipper*, il semble y avoir confusion entre architecture conceptuelle et architecture implémentationnelle. Le méta-modèle considère, comme dans le modèle *Zipper*, que les composants publics (i.e. couches partagées) sont alloués à des processus centralisés. Or ces composants publics peuvent, au niveau implémentationnel, être alloués à des processus répliqués synchronisés. De plus le méta-modèle autorise une architecture sans composant public, ce qui est impossible pour une architecture conceptuelle d'un collecticiel. En l'absence d'au moins un composant public, il n'y a pas de collaboration possible.

En étudiant dans le paragraphe suivant le modèle ALV, nous changeons de style d'architecture. D'un style monolithique, nous passons au style multi-agent.

5.3. ALV

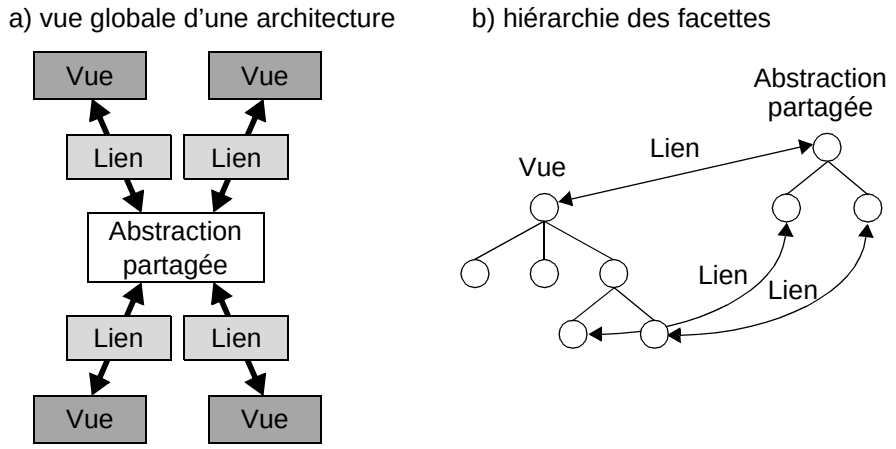


Figure 10
Modèle ALV

Description

Tout comme MVC est lié à SmallTalk, le modèle d'architecture ALV a donné lieu à la réalisation de la plate-forme Rendez-Vous [Hill 1994]. Cette dernière est une infrastructure permettant le développement d'applications collaboratives reposant sur la notion d'objets partagés. La plate-forme Rendez-Vous est analysée en détail dans le Chapitre V. Le collecticiel ainsi développé, selon la terminologie du modèle Arch, serait composé d'un unique Noyau Fonctionnel partagé pour plusieurs instances de la présentation. Une architecture selon ce modèle est une organisation d'agents dont les facettes sont, comme le montre la Figure 10 (a), les suivantes :

- **Une abstraction partagée** (A) qui gère les objets du domaine, partagés par tous les utilisateurs,

- **Une vue répliquée** (V) qui interprète les entrées d'un utilisateur et qui gère les sorties. Les événements générés par l'interaction sont traités au niveau de la vue par des fonctions dédiées qui modifient les données localement,

- **Un lien** (L) qui a, d'une part, la charge de relier la facette abstraction avec une facette vue et qui, d'autre part, doit s'assurer que les données locales à une vue sont conformes avec la représentation de la donnée au niveau de l'abstraction.

D'un point de vue global, l'architecture d'un collecticiel élaborée à l'aide du modèle ALV est constituée d'une abstraction partagée reliée à plusieurs vues réparties, une par utilisateur, à l'aide de liens. Par rapport à

notre terminologie, l'abstraction partagée est un composant public et une vue répliquée est un composant privé.

Ainsi, l'exemple de la Figure 10 (a) représente l'architecture du système à l'exécution avec quatre utilisateurs qui se traduit par l'existence de quatre vues se partageant l'abstraction à l'aide de quatre liens. A grain plus fin, cette architecture est constituée d'une multitude d'agents ALV dont les facettes Abstraction et Vue sont organisées, comme le montre la Figure 10 (b), sous la forme de deux hiérarchies d'agents qui constituent l'abstraction partagée et la vue globale. Le lien assure alors la mise en correspondance entre les facettes Abstraction et Vue d'un agent. Comme nous pouvons le constater, la vue globale et l'abstraction partagée ne sont pas obligatoirement constituées d'une hiérarchie identique de facettes. Il est alors possible que certaines facettes vue soient entièrement locales à la partie cliente et indépendantes de l'aspect collaboratif de l'application (non reliés à une facette abstraction).

Enfin, les liens ont la charge d'assurer la cohérence des données entre leur représentation abstraite et la valeur relative à la présentation. De plus, le lien est défini par un ensemble de contraintes traduisant le degré de couplage : selon le choix des contraintes, il est possible de coupler fortement une partie des vues (i.e. un sous-arbre de la hiérarchie) ; au contraire, certaines parties de la vue peuvent être totalement indépendantes.

Analyse

Ce modèle ALV est en fait une variation du modèle PAC. En effet :

- Les agents ALV et PAC sont constitués des mêmes facettes,
- Les agents ALV ou PAC se synchronisent par une facette Abstraction commune. Les facettes Lien (ou Contrôle en PAC) et Vue (ou Présentation en PAC) associées à la facette Abstraction sont crées pour chaque utilisateur du collecticiel.

Cependant, le modèle PAC propose une organisation des agents selon une hiérarchie traduisant des niveaux de concrétisation ou d'abstraction. Le modèle ALV au contraire ne préconise aucune règle pour structurer les deux hiérarchies de facettes vue et abstraction. Enfin, la partie abstraction est nécessairement partagée.

Ce modèle ne permet pas de distinguer les composants dédiés aux actions individuelles de ceux dédiés aux actions collectives.

Par rapport à la seconde dimension de notre grille, ce modèle considère clairement les ressources collectives, gérées par l'abstraction partagée, et les ressources individuelles, gérées par la vue. Cependant, cette approche semble restrictive puisque l'abstraction partagée et la vue globale sont obligatoirement un composant public et un composant privé. De plus, ce

modèle fait aussi la confusion entre aspect conceptuel et implémentationnel : l'abstraction partagée est nécessairement allouée à un processus centralisé (serveur) et les vues sont allouées à des processus répartis (terminaux graphiques).

Pour l'observabilité des ressources et des actions, nous constatons que le modèle autorise uniquement un échange entre les composants privés (vue) et le composant public (abstraction partagée) et qu'il n'y a aucun échange possible entre composants privés (vues). Aussi, l'observabilité implique, de façon similaire au modèle *Zipper*, un flot de données via le composant public et mis en œuvre par le lien reliant un composant privé au composant public. Par contre, ce modèle autorise à un utilisateur de disposer de données privées au niveau de la vue uniquement, ce qui est une forme d'observabilité des ressources individuelles.

Le paragraphe suivant présente le modèle Clock, un autre modèle multi-agent pour les collecticiels.

5.4. CLOCK ET DRAGONFLY

Description

Le modèle d'architecture Clock [Graham 1997] est un modèle multi-agent reposant sur une approche par composants et élaboré à partir du modèle MVC. Comme le montre la Figure 11 (a), un composant Clock, comparable à un agent MVC, est constitué de trois facettes :

- **Le modèle** gère les objets du domaine encodés sous forme de données de type abstrait (*ADT, Abstract Data type*),
- **Le contrôleur** interprète les interactions de l'utilisateur avec le système,
- **La vue** gère le rendu en sortie.

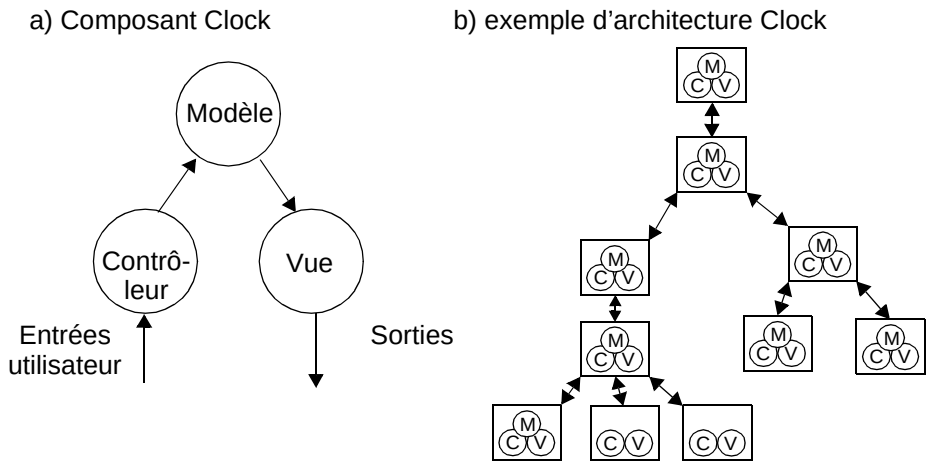


Figure 11
Modèle Clock
a) Composant Clock,
b) Exemple d'architecture.

La différence entre un composant Clock et un agent MVC réside sur le fait que les deux facettes contrôleur et vue ne peuvent plus communiquer entre elles. De plus, la communication de ces deux facettes avec le modèle est désormais mono-directionnelle : le contrôleur indique au modèle toutes les actions réalisées par l'utilisateur et la vue se contente de recevoir toutes les notifications de mise à jour de la présentation.

L'application collaborative développée à l'aide de ce modèle d'architecture se présente, comme le montre la Figure 11 (b), sous forme d'une hiérarchie de composants Clock. Cependant, il est tout à fait possible qu'un composant Clock n'implémente pas toutes les facettes, tout comme les agents du Contrôleur de Dialogue du modèle PAC-Amodeus. En général, les feuilles de la hiérarchie, comme le montre la Figure 11 (b), n'implémentent que la vue et le contrôleur. De plus, cette architecture introduit un mécanisme de contraintes entre les vues : la mise à jour d'une vue de la hiérarchie induit une mise à jour automatique de toutes les vues le long d'une branche. Enfin, comme le montre la Figure 11 (b), il est possible d'empiler plusieurs composants Clock à la racine de façon comparable à l'empilement des couches partagées du modèle de Dewan.

Dans ce modèle, le Noyau Fonctionnel correspond à l'empilement des composants constituant la racine de l'architecture. Les branches représentent l'organisation de l'interface. A l'exécution, chaque utilisateur dispose de sa propre instance de la hiérarchie de composants définissant l'interface. De plus, il est possible de spécifier quelles parties du Noyau Fonctionnel seront allouées à un processus centralisé et le reste à des processus répartis. Vis-à-vis de notre terminologie, les composants constituant la racine sont publics et ceux constituant les branches sont privés.

L'architecture Clock est accompagnée d'un langage déclaratif de programmation permettant de formaliser l'architecture d'un collecticiel. Un des points forts de cette approche est de coupler fortement l'aspect conceptuel et implémentationnel. L'environnement ClockWorks [Graham 1996] est une application de programmation visuelle qui permet de définir l'architecture d'une application dont le code est généré automatiquement grâce au langage Clock.

DragonFly [Anderson 2000] est une extension de Clock qui autorise les composants Clock à posséder leur propre mécanisme de cache de données et à choisir leur propre stratégie de maintenance de l'homogénéité des données. Cette dernière architecture sert de cadre à l'outil de développement TCD (*TeleComputing Developer*) écrit en Java.

Analyse

Le modèle Clock adopte une approche multi-agent comparable au modèle PAC. L'approche retenue présente des similarités avec le modèle ALV

puisque les composants du Noyau Fonctionnel sont nécessairement publics et que les composants constituant l'interface sont privés. De plus, le Noyau Fonctionnel est constitué d'une pile de composants publics de la même manière que l'empilement des couches partagées du modèle de Dewan.

Selon la première dimension de notre grille, le modèle ne propose pas de décomposition fonctionnelle distinguant les fonctionnalités relevant de l'action individuelle de celles relevant de l'action collective. A fortiori, le modèle ne fait aucune distinction entre les fonctionnalités relevant de la production, de la communication ou de la coordination.

Par contre, le modèle explicite les éléments de la seconde dimension de notre grille puisque les composants constituant la racine sont clairement identifiés comme étant publics et les composants constituant les branches comme étant privés. Néanmoins, le modèle Clock adopte la même approche que le modèle ALV puisque les composants privés sont nécessairement les composants constituant l'interface.

L'observabilité des actions est assurée par un protocole d'échanges qui propage les informations et les requêtes le long des branches de la hiérarchie. Le langage Clock offre un ensemble complet de services pour assurer l'interaction entre composants publics et privés. Cependant, ces services relèvent de la programmation, c'est-à-dire de l'implémentation. Ceci est légitime puisque l'objectif principal du modèle Clock est de pouvoir programmer directement l'application en définissant son architecture logicielle. Ces services offrent un moyen pour mettre en œuvre l'observabilité des actions. Néanmoins, le couplage de l'interaction s'exprime en terme de fermeture d'architecture de la même façon que le modèle de Dewan.

Il est en de même pour l'observabilité des ressources du contexte. Le langage Clock permet de spécifier quelles sont les données publiques et privées. Le support de cette propriété n'est pas explicite et la tâche est déléguée au développeur de l'application.

Le paragraphe suivant présente le modèle AMF-C qui affine le modèle PAC selon une approche multi-agent multi-facette.

5.5. AMF-C

Description

Le modèle AMF-C [Tarpin 1997] (AMF-Collaboratif) est une extension du modèle AMF pour les collecticiels (Agent Multi-Facettes [Ouadou 1994]) reposant sur une approche multi-agent multi-facette. Le modèle AMF, comparé au modèle PAC, ne se limite pas aux seules facettes Abstraction, Présentation et Contrôle. En effet, ce modèle autorise, comme le montre la Figure 12 (a), l'ajout de nouvelles facettes

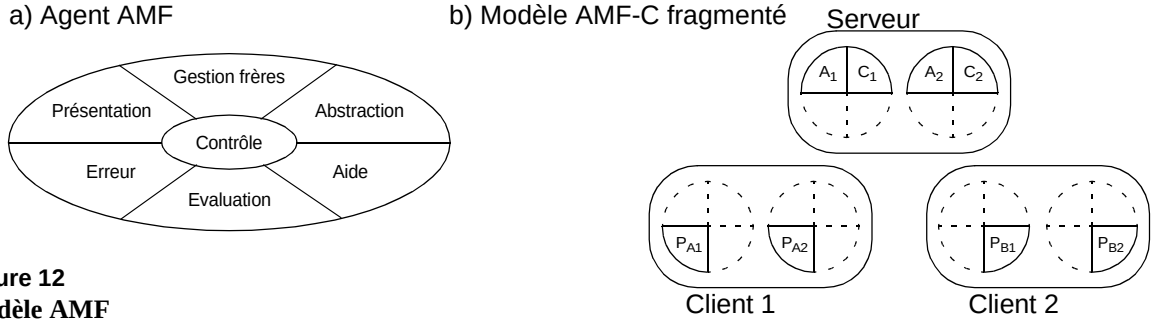


Figure 12
Modèle AMF

telles que la facette dédiée au traitement des erreurs ou la facette de gestion de l'aide. Ces facettes s'articulent autour de la facette Contrôle. De plus, ce modèle est accompagné d'un ensemble d'opérateurs permettant de décrire la nature des échanges et des traitements effectués entre deux facettes.

L'approche AMF-C repose sur la répartition des agents AMF sur différents sites selon deux stratégies : la fragmentation de l'agent et la réplication de l'agent.

La stratégie de fragmentation, comme le montre la Figure 12 (b), consiste à fragmenter et à répartir les facettes d'un agent sur différents sites. Selon l'exemple de la Figure 12 (b), les facettes Abstraction et Contrôle des deux agents AMF sont localisées sur le serveur et les deux facettes présentations sont respectivement localisées au niveau du client 1 et du client 2. Cette approche permet une gestion plus aisée des interactions dans un mode *WYSIWIS* strict : la même présentation est propagée au niveau de tous les utilisateurs. Néanmoins, quand la fragmentation est trop importante, c'est-à-dire quand trop de facettes sont réparties, il est préférable d'adopter une stratégie de réplication.

La stratégie de réplication considère l'agent dans son ensemble à l'aide d'un agent local et d'un agent de référence : un agent de référence est partagé par N clients à l'aide de N agents locaux. Ces agents locaux ont la charge de maintenir une vue cohérente de cet agent de référence partagé. Enfin, l'approche AMF-C s'accompagne de la notion de rôle définissant les droits d'accès aux agents partagés. Comparée à la stratégie précédente, celle-ci favorise le mode *WYSIWIS* relâché de par l'existence d'agents locaux.

Analyse

Les modèles AMF et AMF-C fournissent un découpage fonctionnel à une granularité plus fine que le modèle PAC puisqu'ils définissent de nouvelles facettes composant un agent. De plus, le rôle de la facette Contrôle est précisé, notamment par la définition d'un ensemble d'opérateurs permettant d'expliciter les relations et les transformations entre les autres facettes.

Vis-à-vis de notre grille d'analyse, le modèle AMF-C se concentre principalement sur l'élaboration de stratégies de répartition entre

différents sites et ne propose pas de découpage fonctionnel pour prendre en compte l'action de groupe au sens du modèle du trèfle. Néanmoins, ce modèle prend en compte la notion de composants publics et privés à travers la notion d'agent partagé et d'agent local.

Après l'analyse de trois modèles multi-agents, nous abordons maintenant l'étude des modèles hybrides, en commençant par le modèle CoPAC.

5.6. CoPAC

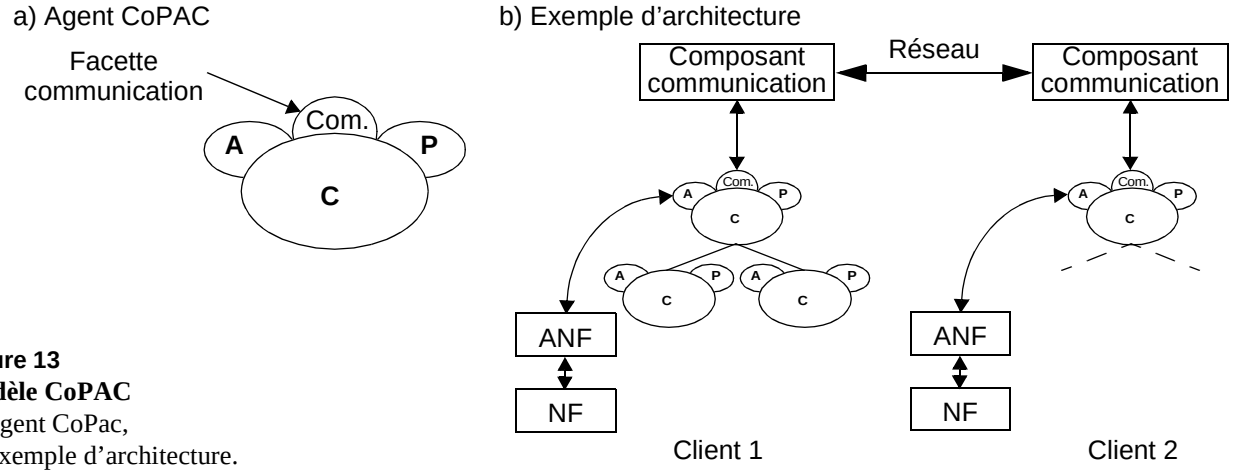


Figure 13
Modèle CoPAC
a) Agent CoPac,
b) Exemple d'architecture.

Description

CoPAC [Salber 1995] est une extension du modèle PAC-Amodeus pour les collecticiels. Selon ce modèle, il existe deux types d'agents, l'agent PAC usuel mono-utilisateur et l'agent collaboratif communiquant avec les autres agents collaboratifs. Il s'agit, comme le montre la Figure 13 (a), d'un agent PAC augmenté d'une nouvelle facette communication permettant aux agents collaboratifs de communiquer entre eux directement. La facette Contrôle se charge alors de distribuer les messages en provenance des facettes Abstraction et Présentation vers la facette Communication et réciproquement. Néanmoins, un agent collaboratif communique uniquement avec ses homologues qui occupent une place identique dans la hiérarchie d'agent peuplant le Contrôleur de Dialogue. L'ajout de cette nouvelle facette s'accompagne de l'introduction d'un niveau d'abstraction supplémentaire au modèle Arch, comme le montre la Figure 13 (b), avec l'existence d'un composant communication. Ce dernier est chargé de gérer l'accès au réseau sous-jacent et la communication avec les autres clients.

Le modèle CoPAC définit aussi un niveau de partage des cinq composants du modèle et des agents PAC. Ce niveau de partage définit le degré de couplage des différents composants selon trois niveaux :

- *Commun* : un composant commun correspond dans notre terminologie à un composant public.

- *Découplé* : un composant découplé correspond à un composant privé et indépendant, c'est-à-dire qui ne peut communiquer avec les autres instances de ce composant.

- *Couplé* : un composant couplé correspond en quelque sorte à la notion de composant synchronisé au sens de Patterson dans le modèle *Zipper*.

Analyse

Dans ce modèle, tout type de composant du modèle Arch peut être public ou privé. Ceci s'applique aussi à chaque agent PAC qui peuple le Contrôleur de Dialogue. Cette approche, contrairement à tous les modèles étudiés, autorise toutes les combinaisons possibles pour rendre des composants publics ou privés.

Selon la première dimension de notre grille, la distinction entre action collective et action individuelle est explicite puisque le modèle distingue les agents PAC mono-utilisateurs des agents CoPAC collaboratifs. De plus, l'action collective dédiée à la communication est rendue explicite par la nouvelle facette communication. L'action collective dédiée à la coordination est déléguée à la facette Contrôle.

Le modèle CoPAC, vis-à-vis de la seconde dimension, considère clairement la notion de composant public (commun) et privé (découplé).

Pour l'observabilité des actions et des ressources, nous constatons que le modèle propose un protocole d'échange entre agents CoPAC à travers la nouvelle facette communication. Cette dernière a la charge de diffuser les messages en provenance du contrôle et réciproquement. De plus, ce modèle définit un degré de couplage des composants.

Enfin, le modèle CoPAC ne propose aucune solution vis-à-vis des problèmes d'observabilité des ressources du contexte.

Le paragraphe suivant présente un autre modèle hybride, PAC*, qui est aussi une extension du modèle PAC-Amodeus.

5.7. PAC*

Description

Dans le modèle PAC* [Calvary 1997], le découpage fonctionnel d'un agent PAC est affiné selon les trois espaces fonctionnels du modèle du trèfle : production, communication et coordination. Comme le montre la Figure 14 (a), les trois facettes d'un agent PAC sont décomposées en trois parties dédiées chacune à une dimension du modèle du trèfle. Par exemple, la facette Abstraction est composée de trois sous-facettes dédiées respectivement à la production, à la communication et à la coordination. Cet agent est ainsi composé de trois tranches ce qui lui vaut le nom de "PAC Napolitain". Par conséquent, un agent couvre un espace fonctionnel, comme le montre la Figure 14 (a), selon deux dimensions

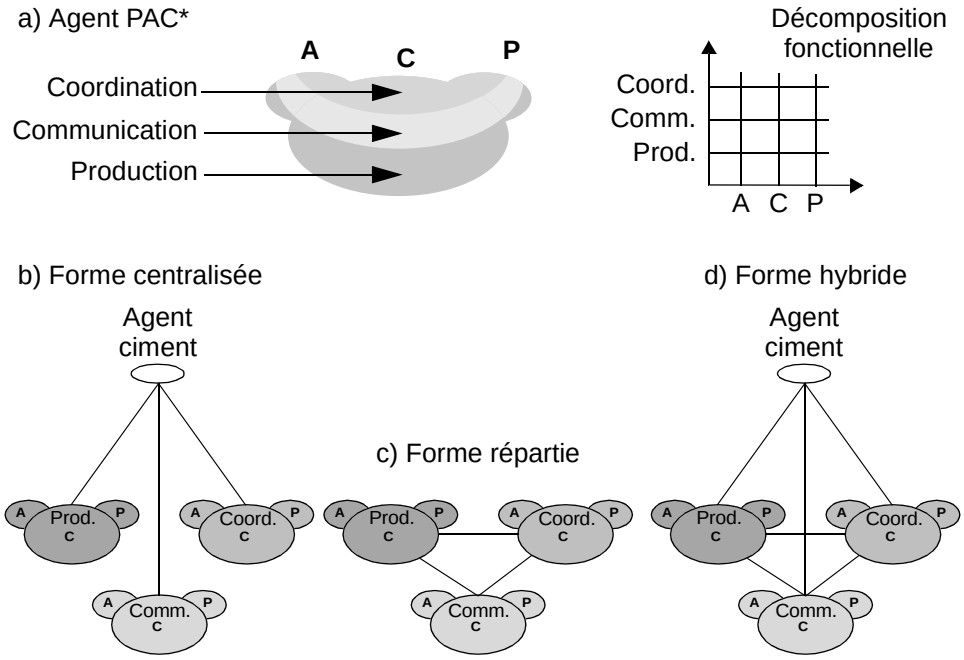


Figure 14
Modèle d'architecture PAC*
a) Agent PAC*,
b) Forme centralisée,
c) Forme répartie,
d) Forme hybride.

orthogonales, les trois facettes de l'agent PAC et les trois dimensions du modèle du trèfle.

Cependant, la vue compacte sous forme d'agent "PAC Napolitain" masque la structure de l'agent. En effet, comme le montre la Figure 14 (b, c et d), un agent PAC* peut exister sous trois formes différentes : la forme centralisée, la forme répartie et la forme hybride. Dans la forme centralisée (Figure 14 (b)), un agent PAC* est composé de trois agents dédiés respectivement à la production, à la communication et à la coordination et reliés à un agent ciment. Ce dernier assure la communication entre les trois agents et le reste du monde, c'est-à-dire les autres agents de la hiérarchie. Dans la forme hybride, les trois agents dédiés communiquent directement entre eux et assurent eux-même la liaison avec les autres agents de la hiérarchie. Enfin, la forme hybride est la combinaison des deux formes précédentes, sachant que l'agent ciment conserve toujours son rôle de lien avec le monde extérieur. La communication directe entre les trois agents dédiés permet d'accroître l'efficacité du système puisque la communication ne relève pas de l'agent ciment, ce qui peut entraîner un effet de "goulot d'étranglement" lié à la centralisation.

Dans ce modèle, un agent PAC* est comparable à un composant public et un agent PAC est comparable à un composant privé. Enfin, la communication entre agents s'inspire du modèle de Dewan. Néanmoins, en termes de réalisation logicielle, la communication peut être réalisée par un seul agent pour éviter une sur-multiplication des connexions et améliorer les performances de l'application collaborative.

Analyse

Ce modèle affine le modèle CoPAC puisqu'il considère explicitement les trois espaces fonctionnels du modèle du trèfle : production, communication et coordination. De plus, ce modèle s'inspire du modèle de Dewan pour définir un protocole d'échange entre agents PAC et PAC* appartenant à une même hiérarchie ou entre deux instances d'une hiérarchie.

Ce modèle, vis-à-vis de la première dimension de notre grille, est le seul à offrir un support complet à l'action collective (production, communication et coordination) par le biais de l'agent PAC*. L'agent PAC usuel répond au besoin de l'action individuelle.

Par rapport à la seconde dimension, le modèle PAC* hérite des propriétés du modèle de Dewan et n'apporte pas plus de précisions. Ainsi, un agent PAC rend explicite les ressources individuelles et un agent PAC* rend explicite les ressources collectives.

Vis-à-vis de la troisième dimension, le modèle hérite une fois de plus des propriétés du modèle de Dewan étant donné que le protocole d'échanges entre agents est inspiré du protocole d'échanges défini par le modèle de Dewan. En ce sens, le modèle PAC* rend explicites les éléments de cette dimension relative à l'observabilité des actions. Par contre, le modèle ne propose pas de solution explicite pour mettre en œuvre le couplage de l'interaction. Une solution pourrait être de s'inspirer du degré de couplage défini par le modèle CoPAC.

Enfin, le modèle PAC* ne propose aucune solution pour traiter l'observabilité des ressources du contexte.

6. Synthèse et conclusion

6.1. SYNTHÈSE

Nous dressons ici un bilan synthétique de l'analyse des modèles d'architecture conceptuelle existants selon notre grille d'analyse.

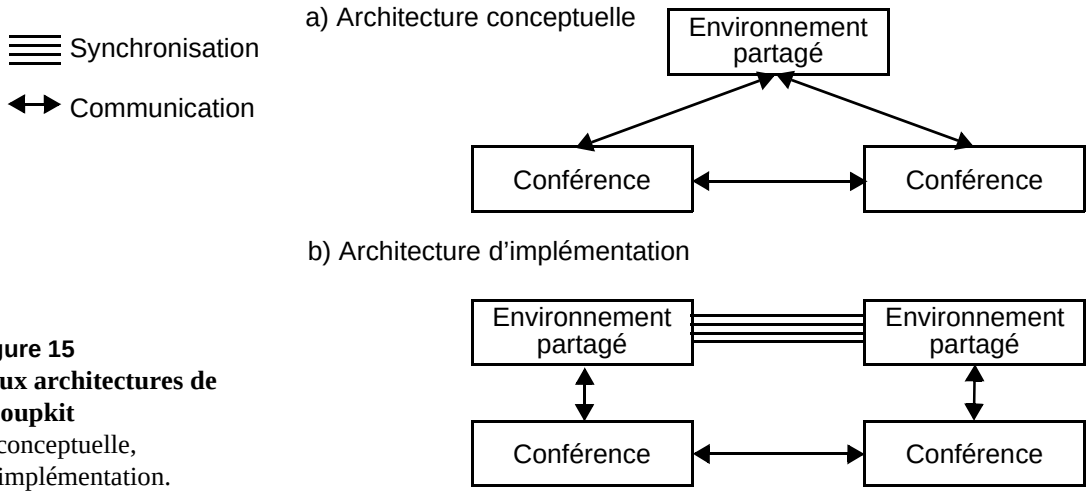


Figure 15
Deux architectures de Groupkit
a) conceptuelle,
b) implémentation.

Conceptuel vs Implémentationnel

La majorité de modèles d'architecture font la confusion entre architecture conceptuelle et implémentationnelle. En effet, la notion de composants **publics** et **privés** dans une architecture conceptuelle est orthogonale à la notion de composants **centralisés** et **répartis**. Cette dernière relève de l'architecture implémentationnelle puisque cela traduit la façon dont les processus sont alloués. A cette confusion, comme nous avons pu l'observer, s'ajoute l'utilisation courante du terme "**répliqué**" pour désigner un composant privé. Il apparaît que beaucoup de modèles d'architecture font la confusion entre ces deux notions, tels que le modèle *Zipper*, AMF-C, PAC* et dans une moindre mesure ALV et CoPAC. Par contre, le modèle Clock et le modèle de Dewan raisonnent en termes de composants publics et privés (i.e. composants partagés et répliqués), mais entretiennent le flou vis-à-vis de la notion de composants centralisés et répartis. Ceci ne permet pas de rendre explicite la notion d'action de groupe vis-à-vis de l'action individuelle. En effet, pour illustrer cette confusion, considérons GroupKit [Roseman 1992]. GroupKit est une boîte à outils écrite en Tcl/Tk, dont une analyse plus complète est présentée dans le Chapitre V. Cette boîte à outils dispose d'un mécanisme de haut niveau, appelé environnement partagé, qui permet aux applications collaboratives (une conférence dans la terminologie de cette boîte à outils) d'accéder à des données publiques. Chaque application possède sa propre copie de cet environnement sachant que les données contenues dans cet environnement sont dupliquées à l'identique. Dès qu'une application modifie une valeur dans sa copie, la modification est automatiquement répercutée au niveau de toutes les autres instances. Du

point de vue de l'implémentation, comme le montre la Figure 15 (b), cet environnement partagé est totalement répliqué et copié dans l'espace des conférences sachant qu'un mécanisme de synchronisation assure un état cohérent entre toutes les instances. D'un point de vue conceptuel, comme le montre la Figure 15 (a), l'environnement partagé peut être vu comme un composant public entre deux conférences (composants privés) puisque les conférences ont une vue identique sur l'environnement quelles que soient les modifications réalisées. En conclusion, cet exemple souligne la distinction entre architecture conceptuelle et architecture d'implémentation, distinction souvent floue dans les modèles d'architecture existants. En l'occurrence, l'existence d'un composant public dans une architecture conceptuelle n'implique en aucun cas la nécessité de disposer d'un processus centralisé à l'implémentation.

Actions collectives et individuelles

Le modèle PAC* et dans une certaine mesure le modèle CoPAC sont les deux seuls modèles à offrir un découpage fonctionnel relatif à l'action collective en prenant en compte les trois espaces fonctionnels du modèle du trèfle : production, communication et coordination. Le modèle de Dewan raisonne à plus gros grain en distinguant les fonctionnalités relevant de l'action collective des fonctionnalités relevant de l'action individuelle.

Les autres modèles, c'est-à-dire le modèle *Zipper*, ALV, Clock et AMF-C restent à très gros grain en ne faisant pas la différence entre ces deux types de fonctions.

Ressources collectives et individuelles

Tous les modèles identifient les ressources collectives en termes de composants publics et les ressources individuelles en termes de composants privés. Cependant, le modèle ALV et le modèle Clock comme pour les actions, sont plus restrictifs puisque les ressources collectives sont gérées uniquement par les composants du Noyau Fonctionnel et les ressources individuelles par les composants constituant l'interface.

Observabilité des actions

Vis-à-vis de la troisième dimension de notre grille, les éléments relatifs à l'observabilité des actions sont rendus explicites dans le modèle de Dewan et dans le modèle PAC*. Le protocole d'échanges est mis en place à l'aide des événements (collaboratifs et d'interaction) et, pour le modèle de Dewan, à l'aide du degré de fermeture de l'architecture. Il en est de même pour le modèle CoPAC qui met en œuvre ce protocole par le biais de la nouvelle facette communication de l'agent CoPAC et par le biais d'un degré de couplage définissant un degré de partage des composants.

Dans le cas des modèles ALV et *Zipper*, l'observabilité est mise en œuvre à travers un mécanisme de synchronisation des états et d'un lien maintenant la cohérence des données entre la facette vue et la facette

abstraction. Cette approche relève surtout de l'implémentation. De plus, le protocole d'échanges ainsi mis en place n'autorise pas les échanges entre deux composants privés.

Le modèle Clock est principalement tourné vers l'implémentation et, par le biais du langage Clock, il offre un pouvoir d'expression élaboré pour rendre observables les actions.

Observabilité des ressources

Seuls le modèle de Dewan et le modèle PAC* raisonnent sur l'observabilité des ressources du contexte, principalement à gros grain : soit les ressources sont privées et inaccessibles par les autres composants privés, soit les ressources sont publiques et conséquemment totalement accessibles et modifiables par tous les utilisateurs.

6.2. CONCLUSION

En conclusion, aucun modèle d'architecture étudié pour les collecticiels n'est entièrement satisfaisant vis-à-vis de notre grille d'analyse. Les deux modèles qui offrent la meilleure réponse sont le modèle de Dewan et le modèle PAC*, mais ils sont incomplets pour traiter explicitement des éléments liés à l'observabilité. Néanmoins, le modèle PAC* est le seul modèle à proposer une décomposition fonctionnelle relative à l'action collective qui soit complète vis-à-vis des trois espaces fonctionnels du modèle du trèfle.

Dans le chapitre suivant, nous présentons notre modèle d'architecture Clover, que nous motiverons par une analyse selon notre grille.

Chapitre IV *Le modèle et métamodèle d'architecture Clover*

1. Introduction

De l'étude des modèles d'architecture conceptuels du chapitre précédent, nous retenons deux modèles : le modèle de Dewan et le modèle PAC*. Par rapport aux éléments de l'activité de groupe organisés dans notre grille d'analyse, le modèle de Dewan préconise des composants publics et privés et un protocole d'échanges de données entre composants qui répond aux besoins d'observabilité des actions/ressources collectives et individuelles. Le modèle PAC*, quant à lui, véhicule une décomposition selon les trois facettes des actions collectives : production, communication et coordination.

Dans ce chapitre, nous proposons un nouveau modèle d'architecture conceptuelle, le modèle Clover qui étend le modèle de Dewan en appliquant la décomposition en trois facettes de PAC*. Le chapitre est organisé de la façon suivante : la première partie présente notre modèle d'architecture Clover et la seconde une généralisation du modèle, le métamodèle Clover. Une première validation du modèle Clover consiste à montrer qu'il correspond à une pratique implicite des développeurs. Aussi, dans la troisième partie nous étudions les architectures conceptuelles de trois collecticiels existants afin de montrer leur adéquation au modèle. Nous concluons enfin par un résumé des points contributifs du chapitre.

2. Modèle d'architecture Clover

A partir du métamodèle d'architecture Clover, présenté dans la partie suivante, plusieurs modèles d'architecture peuvent être dérivés. Nous présentons une instance de ce métamodèle dans cette partie : le modèle d'architecture Clover. En effet, il nous a semblé plus pédagogique d'avoir un discours du concret vers l'abstrait et donc d'exposer d'abord un modèle d'architecture, puis sa généralisation le métamodèle.

Dans cette partie, nous décrivons le modèle Clover puis son rationnel de conception. Nous soulignons ensuite les propriétés véhiculées par le modèle, pour conclure par un positionnement du modèle par rapport à notre grille d'analyse.

2.1. DESCRIPTION

Le modèle d'architecture Clover [Laurillau 2002a] [Laurillau 2002b], présenté à la Figure 1, est dérivé du métamodèle d'architecture générique pour les collecticiels de Dewan [Dewan 1999] en appliquant les cinq niveaux d'abstraction du modèle de référence Arch [Bass 1992]. Néanmoins, notre modèle compte six niveaux, c'est-à-dire un niveau de plus que le modèle Arch. En effet, le Noyau Fonctionnel (NF) est divisé en deux niveaux : le *Trèfle Fonctionnel **privé***, et le *Trèfle Fonctionnel **public***. Ce dernier gère l'ensemble des objets du domaine communs à l'ensemble des utilisateurs, en fonction des restrictions imposées par l'application, et offre un ensemble de services ou de fonctionnalités qui permettent de les manipuler. A l'opposé, le niveau privé gère l'ensemble des objets du domaine à caractère privé, c'est-à-dire propres à un seul utilisateur. L'ensemble des objets privés et publics constitue pour un utilisateur donné ce que Dewan nomme "l'état d'interaction" du système.

Dans la suite de la discussion, nous utiliserons les termes de niveau, couche et composant de manière interchangeable. Bien sûr, un composant peut englober plusieurs couches, mais pour simplifier le discours, nous considérons qu'il n'y a qu'une couche par composant.

Globalement, comme le montre la Figure 1, l'architecture est constituée de niveaux publics et privés. Tous les niveaux du modèle Arch sont privés à l'exception du Trèfle Fonctionnel **public**. Dans ce modèle, nous nous sommes principalement intéressés au Noyau Fonctionnel et nous n'avons émis aucune hypothèse sur la nature des autres niveaux. Pour simplifier la figure, nous n'avons pas représenté toutes les flèches symbolisant la communication entre les branches. Celles-ci sont représentées dans le métamodèle d'architecture de Dewan décrit au Chapitre III.

La particularité de cette architecture repose sur la structure des Trèfles Fonctionnels : une abstraction (*wrapper*) encapsulant trois sous-composants dédiés à chacune des trois facettes du modèle du trèfle, c'est-

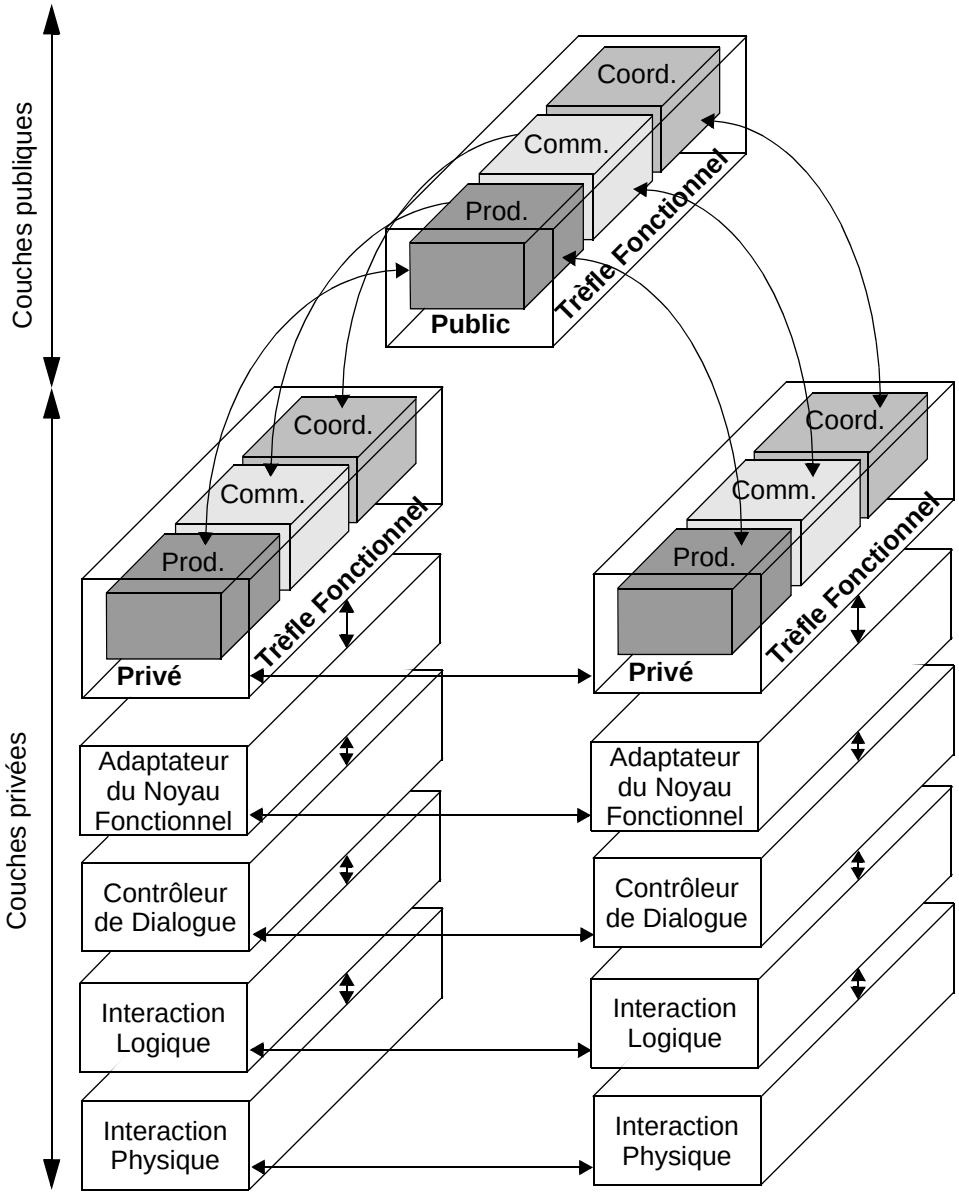


Figure 1
Modèle d'architecture Clover

à-dire trois sous-composants dédiés, respectivement, à la production, à la communication et à la coordination. Ainsi chaque sous-composant offre des services qui relèvent d'une des trois classes fonctionnelles du modèle du trèfle. Le rôle joué par l'abstraction du Trèfle Fonctionnel est triple :

- L'encapsulation réalisée par cette abstraction agit comme une sorte de glu fonctionnelle : elle assure la communication entre les trois sous-composants et les autres niveaux dans l'architecture. En effet, cette abstraction masque aux autres couches cette structure interne, organisant les fonctionnalités selon les trois facettes du modèle du trèfle. Ainsi, le composant peut être empilé avec des niveaux qui n'ont aucune connaissance de l'existence de ces trois sous-composants, c'est-à-dire qui ne différencient pas les fonctionnalités relevant de la production, de la communication ou de la coordination. Cette approche est aussi valide avec les niveaux qui n'ont aucune connaissance de

l'aspect collaboratif de l'application. Ainsi le Trèfle Fonctionnel **privé** joue le rôle de médiateur entre l'Adaptateur du Noyau Fonctionnel (niveau qui n'implémente pas nécessairement les trois types de sous-composants) et le Trèfle Fonctionnel **public**.

- Cette abstraction gère l'ensemble des objets du domaine communs aux trois sous-composants. Cependant, tous les objets ne sont pas identifiés comme relevant d'une des trois catégories définies par le modèle du trèfle. Au contraire, un objet peut être utilisé à la fois par les trois sous-composants. Par exemple, l'objet résultant de l'écriture d'un texte avec un système de tableau blanc partagé sert à la fois pour la communication (échange d'un message) et pour la production (contenu du tableau enrichi d'un nouveau dessin). Néanmoins, chaque sous-composant opère sur ces objets un traitement spécifique selon l'aspect production, communication ou coordination. Considérons l'exemple du tableau blanc partagé, un seul traitement est effectué, lié à la production, qui consiste à ajouter le nouveau dessin dans une image représentant le contenu du tableau. De plus, les ressources de ce composant sont filtrées selon une politique de publication comme définie dans le Chapitre II. Il est alors possible de définir un degré de publication du Trèfle Fonctionnel : ce degré peut varier du "totalement accessible" à "inaccessible" en passant par "partiellement accessible". Pour le Trèfle Fonctionnel **privé**, ce degré doit pouvoir être changé dynamiquement.

- Cette abstraction offre un ensemble de services, autres que ceux fournis par les trois sous-composants, incluant les services systèmes ainsi que les fonctionnalités relevant de l'action individuelle. En effet, ces services ne relèvent pas de l'un des trois sous-composants.

Au sein du métamodèle d'architecture de Dewan, la communication entre les différentes couches est possible grâce à deux types d'événements (Chapitre III) :

- Les événements d'interaction sont échangés le long des branches et de la racine.

- Les événements de collaboration sont échangés entre couches de branches distinctes.

Notre modèle respecte les mêmes règles que le modèle de Dewan. Cependant, les événements d'interaction et de collaboration sont différenciés selon trois catégories : production, communication et coordination. Cet affinement permet une communication directe entre sous-composants dédiés. Notre modèle dispose néanmoins d'un

événement générique qui assure la communication avec les autres couches qui ne sont pas des Trèfles Fonctionnels.

2.2. CHOIX DE CONCEPTION DU MODÈLE

Nous expliquons les choix qui ont été faits pour définir ce modèle d'architecture en considérant successivement les cinq niveaux d'abstraction du modèle Arch :

- Le Noyau Fonctionnel : les composants publics et privés du Noyau Fonctionnel et l'approche par composant.
- Le Contrôleur de Dialogue et L'Adaptateur du Noyau Fonctionnel : une extension au modèle PAC*
- Les Composants de Présentation : utilité du modèle du trèfle pour la conception des interfaces utilisateur.

Un Noyau Fonctionnel public et privé

Dans ce modèle d'architecture, le Noyau Fonctionnel est décomposé en deux parties, le Trèfle Fonctionnel public et le Trèfle Fonctionnel privé, permettant une distinction claire entre l'état public et l'état privé de l'application. En effet, comme il est souligné dans [Phillips 1999], tout collecticiel met œuvre un état public et un état privé.

Néanmoins, le Trèfle Fonctionnel **privé** peut être inexistant. C'est le cas des applications rendues collaboratives sans en modifier leurs codes : l'approche de développement intitulée "collaboration transparente". Cette approche consiste à transformer une application mono-utilisateur en une application multi-utilisateur grâce à la modification des bibliothèques systèmes et non du code de l'application. Ainsi, les fonctionnalités qui relèvent de l'action individuelle sont rendues collaboratives et l'état privé disparaît pour un état unique et public.

Par contre, il existe toujours un Trèfle Fonctionnel **public**, aussi réduit soit-il : nous pouvons considérer que le plus petit dénominateur commun est l'infrastructure de communication entre deux processus communicants (cela peut-être un réseau informatique ou une communication inter-processus si l'exécution de l'application est réalisée sur une seule et unique machine). Si cet état public est inexistant, alors les deux clients sont totalement indépendants et il n'y a plus aucune collaboration.

Imposant un Trèfle Fonctionnel **public**, notre approche diffère du modèle de Dewan. En effet le modèle de Dewan autorise une architecture totalement répliquée, c'est-à-dire une architecture sans couche publique. Dans notre architecture, cela correspondrait à l'absence du Trèfle Fonctionnel public. Cette divergence s'explique par le fait que le modèle de Dewan mélange les aspects d'une architecture conceptuelle avec ceux d'une architecture implémentationnelle. Comme nous l'avons expliqué, la

dimension "public-privé" d'un composant au sein d'une architecture conceptuelle est orthogonale à celle de "centralisé-réparti" d'un processus au sein d'une architecture implémentationnelle. En particulier, l'existence d'une couche publique dans une architecture conceptuelle n'implique en aucun cas la nécessité de disposer d'un processus centralisé à l'implémentation.

Enfin, l'architecture Clover est refermée (*zipped*) au niveau du Trèfle Fonctionnel public : le point de branchement. Ce point de branchement implique un degré de couplage faible. Un degré de couplage faible permet un mode *WYSIWIS* relâché (*What I See Is What You See*). A l'opposé, un degré de couplage fort, défini par un point de branchement au niveau des composants Interaction, correspond à un mode *WYSIWIS* strict. D'autres points de branchement et donc des degrés de couplage différents sont certes possibles. Cette possibilité est explicitée par le métamodèle Clover.

L'approche par composants

Les cinq niveaux d'abstraction du modèle Arch et le découpage fonctionnel selon le modèle du trèfle sont deux approches orthogonales. D'une part, le modèle Arch offre cinq niveaux d'abstraction permettant de définir une séparation claire entre l'interface utilisateur et la sémantique de l'application. D'autre part, le modèle du trèfle définit trois classes de fonctionnalités : production, communication et coordination.

Pour préserver les propriétés véhiculées par ces deux décompositions fonctionnelles, nous avons opté pour une approche par composants. Un composant est constitué d'une abstraction encapsulant trois sous-composants dédiés à la production, communication et coordination. Ainsi, l'abstraction préserve les propriétés véhiculées par le modèle Arch : il est possible d'empiler des couches de différentes natures, c'est-à-dire pouvant être collaborative ou non. Concrètement, dans le modèle Clover, le Trèfle Fonctionnel privé peut communiquer avec le Trèfle Fonctionnel public (un composant collaboratif) et l'Adaptateur de Noyau Fonctionnel (un composant non collaboratif).

Une extension au modèle PAC*

Le modèle PAC*, présenté au Chapitre III, est la version collaborative du modèle d'architecture hybride PAC-Amodeus. Dans le modèle PAC*, le Contrôleur de Dialogue est composé d'une hiérarchie d'agents PAC et PAC* qui échangent des événements avec le Noyau Fonctionnel via l'Adaptateur du Noyau Fonctionnel. Ainsi, notre modèle d'architecture Clover permet aux trois facettes Abstraction d'un agent PAC* (Chapitre III du Chapitre III) de communiquer avec le sous-composant correspondant du Trèfle Fonctionnel privé, comme le montre la Figure 2. Cette communication n'est pas directe puisque les événements transitent via l'Adaptateur du Noyau Fonctionnel (qui n'est pas nécessairement

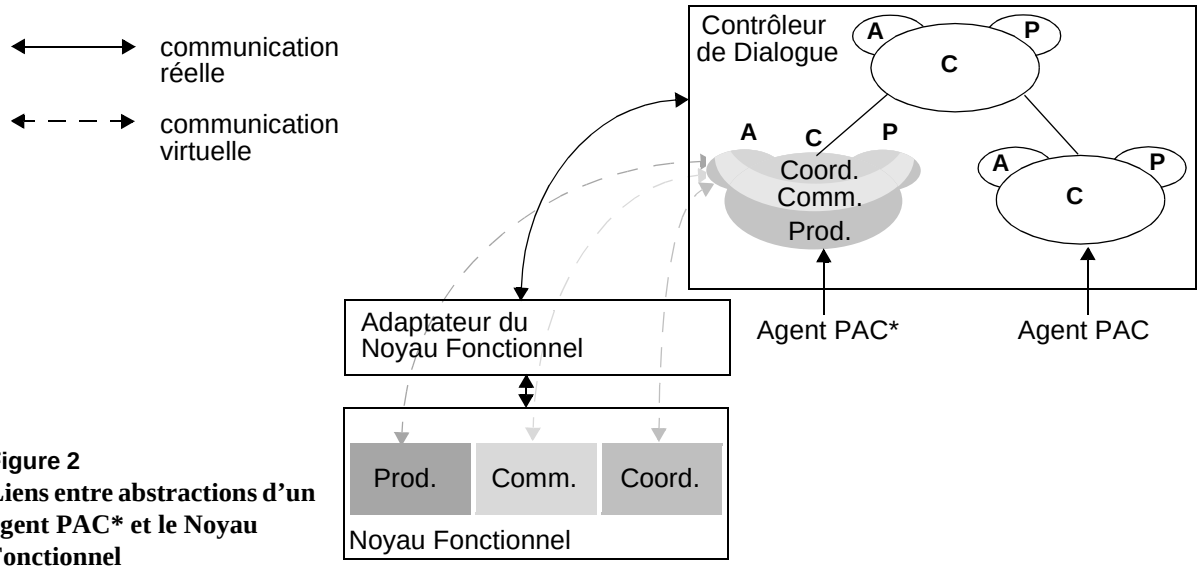


Figure 2
Liens entre abstractions d'un agent PAC* et le Noyau Fonctionnel

structuré selon les trois facettes du modèle du trèfle), puis via l'interface logicielle du Trèfle Fonctionnel privé**.**

De plus, comme le montre la Figure 14 du Chapitre III, un agent PAC* dans sa forme hybride est constitué d'un agent ciment. Ce dernier est chargé de gérer la communication entre les trois sous-agents et le monde extérieur, c'est-à-dire les autres agents de la hiérarchie. L'abstraction du Trèfle Fonctionnel de notre architecture Clover joue un rôle similaire étant donné qu'elle gère la communication entre les différentes couches (le monde extérieur) et les trois sous-composants dédiés à la production, communication et coordination.

L'interface utilisateur et le modèle du trèfle

Le modèle du trèfle, présenté au Chapitre II, classe les fonctionnalités d'un collecticiel selon trois espaces : espace de production, espace de communication et espace de coordination. Ce modèle est un support utile à la spécification fonctionnelle des collecticiels, mais semble difficilement applicable à la conception de l'interface homme-machine (IHM). En effet, de nombreux collecticiels proposent des objets graphiques d'interaction combinant les trois facettes du modèle du trèfle. Par exemple, la barre de défilement collaborative [Calvary 1997], version étendue de la barre de défilement de l'éditeur de texte SASSE [Baecker 1992], couvre les trois catégories du modèle du trèfle. Comme le montre la Figure 3, la barre de défilement de droite, dédiée à la production, est une barre de défilement ordinaire pour faire défiler un texte par exemple. La barre de défilement de gauche, dédiée à la communication et à la coordination, dispose de plusieurs curseurs, un par utilisateur observable à travers une vignette vidéo contenue à l'intérieur. Du point de vue fonctionnel, nous concluons que la position courante d'un utilisateur dans le texte contribue à la coordination, tandis que le service vidéo concerne la communication. Du point de vue de l'interface, un unique objet d'interaction, en l'occurrence la barre de défilement, est

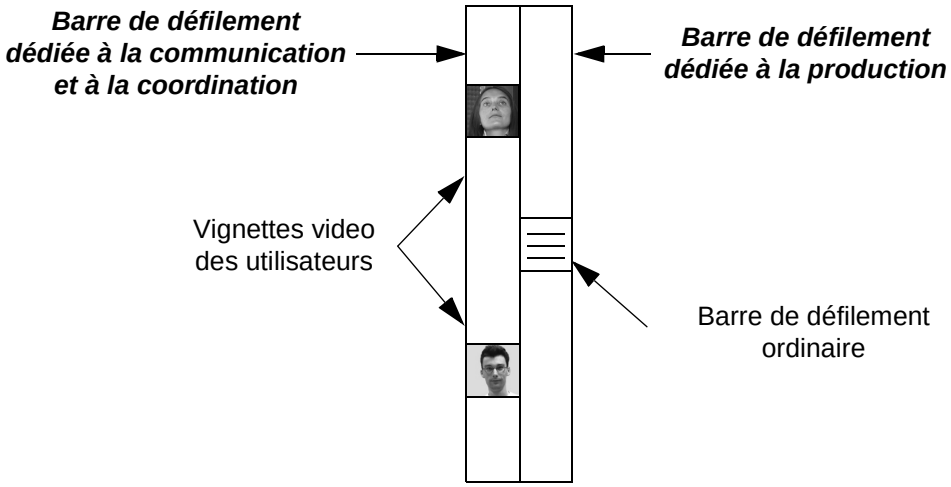


Figure 3
Barre de défilement collaborative

utilisé à la fois pour la communication et pour la coordination. Ceci souligne le fait qu'il est difficile, voire non souhaitable, pour l'utilisabilité du collecticiel de différencier les trois catégories de services au niveau des objets de présentation. Par conséquent, nous n'avons pas appliqué la décomposition fonctionnelle selon les trois espaces du modèle du trèfle aux deux composants de présentation.

2.3. PROPRIÉTÉS

Ce modèle d'architecture en trèfle véhicule les propriétés suivantes :

- Au niveau conceptuel, le découpage fonctionnel permet d'établir un lien direct entre les concepts issus de la phase de conception et la modélisation de l'architecture logicielle. En effet, le découpage fonctionnel selon le modèle du trèfle sert de cadre de conception pour identifier les fonctionnalités du collecticiel. Le découpage préconisé par le modèle Clover permet donc d'expliciter au niveau de l'architecture logicielle des concepts manipulés lors de la phase de conception du collecticiel et en particulier lors de la phase de spécification fonctionnelle.

- Au niveau implémentationnel, ce découpage fonctionnel se traduit par une modularité accrue du logiciel qui, à son tour, implique une plus grande modifiabilité. La modifiabilité est une propriété importante dans le cadre d'une conception itérative centrée sur l'utilisateur. Par exemple, prenons le cas d'un tableau blanc partagé, il serait aisé d'ajouter un service de communication vidéo en ajoutant un module dédié à la communication sans affecter les autres modules, en particulier celui dédié à la production gérant la zone de dessin.
  De plus, une forte modularité diminue la complexité de programmation. Par expérience, le développement du système CoVitesse, présenté au Chapitre VII, a été facilité grâce à l'architecture Clover. L'application du modèle a permis de développer chaque partie

sans être obligé d'avoir une vue globale de l'ensemble du logiciel. Par exemple, les trois sous-composants des Trèfles Fonctionnels publics et privés ont été développés séparément, diminuant très significativement la complexité de la programmation.

2.4. ARCHITECTURE CLOVER ET GRILLE D'ANALYSE

Outre les propriétés véhiculées par le modèle, nous étudions ici sa capacité à intégrer les éléments de l'activité de groupe traduite dans notre grille. Nous appliquons ici la même démarche pour notre étude des modèles d'architecture du chapitre précédent.

Actions collectives et individuelles

Le modèle d'architecture Clover rend explicites les actions collectives (production, communication et coordination) et les actions individuelles à travers la structuration des composants. Les composants collaboratifs (contenant les fonctions dédiées aux actions collectives) sont structurés en trois sous-composants. Les composants non collaboratifs (contenant les fonctions dédiées aux actions individuelles) ne sont pas affinés en sous-composant.

Ressources collectives et individuelles

Ce modèle dérive du modèle d'architecture de Dewan. Ainsi, il hérite de ses propriétés et notamment de la notion de composants publics et privés. Aussi, un composant public gère des ressources collectives tandis qu'un composant privé gère des ressources individuelles. Le modèle d'architecture Clover rend donc explicites les éléments de la grille d'analyse relatifs aux ressources collectives et individuelles.

Observabilité des actions et des ressources

Comme nous l'avons expliqué dans le paragraphe 2.1, nous affinons le protocole d'échanges entre composants du modèle de Dewan en identifiant trois types d'événements collaboratifs et d'interaction : ces types d'événements sont respectivement dédiés à la production, à la communication et à la coordination. Ce protocole d'échanges du modèle Clover permet de modéliser les flots de données liés à l'observabilité des actions et des ressources. Par exemple un échange d'événements de production entre deux composants de branches différentes peut traduire un flot de données relatif à la rétroaction de groupe.

Après avoir présenté un modèle d'architecture Clover et ses propriétés, nous présentons une généralisation du modèle, le métamodèle Clover.

3. Métamodèle d'architecture Clover

Le premier paragraphe présente notre métamodèle d'architecture et le paragraphe suivant expose les propriétés qu'il véhicule.

3.1. DESCRIPTION

Comme le métamodèle de Dewan, le métamodèle Clover, présenté à la Figure 4, structure un collecticiel en un nombre variable de couches. Si nous comparons le métamodèle Clover (Figure 4) avec le modèle Clover (Figure 3) de la partie précédente, la couche L désigne le Trèfle Fonctionnel **privé** tandis que les couches L+1 à N correspondent au Trèfle Fonctionnel **public**.

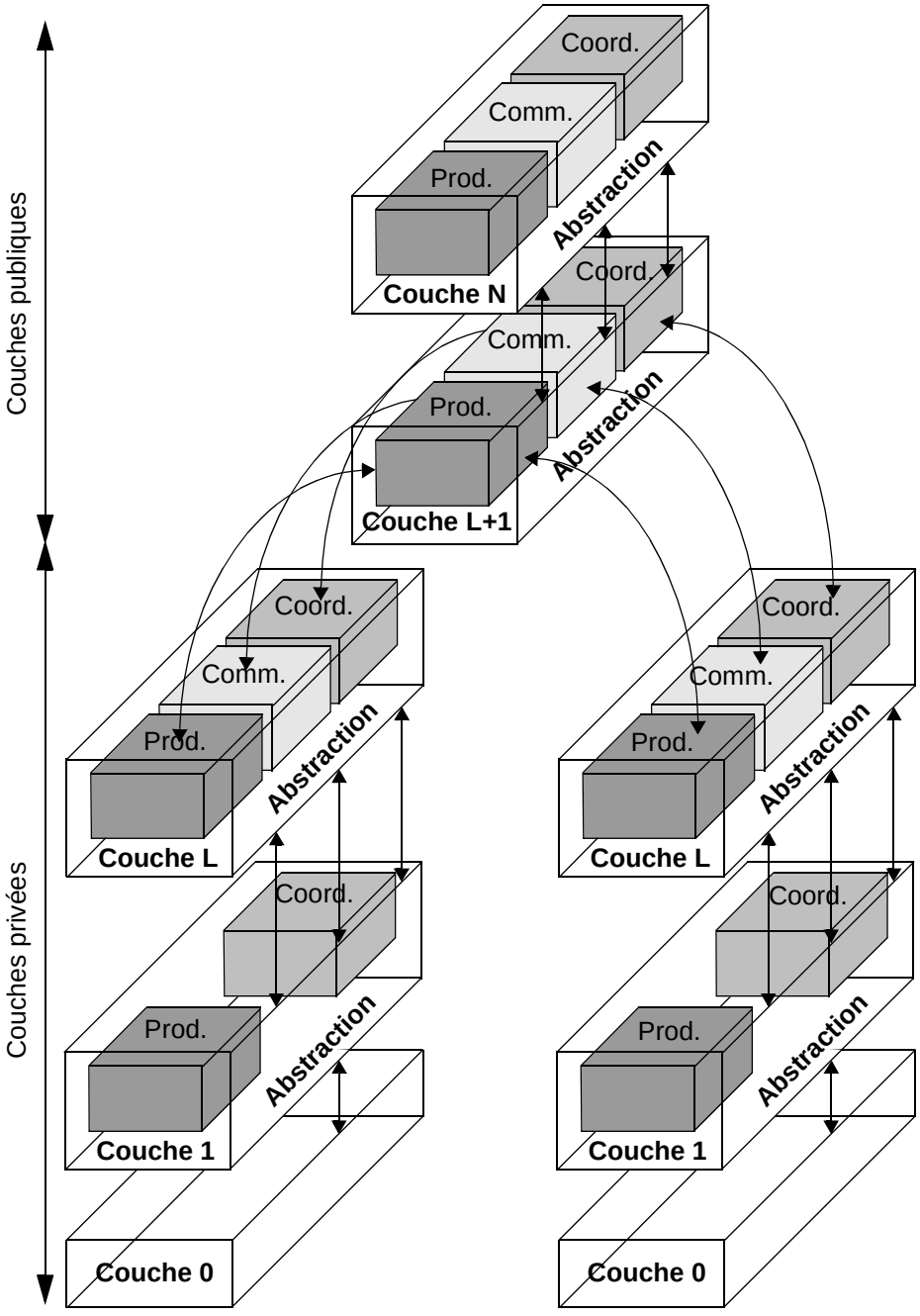


Figure 4
Métamodèle d'architecture Clover

L'originalité du métamodèle réside dans la structuration des unités logicielles (i.e. composants logiciels) constitutives de la racine et des branches, une unité étant notée Trèfle Fonctionnel (TF). Un Trèfle Fonctionnel est généralement décomposé en trois sous-composants (production, communication et coordination) englobés par une abstraction ou interface logicielle. La communication entre ces unités est réalisée grâce à des événements qui sont soit génériques, soit dédiés à la production, à la communication ou à la coordination. Notons que pour ne pas surcharger la Figure 4, nous n'avons pas représenté l'échange d'événements entre des unités de branches distinctes.

Le rôle principal de l'abstraction d'un composant Trèfle Fonctionnel est de masquer la structuration interne du composant. Ainsi un composant Trèfle Fonctionnel peut être empilé avec des composants non structurés selon les facettes du trèfle des collecticiels. En particulier, les couches les plus basses dépendantes du matériel ne sont généralement pas dédiées au développement de collecticiels et donc ne sont pas structurées selon les trois facettes du trèfle des collecticiels. Par exemple à la Figure 4, la couche 0 est un composant qui n'est pas détaillé en sous-composants qui seraient dédiés à la collaboration. De plus, la décomposition en trois sous-composants d'une unité Trèfle Fonctionnel n'est pas obligatoire. Un Trèfle Fonctionnel peut ne comporter que deux sous-composants, par exemple Production et Coordination comme la couche 1 de la Figure 4.

En synthèse, l'unité logicielle constitutive du métamodèle, un Trèfle Fonctionnel, contient une abstraction ou interface logicielle qui à son tour englobe éventuellement un, deux ou trois sous-composants, respectivement dédiés à la production, communication et coordination. La cohabitation de styles offre l'avantage de choisir le style le mieux adapté à telle ou telle couche de l'organisation structurelle. Inversement, l'hétérogénéité implique de maintenir une interface logicielle au niveau de chaque couche.

3.2. PROPRIÉTÉS

Le métamodèle ne fixe pas le nombre de couches. Cette flexibilité autorise l'ajout de couches pour mieux répartir les fonctionnalités, et permet ainsi d'accroître la modularité du code. Le découpage fonctionnel interne à une couche selon les trois facettes du trèfle des collecticiels contribue aussi à augmenter la modularité. Comme expliquée dans le paragraphe précédent, la modularité augmente la modifiabilité du code et réduit la complexité de programmation. De plus la modularité concourt à une meilleure réutilisabilité, réutilisabilité d'une couche complète ou d'une facette d'une couche, comme un service de communication.

Le métamodèle autorise un découpage fonctionnel partiel d'une couche : les trois sous-composants dédiés à la production, la communication et la coordination ne sont pas nécessairement tous présents. Cette flexibilité

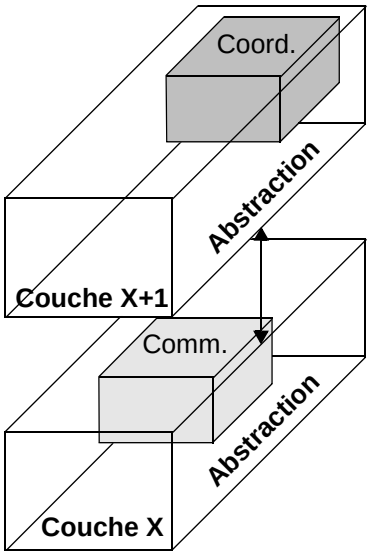


Figure 5
Partie d'une architecture conceptuelle d'un système de coordination reposant sur la communication

permet l'empilement de couches hétérogènes, dont le découpage fonctionnel est partiel. Par exemple, les systèmes de messagerie instantanée (*Instant Messaging*) sont souvent utilisés pour coordonner différents utilisateurs [Whittaker 2000]. Pour réaliser un tel système de coordination reposant sur la communication, l'architecture sera composée, par exemple, comme le montre la Figure 5, d'une couche dédiée à la coordination empilée au-dessus d'une couche dédiée à la communication.

De plus, le métamodèle considère la production, la communication et la coordination sur le même plan, à l'opposé des approches qui examinent la coordination indépendamment des aspects de production et de communication. Par exemple dans [Dewan 1999], le gestionnaire de session dédié à la coordination est un composant externe à l'architecture du collecticiel. Ce composant externe gère les utilisateurs et les groupes au cours d'une session, et donc a la responsabilité de créer dynamiquement des branches dans l'architecture logicielle. Dans le métamodèle Clover, le gestionnaire de session est conséquemment situé dans une couche publique de la racine. Au niveau de l'architecture conceptuelle, nous concevons toujours au moins une couche publique. Cette dernière n'implique pas une architecture implémentationnelle centralisée. Comme nous l'avons expliqué, la dimension "public-privé" du niveau conceptuel est orthogonale à la dimension "centralisé-réparti" du niveau implémentationnel. Ainsi une couche publique de l'architecture conceptuelle peut se traduire par un processus unique centralisé ou par un ensemble de processus répliqués synchronisés au niveau de l'architecture implémentationnelle.

Enfin, le métamodèle étend celui de Dewan [Dewan 1999]. Une architecture selon le métamodèle de Dewan est caractérisée par un degré de collaboration (*awareness degree*). Ce degré désigne le niveau de la couche la plus haute dans l'architecture qui est dépendante de l'aspect

collaboratif. Avec le métamodèle Clover, nous déclinons ce degré de collaboration en trois mesures : degré de production collaborative, degré de communication et degré de coordination. Cet affinement permet par exemple d'établir une classification des systèmes existants plus précise que celle décrite dans [Dewan 1999].

4. Etude de collecticiels existants

Nous venons d'exposer le métamodèle Clover et un modèle Clover instance de ce métamodèle. Valider un modèle d'architecture est une entreprise difficile, qui nécessiterait son application par de nombreuses équipes de développeurs pour la réalisation de collecticiels aux caractéristiques différentes. Nous verrons au Chapitre VI, comment nous l'avons appliqué pour la réalisation de deux collecticiels. Outre son application à la réalisation de collecticiels, une approche de validation du modèle est de montrer qu'il traduit une pratique logicielle, c'est-à-dire que le modèle explicite un savoir-faire implicite des développeurs de collecticiels. Ceci constitue l'objet de cette partie. Pour cela, nous étudions l'architecture conceptuelle de trois collecticiels existants à la lumière de notre modèle : le système MediaSpace, l'éditeur de texte collaboratif NetEdit et un jeu de ping-pong collaboratif. Avant d'analyser ces trois collecticiels, nous expliquons la méthode adoptée pour en déterminer l'architecture conceptuelle.

4.1. MÉTHODE

Les architectures conceptuelles ont été déduites à partir du code source et des documents de réalisation quand ceux-ci étaient disponibles. Les trois systèmes sont écrits en Java. L'architecture conceptuelle a été élaborée par un travail de rétro-ingénierie et ne correspond pas nécessairement à l'architecture telle qu'elle a pu être conçue par le développeur du collecticiel.

Lors de ce travail de rétro-ingénierie, nous avons particulièrement observé les deux points suivants : l'organisation des fonctionnalités selon les trois catégories fonctionnelles définies par le modèle du trèfle et la distinction entre espace public et espace privé. Pour ce dernier critère, la méthode a consisté à déterminer quelles sont les données partagées (échangées ou stockées sur un processus centralisé) et modifiables par tous les clients. Pour organiser les fonctionnalités selon la production, la coordination et la communication, nous avons classé les fonctions des modules existants (en général restreint à une seule classe).

Enfin notre étude s'est concentrée sur la partie Noyau Fonctionnel. Une partie du travail a donc consisté à séparer les modules relevant de l'interface utilisateur, des modules relevant du Noyau Fonctionnel.

4.2. SYSTÈME MEDIASPACE

Figure 6
Copie d'écran du système MediaSpace

Le système MediaSpace [Coutaz 1999], écrit en Java et développé au sein de l'équipe IIHM (*Ingénierie de l'Interaction Homme-Machine*) du laboratoire CLIPS-IMAG de Grenoble, est un système conçu pour favoriser et améliorer la conscience de groupe en facilitant la communication et les rencontres informelles entre collègues d'une même équipe. La communication est assurée essentiellement par la vidéo mais aussi par l'échange textuel (*chat*). Ce type de système s'adresse à des groupes de travail dont les membres sont distants : membres répartis entre deux étages, entre deux bâtiments, deux villes ou deux pays. Une grande attention a été portée sur la protection de la vie privée par le biais de filtres modifiant l'image de soi visible par les autres membres. Par exemple, comme le montre la Figure 6, l'image du bureau 206 ne permet pas d'identifier ce qui est affiché sur l'écran de la personne présente dans la pièce puisque la zone de l'écran est masquée par un filtre. Différents filtres sont actuellement disponibles tels que le store vénitien (Bureau 207 de la Figure 6), le filtre d'activité qui ne représente que le contour d'objets qui ont été déplacés (technique de différences d'images) et le filtre d'obstruction d'une zone sélectionnée qui, par traitement numérique,

donne l'impression d'être vidée de son contenu (image du Bureau 206 dans la Figure 6).

Il est aussi possible de sélectionner dans la liste complète des utilisateurs, qui l'on souhaite voir ou ne pas voir. Automatiquement, la mosaïque des images est mise à jour en retirant celles qui ne sont plus désirées. Une réglette de zoom (*slider*), située à droite à la Figure 6, permet de grossir certaines images. Le système permet aussi de consulter les informations concernant les autres utilisateurs si celles-ci ont été rendues observables.

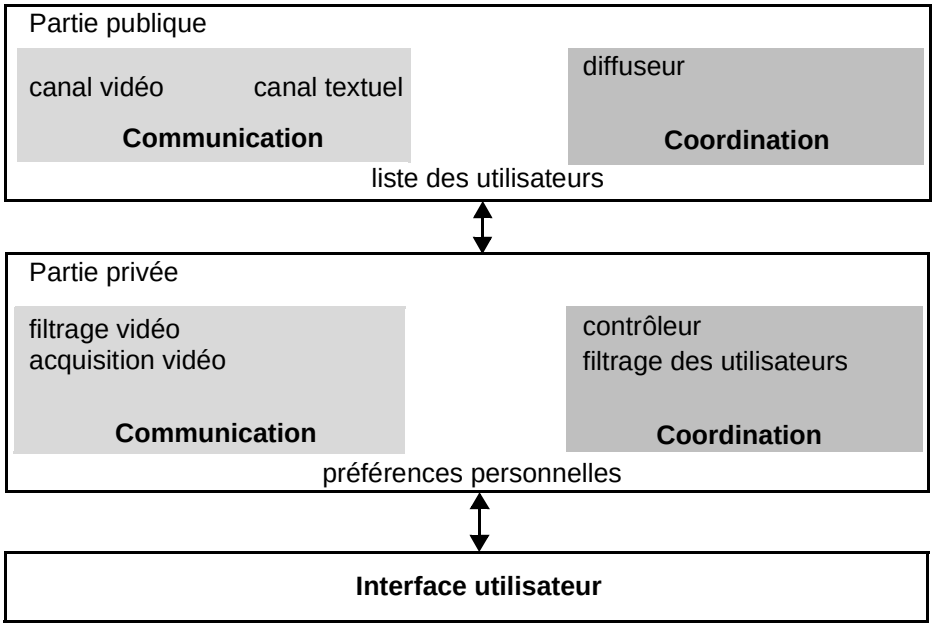


Figure 7
Architecture conceptuelle du système MediaSpace

Architecture conceptuelle

Au niveau implémentationnel, notons que le système est entièrement distribué : il n'y a aucun processus centralisé.

Comme le montre l'architecture conceptuelle de la Figure 7, le système repose principalement sur la communication et la coordination.

La communication entre les différents utilisateurs est possible grâce à deux canaux partagés, l'un pour la vidéo et l'autre pour les messages textuels (*chat*). Ces deux services sont localisés dans la partie publique. Du point de vue de la coordination, le composant public offre un service de diffusion, reposant sur le protocole multicast, qui avertit l'ensemble des utilisateurs dès qu'un nouveau membre rejoint la session en cours. La principale donnée publique est la liste des utilisateurs qui est mise à jour dès qu'un utilisateur entre ou quitte la session en cours.

La partie privée implémente aussi des composants dédiés à la communication et à la coordination. Le composant dédié à la communication correspond aux services d'acquisition vidéo et de filtrage. Le service de filtrage engendre une modification de l'image en fonction du filtre. Dans le système actuel, il s'agit soit du store vénitien, soit du filtre d'activité.

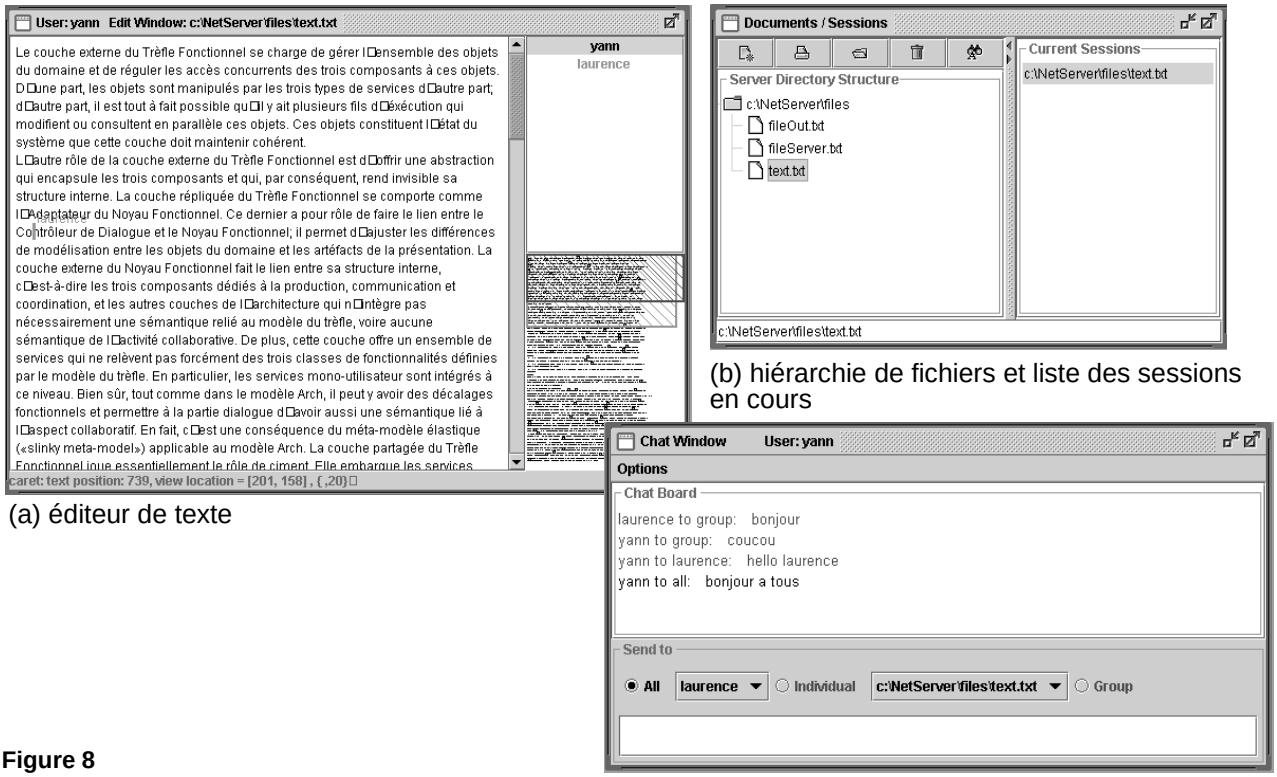
Le composant dédié à la coordination est composé d'un contrôleur qui assure la diffusion d'une mise à jour d'informations personnelles (nom, numéro de téléphone, bureau, etc) de client à client. Ces données sont considérées comme sensibles ce qui justifie une communication de client à client reposant sur le protocole TCP. D'autre part, ce composant offre aussi un service de filtrage des utilisateurs qui permet à un client de sélectionner les utilisateurs qu'il souhaite voir. Enfin, cette partie privée gère les préférences utilisateurs, c'est-à-dire le filtre activé et les informations personnelles le concernant.

Les autres couches sont dédiées à la gestion de l'interface utilisateur.

En synthèse, les actions collectives se limitent à joindre et quitter une session en cours et les actions individuelles, à consulter les informations sur les autres utilisateurs (identité et vidéo) ainsi qu'à paramétrer l'interaction par le biais de filtres. Les ressources collectives incluent une liste des utilisateurs et les images vidéo. Les ressources individuelles sont constituées des informations personnelles, de son image vidéo et des préférences de filtrage. La conscience de groupe est principalement assurée par la mise à jour des images vidéo. La rétroaction de groupe se limite à l'apparition et la disparition d'images vidéo qui correspondent à l'arrivée ou au départ d'un nouveau membre de la session. Enfin, l'ensemble des ressources sont collectives, c'est-à-dire la liste des membres, est totalement observable, tandis que sa propre image vidéo peut-être filtrée ou complètement cachée. La réciprocité est de plus mise en œuvre lorsqu'un utilisateur ne désire pas être vu ou ne pas voir un autre utilisateur : l'image des deux est alors cachée au niveau de l'interface des deux clients.

4.3. EDITEUR COLLABORATIF NETEDIT

L'éditeur collaboratif NetEdit [Zafer 2001], écrit en Java, permet l'édition multiple de documents par plusieurs utilisateurs. Les différentes sessions sont associées à un document. Pour créer ou accéder à une session en cours, un utilisateur peut obtenir la liste des sessions et la liste des fichiers disponibles à l'aide d'une fenêtre représentant la hiérarchie des fichiers (*file browser*) du disque dur. Comme le montre la Figure 8 (b), cette fenêtre est divisée en deux parties, l'une contenant la hiérarchie de fichiers, l'autre contenant l'ensemble des sessions d'édition de documents. Un double-clic sur le nom d'une session fait apparaître un menu contextuel affichant la liste des participants. La sélection d'une session en cours permet à l'utilisateur de rejoindre automatiquement cette session et fait apparaître l'éditeur de texte (Figure 8 (a)). La barre de défilement du texte est une barre de défilement collaborative qui a pour effet de déplacer la zone de texte pour tous les membres de la session. Une zone radar (*radar view*) à droite du texte indique, par différentes couleurs, les zones visibles par les différents utilisateurs. De plus, un



(a) éditeur de texte

(b) hiérarchie de fichiers et liste des sessions en cours

(c) forum de discussion

Figure 8
Copie d'écran de l'éditeur NetEdit

curseur coloré est assigné à chaque utilisateur pour rendre observable la position de chacun dans le texte. Enfin une troisième zone en haut à droite du texte, donne la liste des participants. L'outil propose aussi un forum de discussion, visible à la Figure 8 (c), avec la possibilité d'envoyer un message à l'ensemble des utilisateurs, toutes sessions confondues, à une personne en particulier ou bien à un groupe d'utilisateurs, membres d'une session en cours. La notion de groupe est assimilée aux utilisateurs modifiant le document.

Architecture conceptuelle

L'architecture d'implémentation repose sur un modèle client-serveur. Au sein de l'architecture conceptuelle de la Figure 9, le serveur correspond à la partie publique et le client à la partie privée.

L'ensemble des données publiques est stocké sur le serveur. La partie publique est constituée de trois composants : un premier pour la production qui est dédié à l'édition partagée, un deuxième pour la communication qui assure l'ensemble des échanges de messages entre les différents utilisateurs et les différentes sessions, et un troisième pour la coordination qui gère à la fois les sessions d'édition et les changements d'états à diffuser aux participants (rétroaction de groupe). Ces changements d'états incluent les modifications apportées dans le texte et les déplacements de la barre de défilement collaborative pour mettre à jour les vues de tous les utilisateurs et les zones radars. Le serveur

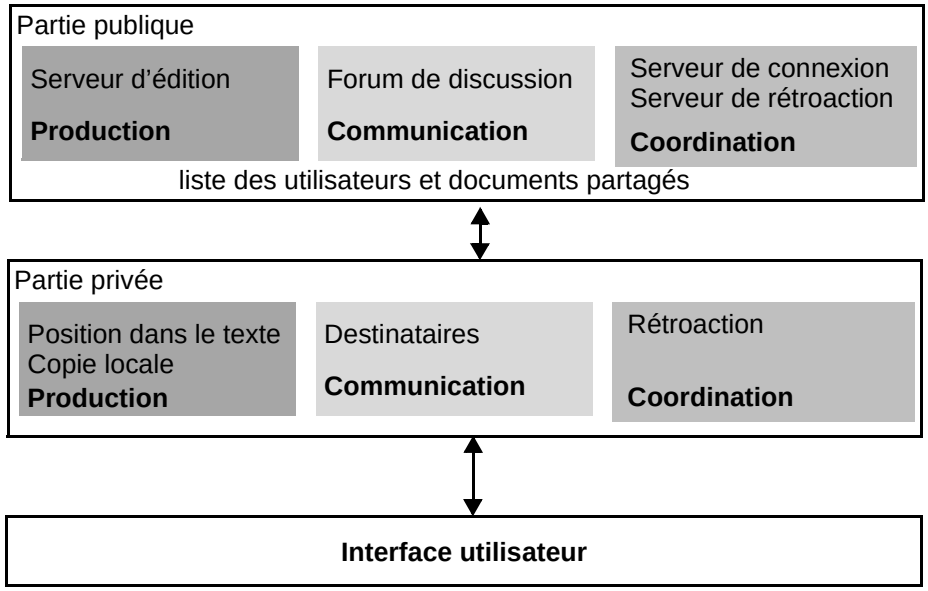


Figure 9
Architecture conceptuelle de l'éditeur NetEdit

d'édition gère les actions concurrentes d'insertion et de suppression de texte pour assurer un état cohérent parmi toutes les copies du texte.

Localement à chaque client, la partie privée implémente trois composants distincts. Le composant dédié à la production gère la copie locale et les modifications locales apportées par l'utilisateur. Le composant dédié à la communication gère principalement la liste des destinataires des messages et le composant dédié à la coordination les mises à jour en provenance des autres utilisateurs.

Les autres composants sont dédiés à la gestion de l'interface utilisateur.

En résumé, le code du système NetEdit contient des composants relatifs à l'action collective (production, communication et coordination). De plus, le Noyau Fonctionnel est décomposé en composants publics et privés. Cependant, les ressources individuelles gérées par les composants privés sont essentiellement des données liées à l'interface. Toutefois, nous n'avons pu identifier que quelques ressources relevant uniquement du Noyau Fonctionnel comme la liste des destinataires d'un message dans le forum de discussion. D'autre part, le système NetEdit met explicitement en œuvre un service de rétroaction qui a la charge de répercuter au niveau de tous les clients l'exécution de toutes actions collectives comme l'envoi de message, l'arrivée et le départ d'un membre de la session courante ou la modification du texte. Enfin, toutes les ressources, individuelles et collectives, sont entièrement observables. Toutefois, il existe un système de filtrage en sortie dans le forum de discussion qui permet d'envoyer un message à un nombre restreint de correspondants.

4.4. JEU DE PING-PONG COLLABORATIF

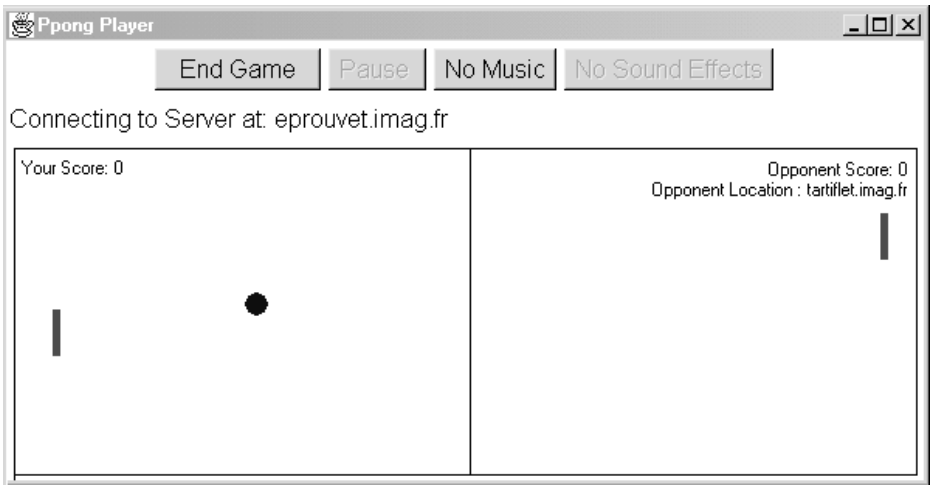


Figure 10
Copie d'écran du jeu de ping-pong collaboratif

Le jeu de ping-pong collaboratif est une application écrite en Java et basée sur la boîte à outils JSDT (*Java Shared Data Toolkit*) de Sun [Java 2002]. Ce jeu est issu des exemples fournis avec le kit de développement. Le principe est très simple : deux joueurs se connectent via un serveur pour réaliser une partie de ping-pong. Comme le montre la Figure 10, l'écran représente la table de ping-pong avec les raquettes de chaque joueur et la balle qui évolue sur le terrain. Le jeu démarre dès que les deux joueurs ont appuyé sur le bouton "start game". Un joueur voit évoluer la raquette de son adversaire. Le système gère la synchronisation entre les deux clients pour assurer une cohérence de l'état entre les deux vues. Le jeu s'arrête dès que la partie est terminée ou dès que l'un des joueurs quitte la partie.

Architecture conceptuelle

Le système est principalement dédié à la coordination, c'est-à-dire le démarrage des parties et la mise en relation des joueurs. L'architecture d'implémentation repose sur un modèle client-serveur, comme dans NetEdit. Comme le montre la Figure 11, la partie serveur est publique, tandis que la partie client est privée.

La partie publique (serveur) gère l'ensemble des parties en cours, l'ensemble des joueurs, calcule le score et gère l'évolution de la balle. La partie publique implémente principalement un composant dédié à la coordination qui propose des services d'enregistrement des joueurs et des parties, et gère le démarrage des différentes parties. Cette partie publique implémente un composant minimal dédié à la production de gestion des scores.

La partie privée, gérée à l'exécution par les processus clients, implémente uniquement un composant dédié à la coordination dont le principal service est de préparer le joueur au démarrage d'une partie. La communication des changements d'états, comme le montre la Figure 11, est directe entre les clients. En effet, les parties privées échangent

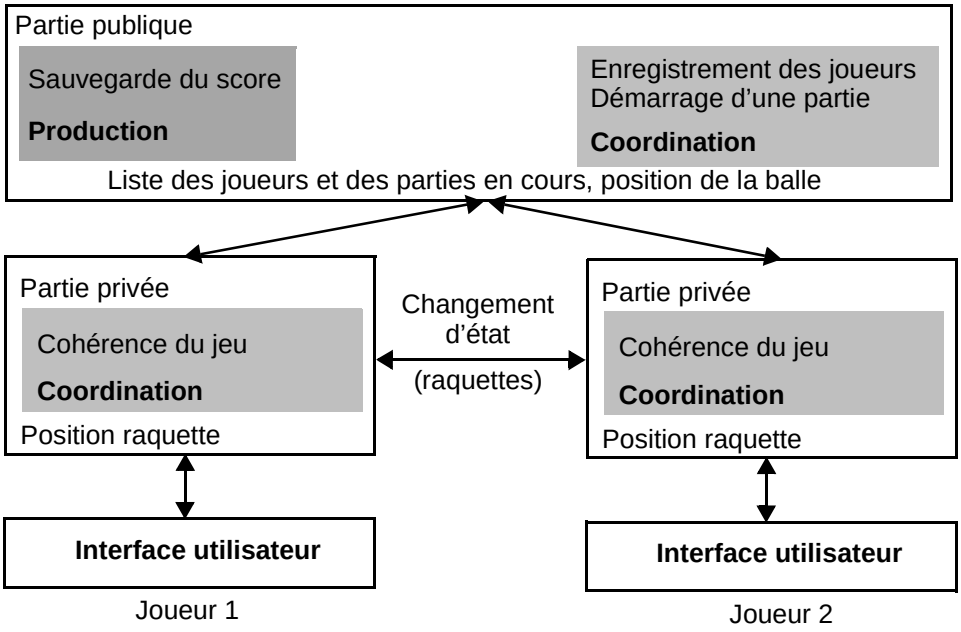


Figure 11
Architecture conceptuelle du jeu de ping-pong collaboratif

directement des messages indiquant les positions des raquettes ainsi que les interruptions éventuelles du jeu (quitter, arrêter ou redémarrer le jeu). Enfin, cette partie privée gère la position de la raquette.

Les autres parties du code implémentent l'interface utilisateur.

En résumé, le système identifie des ressources collectives comme le score, la liste des joueurs ou la position de la balle, des ressources individuelles comme la position de la raquette. La rétroaction de groupe est assurée par un composant public (affichage du score, attente ou départ d'un joueur) et par l'échange de messages de type coordination entre des composants privés (position des raquettes).

4.5. SYNTHÈSE

En synthèse, pour les trois systèmes, nous concluons que les développeurs ont organisé les fonctionnalités selon les trois espaces fonctionnels du modèle du trèfle. De plus nous avons constaté qu'outre la modularité (trois composants), une dépendance limitée entre les composants production, communication et coordination. Le système MédiaSpace inclut deux modules distincts dédiés à la communication et à la production. Le système NetEdit met en œuvre les trois types de modules (production, communication et coordination). Néanmoins, la coordination est minime et consiste essentiellement à synchroniser les objets graphiques collaboratifs comme la zone radar indiquant les vues des différents utilisateurs sur le texte. Le jeu de Ping-Pong met essentiellement en œuvre la coordination, sachant que la production se limite à la gestion des scores.

D'autre part, nous observons, dans le cas du système NetEdit et du jeu de Ping-Pong, une séparation claire entre composants privés et composants publics. Ceci est essentiellement dû au fait que l'architecture

d'implémentation repose sur un modèle client-serveur : la partie privée est associée au client tandis que la partie publique est associée au serveur. Cette séparation est moins nette dans le système MédiaSpace. En effet, le système repose sur une architecture d'implémentation totalement répartie. Cependant, il nous a été possible de déterminer la partie publique de la partie privée en recensant quelles sont les fonctionnalités et les données accessibles par tous les clients. Pour la partie publique, il s'agit des canaux de communication ainsi que les fonctions de gestion de session (joindre/quitter une session). La partie privée est associée à l'ensemble des préférences de filtrage des images (communication) et de filtrage des membres de la session (coordination).

5. *Résumé des contributions*

Nous retenons de ce chapitre les deux points contributifs suivants :

- **le métamodèle d'architecture Clover :**
  Nous avons proposé un métamodèle d'architecture logicielle, Clover, pour les collecticiels. Ce métamodèle résulte de la combinaison de l'approche en couches publiques et privées du métamodèle de Dewan avec la décomposition fonctionnelle du trèfle des collecticiels. L'originalité du métamodèle réside dans la structuration de ses unités logicielles constitutives en trois sous-composants optionnels (production, communication et coordination) englobés par une interface logicielle. Le métamodèle Clover résulte d'une combinaison motivée de modèles d'architecture existants (Chapitre III), sélectionnés pour leurs propriétés complémentaires, vis-à-vis de notre grille d'analyse.

- **un modèle d'architecture Clover :**
  Un modèle d'architecture Clover, instance du métamodèle, a été présenté. Ce modèle affine le Noyau Fonctionnel en deux composants, l'un public, l'autre privé. Ces deux composants constitutifs du Noyau Fonctionnel sont à leur tour décomposés selon les trois facettes du trèfle des collecticiels, pour obtenir trois sous-composants encapsulés par une interface logicielle. Cette interface a pour rôle de masquer la décomposition interne du composant selon les facettes du trèfle. Le système CoVitesse entièrement conçu à partir de ce modèle, fournit un exemple d'application du modèle au Chapitre VII.

Pour le méta-modèle et le modèle Clover, nous avons exposé les propriétés véhiculées. Fidèles à notre démarche de travail, nous avons aussi analysé le modèle Clover à la lumière de notre grille d'analyse. Enfin, nous avons décortiqué le code de trois collecticiels afin d'en étudier leurs architectures conceptuelles par rapport à notre modèle. L'objectif est une première tentative de validation du modèle en montrant que le modèle traduit une pratique logicielle, un savoir-faire implicite des développeurs de collecticiels.

L'étape suivante est de définir un outil de développement de collecticiels qui respectent notre modèle d'architecture Clover. L'outil de développement constitue alors un support à l'application du modèle. Ceci fait l'objet des chapitres suivants. Nous commençons dans le chapitre suivant par étudier les outils existants.

Chapitre V *Outils de développement de collecticiels*

1. Introduction

Dans le chapitre précédent, nous avons présenté un modèle d'architecture pour les collecticiels, le modèle d'architecture Clover. Le modèle préconise que le Noyau Fonctionnel soit structuré selon trois sous-composants dédiés à chacune des facettes du modèle du trèfle : production, communication et coordination. L'étude architecturale de trois collecticiels a permis de montrer que les développeurs, s'appuyant sur un savoir-faire implicite, ont organisé les fonctions suivant ces trois facettes.

Le passage d'un modèle d'architecture à la réalisation d'un outil de développement de collecticiels organisés selon le modèle considéré est une étape classique : l'outil de développement constitue un support à l'application du modèle. Pour les interfaces mono-utilisateur, citons par exemple le générateur d'interfaces SERPENT qui repose sur le modèle d'architecture Arch (Chapitre III). Pour les collecticiels, citons le modèle d'architecture DragonFly (Chapitre III) et la boîte à outils TCD ou encore le modèle d'architecture ALV (Chapitre III) et son environnement de développement Rendez-Vous. Notre objectif est de concevoir et développer une infrastructure pour le développement de collecticiels organisés selon notre modèle Clover. Aussi, dans ce chapitre, nous étudions les outils existants, puis nous présentons notre plate-forme de développement au chapitre suivant.

L'analyse des outils existants est conduite de la même manière que celle des modèles d'architecture (Chapitre III). En effet, nous étudions les outils par rapport à la grille d'analyse définie au Chapitre II. A ces fins et tandis que pour les modèles d'architecture, la grille avait été traduite en termes architecturaux, la grille est traduite ici en termes de solutions

logicielles, dans la partie 3. Nous pouvons alors étudier les outils existants dans la partie 4, à la lumière de la grille ainsi traduite, pour conclure le chapitre par un bilan comparatif des outils existants. Nous débutons ce chapitre, consacré aux outils de réalisation, par une typologie des outils existants.

2. Typologie des outils de développement

Avant d'analyser les outils existants selon notre grille, nous définissons dans cette partie les classes d'outils. Nous présentons d'abord trois approches de développement d'un collecticiel, pour ensuite cerner les classes d'outils qui font l'objet de notre étude.

2.1. APPROCHES DE DÉVELOPPEMENT

Il convient d'étudier les approches de développement d'un collecticiel afin de situer notre étude. En effet selon l'approche adoptée, les outils sont différents. Nous distinguons trois approches de développement d'un collecticiel :

- **la collaboration "implicite"** (*collaboration unaware*) : il s'agit d'applications mono-utilisateur exécutées dans un environnement partagé comme VNC [VNC 2002] ou Timbuktu [Netopia 2002]. Par exemple, les deux systèmes VNC et Timbuktu permettent l'accès simultané par plusieurs à une seule machine : ainsi il est possible d'utiliser une application mono-utilisateur par le biais d'un unique pointeur partagé par tous les clients.

- **la collaboration "transparente"** (*collaboration transparency*) : l'approche consiste à transformer une application mono-utilisateur en une application multi-utilisateur, tout en conservant intact le code de l'application initiale. La technique se fonde sur le détournement des appels standards des bibliothèques systèmes vers des bibliothèques chargées de gérer l'aspect collaboratif, comme le transport des données sur le réseau, la gestion des accès concurrents ou encore la diffusion des événements générés par les objets graphiques de l'interface utilisateur. Flexible JAMM [Begole 1999] et JCE [Abdel-Wahad 1999] sont des boîtes à outils écrites en Java qui s'inscrivent dans cette approche de développement d'un collecticiel : la librairie AWT (*Abstract Window Toolkit*, boîte à outils graphique standard de Java) est modifiée pour rendre les objets graphiques collaboratifs : barre de défilement collaboratif, télépointeur, canevas partagé constituent des exemples d'objets graphiques collaboratifs.

- **la collaboration "explicite"** (*collaboration aware*) : ce sont des collecticiels conçus et développés pour favoriser l'activité collaborative.

Nous cernons notre étude à la **collaboration explicite pour des collecticiels synchrones**. Pour cette approche de développement, nous différencions deux classes d'outils : les boîtes à outils et les plates-formes.

2.2. LES BOÎTES À OUTILS

Une boîte à outils (*toolkit*) est une bibliothèque de composants logiciels accessibles par le programme via des appels procéduraux. L'abstraction offerte par les composants de la bibliothèque réduit l'effort de programmation

Les composants d'une boîte à outils sont souvent spécialisés. C'est le cas des boîtes à outils pour la programmation d'interfaces utilisateur (*UI toolkit*) dont les principaux composants sont des objets graphiques (*widget*) tels que les boutons ou les barres de défilement. Par exemple, dans le monde de la programmation Java [Java 2002], la première boîte à outils graphique développée par Sun est l'*AWT* (*Abstract Window Toolkit*). Un autre exemple de spécialisation est l'*ARToolkit* [ARToolkit 2002] qui est une boîte à outils dédiée à la programmation d'applications de réalité augmentée.

Nous constatons que les boîtes à outils sont de plus en plus volumineuses : le nombre de composants augmente, impliquant une prise en main par le développeur de plus en plus difficile. Par exemple, l'*AWT* a migré vers la bibliothèque *Swing* qui offre plus du double de composants.

Les boîtes à outils, que nous étudions, sont celles dédiées au développement de collecticiels (*groupware toolkit* ou *CSCW toolkit*). Comme nous le verrons lors de l'analyse des outils existants, la majorité des boîtes à outils sont étendues pour définir des plates-formes.

2.3. LES PLATES-FORMES ET INFRASTRUCTURES

Il n'existe pas de définition consensuelle d'une plate-forme logicielle ou d'une infrastructure logicielle. Nous définissons une infrastructure ou plate-forme comme une boîte à outils imposant un schéma d'exécution, c'est-à-dire une répartition des processus lors de son utilisation. Le terme couramment employé est le terme d'architecture de distribution. Néanmoins pour éviter de faire la confusion avec la notion d'architecture conceptuelle et implémentationnelle, nous utilisons le terme, schéma d'exécution.

Il existe trois types de schémas d'exécution : centralisé, répliqué et hybride. L'étude menée par G. Phillips [Phillips 1999] décrit plus en détail ces différents schémas et leurs variations.

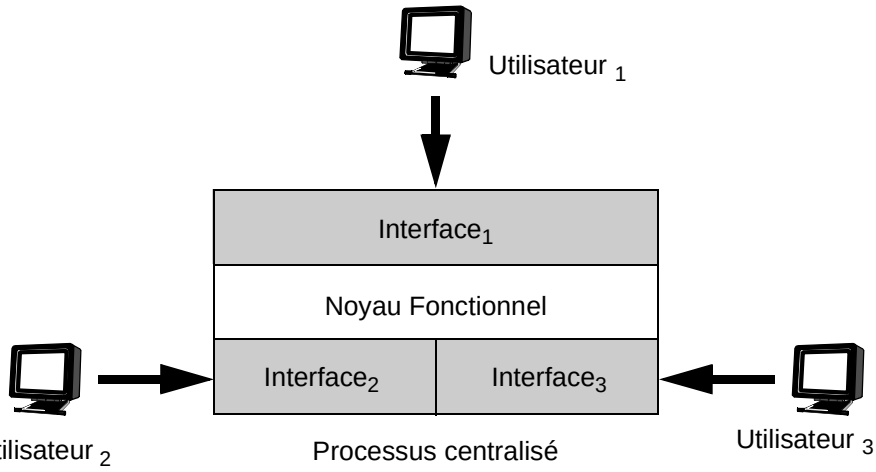


Figure 1
Schéma d'exécution centralisée

Le schéma d'exécution centralisé consiste à l'exécution d'un seul processus localisé sur un unique serveur. Ce processus gère à la fois, comme le montre la Figure 1, le noyau fonctionnel de l'application et les interfaces de chaque client. Dans ce schéma, chaque utilisateur dispose de sa propre instance d'interface. Par exemple, le système X-Windows [X-Windows 2002] repose sur ce schéma. L'intérêt principal de cette organisation est d'éliminer la majorité des problèmes liés aux accès concurrents et au maintien de l'homogénéité des données. Le développement d'une application multi-utilisateur en est ainsi facilité. Cependant, une implémentation reposant sur un unique processus se traduira par des performances fortement dégradées. Les pertes de performance sont surtout dues aux retards induits par l'infrastructure réseau et par l'effet "goulot d'étranglement" au niveau du processus centralisé. Par exemple un retour d'information immédiat ne peut être garanti au niveau des stations de chaque utilisateur. Ces problèmes sont perceptibles avec le système X-Windows par exemple.

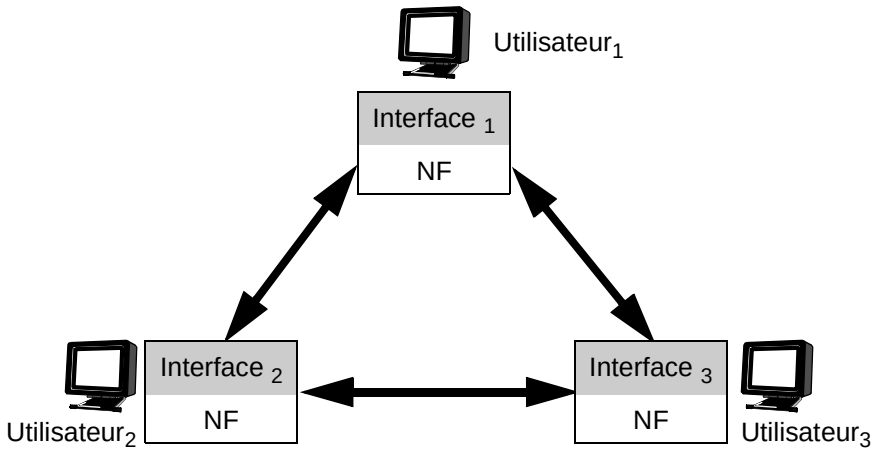


Figure 2
Schéma d'exécution répliquée

Le second schéma, à l'opposé du schéma centralisé, est une distribution des processus totalement répartie. Comme le montre la Figure 2, à chaque client est associé un processus gérant à la fois l'interface et une version répliquée du noyau fonctionnel. L'intérêt majeur de ce modèle est d'offrir

des systèmes très réactifs. L'interface accède directement au noyau fonctionnel et cette fois-ci, il n'y a plus d'effet de "goulot d'étranglement" : les processus de chaque utilisateur communiquent un à un. De plus, le système est peu sensible aux pannes : si l'un des clients s'arrête de fonctionner, le système global est toujours actif. Cependant, le développement d'un collecticiel est nettement plus complexe selon ce schéma. En effet, l'état de chaque noyau fonctionnel répliqué doit rester cohérent et tout changement d'état de l'une des versions répliquées doit automatiquement être diffusé à l'ensemble des clients avec le temps de retard imposé par l'infrastructure réseau. Notamment, le programmeur doit mettre en œuvre des mécanismes gérant les accès concurrents aux données partagées. Enfin, ce modèle est peu adapté au cas d'utilisateurs joignant une session en cours de route. En effet, ce cas nécessite de mémoriser l'historique de l'activité pour que le nouvel arrivant se retrouve avec le même état que les autres clients : à ces fins, chaque client doit stocker localement les actions réalisées.

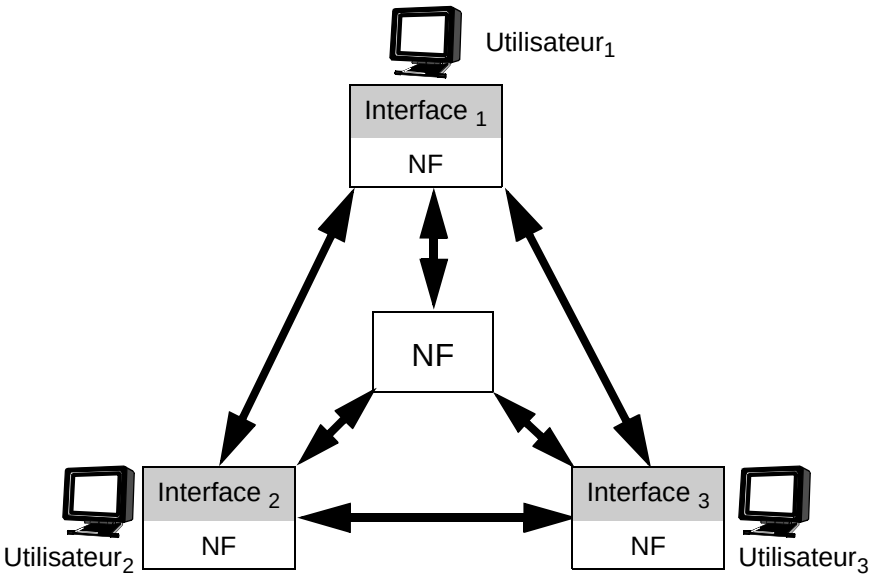


Figure 3
Schéma d'exécution hybride

Enfin, le troisième schéma, combinaison du modèle centralisé et répliqué, est un schéma hybride. Comme le montre la Figure 3, chaque client dispose de sa propre instance d'interface, de sa propre instance du Noyau Fonctionnel et partage avec les autres clients un Noyau Fonctionnel commun. Ce dernier est géré par un processus centralisé. Ainsi, les Noyaux Fonctionnels propres à chaque client sont indépendants les uns des autres. Le processus centralisé, qui abrite le Noyau Fonctionnel commun, synchronise l'ensemble des clients pour garantir une vue cohérente de l'état du système. Ce modèle est un bon compromis entre les deux schémas précédents car il assure de bonnes performances en termes d'interaction et d'accès aux données partagées. Cependant, l'implémentation reste plus complexe que le cas du schéma centralisé,

mais il n'est plus nécessaire de mettre en œuvre des algorithmes de maintien de la cohérence des données répliquées.

Nous examinerons certains outils qui imposent un schéma d'exécution. Par exemple, l'outil Rendez-Vous [Hill 1994], repose sur un système de fenêtrage partagé basé sur le système X-Windows [X-Windows 2002] qui à l'exécution impose que tous les processus clients soient centralisés sur le même serveur. A l'opposé d'autres outils offrent aux développeurs le choix du schéma d'exécution. Enfin certains outils permettent un schéma d'exécution dynamique qui s'adapte automatiquement à l'utilisation du collecticiel.

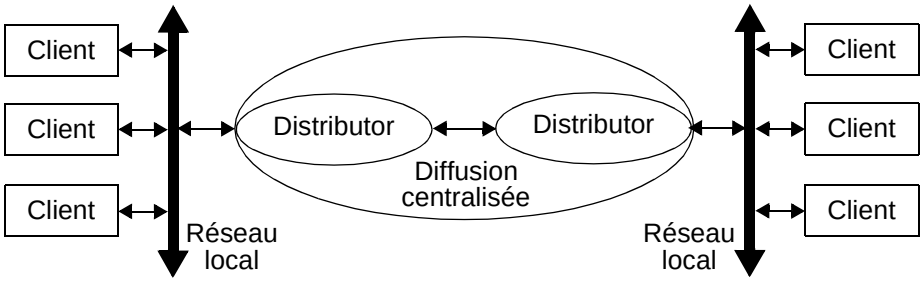


Figure 4
Schéma d'exécution de l'outil CORONA

Par exemple, l'outil CORONA [Hall 1996] offre une infrastructure dédiée à l'activité de communication qui repose sur un schéma d'exécution dynamique. Cette infrastructure met en œuvre différents modes de collaboration pour favoriser la communication de groupe notamment par le biais de groupes de diffusion. Le but est d'assurer un service communication fiable à grande échelle. Ainsi, comme le montre la Figure 4, lorsque la communication est locale à un site (par exemple un réseau intranet), les processus sont entièrement répartis : le système établit la communication en se basant sur le protocole multicast. Par contre, dès que la communication est établie entre des processus localisés sur deux sites distants, le schéma d'exécution devient hybride : localement à chaque site, les processus sont toujours répartis et la communication entre les deux sites est établie par le biais de processus dédiés appelés *Distributor*.

Nous avons présenté deux classes d'outils, les boîtes à outils et les plates-formes, pour le développement de collecticiels. Dans la suite de ce chapitre, nous examinons des outils de chacune de ces deux classes selon notre grille d'analyse.

3. Grille d'analyse

Pour étudier les outils de développement des collecticiels selon la démarche adoptée, il convient de traduire les quatre dimensions de notre grille en des termes logiciels.

3.1. ACTION COLLECTIVE ET INDIVIDUELLE

Il est difficile de définir des critères d'analyse concernant le support de l'action individuelle, car les fonctions logicielles correspondantes dépendent beaucoup du domaine d'application et de la nature du collecticiel. Aussi les outils de développement de par leur caractère générique n'offrent pas de fonctions réutilisables dédiées à l'action individuelle. Par contre, il semble important de souligner que cela ne remet pas en cause l'idée qu'un outil doit pouvoir faire la distinction action individuelle et action collective. Ce point est essentiel pour considérer l'observabilité des actions dans son ensemble.

Couverture fonctionnelle et modèle du trèfle

Le point de vue retenu dans l'étude des différents outils pour le développement des collecticiels porte donc principalement sur l'action collective. Plus précisément, nous nous attachons au traitement des trois aspects de l'action collective : production, communication et coordination. Nous définissons la couverture fonctionnelle de l'outil par sa capacité à couvrir les trois aspects : production, communication et coordination.

Niveaux d'abstraction d'un outil

Pour analyser l'étendue de la couverture fonctionnelle des outils, nous définissons trois niveaux d'abstraction des fonctions offertes par l'outil :

- *Mécanismes logiciels (bas niveau)* : il s'agit de fonctionnalités de bas niveau d'abstraction, centrées sur le système et exprimées uniquement en termes algorithmiques. Par exemple, dans le cas d'un éditeur partagé, un mécanisme de fusion de versions de document est utilisé. En effet, il est nécessaire de fusionner les différentes modifications réalisées par les utilisateurs pour ne produire qu'un seul document, les modifications apportées étant considérées comme autant de versions du document. A ce niveau, le point de vue est principalement technique.

- *Services pour la collaboration (niveau intermédiaire)* : il s'agit de services, s'appuyant sur les mécanismes logiciels du niveau d'abstraction inférieur, pour offrir des fonctionnalités adaptées à l'action collective. Par exemple, un service de gestion de la concurrence repose sur les différents algorithmes de contrôle de la concurrence (optimiste, pessimiste ou autre) et choisira l'algorithme adapté au contexte. Par exemple, si une action critique doit être effectuée, ce service de gestion de concurrence choisira un algorithme

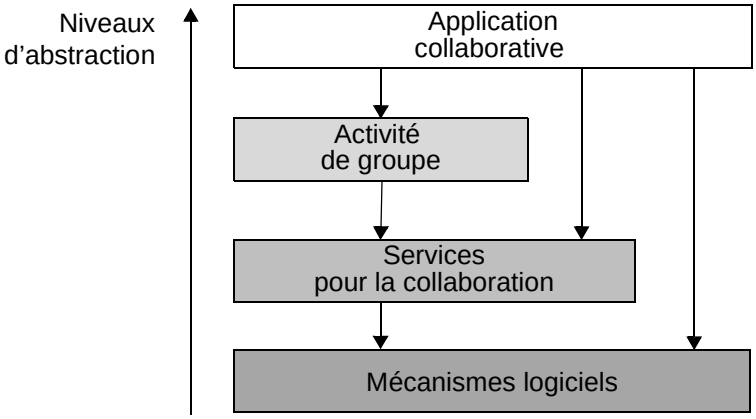


Figure 5
Architecture en couche et niveaux d'abstraction

de contrôle de concurrence pessimiste afin que cette action s'exécute en toute sécurité, c'est-à-dire en évitant que le système tombe dans un état incohérent : toutes les autres actions sont suspendues tant que cette action en cours n'est pas terminée (algorithme préventif). Pour des actions qui ne requièrent pas un haut niveau de sécurité, le service peut alors utiliser un algorithme de gestion de concurrence optimiste qui autorise toutes les actions à être exécutées simultanément : cet algorithme prévoit alors de corriger les instabilités potentielles du système si celui-ci se retrouve dans un état incohérent (algorithme curatif). En d'autres termes, un service doit être en mesure de choisir les bons algorithmes selon les contextes d'utilisation. Dewan [Dewan 2001] a dressé une liste de services propres aux collecticiels de ce niveau d'abstraction, tels que le "défaire/refaire" ou la fusion de données.

- *Activité de groupe (haut niveau)* : il s'agit de primitives dédiées à l'action collective indépendamment des problèmes systèmes. Par exemple : créer un groupe, joindre un groupe, créer une zone de communication privée, communiquer dans une zone dédiée ou modifier un document partagé. Ces primitives reposent sur les services pour la collaboration. Par exemple, pour joindre un groupe, il est nécessaire de disposer d'un service de gestion de session, de même pour modifier un document partagé, il est nécessaire de disposer d'un service de contrôle de la concurrence.

A la Figure 5, nous représentons les trois niveaux d'abstraction des fonctions offertes par un outil selon une architecture en couche. Les services pour la collaboration sont construits à partir de mécanismes logiciels tandis que les primitives dédiées à l'activité de groupe reposent sur des services pour la collaboration. Comme le souligne le schéma de la Figure 5, une application accède aux fonctions offertes par un outil, ces fonctions pouvant être de niveaux d'abstraction différents. Certes, plus les

fonctions offertes par l'outil sont de haut niveau, plus le développement du programme client est simplifié [Nigay 1994].

De plus, les fonctions de haut niveau d'abstraction définissent une couche logicielle entre le programme client et les mécanismes logiciels. Cette couche sert d'interface logicielle et rend l'application indépendante des mécanismes logiciels. Par exemple, une application collaborative peut être indépendante des infrastructures de communication sous-jacentes.

Enfin, un haut niveau d'abstraction permet de masquer aux développeurs les détails techniques d'implémentation pour favoriser un développement centré sur l'activité de groupe, les aspects collaboratifs de l'application à développer.

Lors de l'analyse des outils existants, **le niveau d'abstraction de référence retenu est celui centré sur l'activité de groupe**.

3.2. RESSOURCES DU CONTEXTE

Selon la dimension intitulée "ressources du contexte", nous distinguons les ressources individuelles et les ressources collectives. Cette distinction véhicule la notion de propriétaire : les ressources sont-elles la propriété exclusive d'un seul utilisateur ou appartiennent-elles à un groupe d'utilisateurs? Cette dimension se traduit au niveau logiciel par la capacité de l'outil à permettre d'associer un seul propriétaire ou un groupe de propriétaires à des données. Par exemple, le système Unix, que l'on peut qualifier de collecticiel puisqu'il gère des processus concurrents, associe un propriétaire unique à chaque fichier, un fichier étant une ressource du contexte.

3.3. OBSERVABILITÉ DES ACTIONS ET DES RESSOURCES DU CONTEXTE

Selon ces deux dimensions de la grille, nous identifions deux formes d'observabilité : observabilité des actions/ressources individuelles et observabilité des actions/ressources collectives. En des termes architecturaux, nous avons observé que ces formes d'observabilité se traduisent par des flots de données entre composants logiciels. Les données échangées traduisent les différents états d'une action (état initial, état final et états intermédiaires de l'action) ou correspondent à une ressource du contexte. Du point de vue logiciel, nous étudions ici si l'outil offre des fonctions pour réaliser ou aider à réaliser ces flots de données. La réalisation logicielle d'un flot de données entre deux composants source et destinataire implique :

1 **Disponibilité des données source** :
Nous étudions si l'outil offre ou favorise la mise en œuvre des mécanismes de droit d'accès et de filtrage des données.

* Droits d'accès à des données privées et collectives. Les droits d'accès sont à mettre en relation avec la notion de propriétaire liée à une ressource. Par exemple sous Unix, des droits d'accès en

écriture, en lecture ou en exécution sont définissables pour un fichier donné par le propriétaire.

* Filtrage des données. Par exemple, un courrier électronique crypté est une donnée filtrée : une partie des informations est observable comme le destinaire du courrier, l'autre est masquée comme le contenu du courrier.

2 **Diffusion des données** :
Nous étudions si l'outil aide le développeur à réaliser la propagation des données. Par exemple un mécanisme de diffusion d'événements en multicast, des variables de couplage ou encore un système de gestion de contraintes sont des exemples de services pour la diffusion, classiquement utilisés pour la rétroaction de groupe (observabilité des actions collectives).

3.4. SYNTHÈSE

En synthèse, la grille que nous appliquons à l'analyse des outils existants est la suivante :

- **Actions collectives** :
  * Niveau d'abstraction des services offerts par l'outil : niveau activité de groupe, niveau services pour la collaboration et niveau mécanismes logiciels.
  * Couverture fonctionnelle de l'outil selon les trois espaces du modèle du trèfle : production, communication et coordination.
- **Ressources du contexte** : Propriétaire d'une donnée.
- **Observabilité des actions collectives et des ressources :**
  * Droits d'accès et filtrage des données.
  * Diffusion des données.

4. Les outils existants

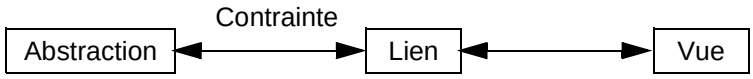
Dans cette partie, nous analysons plusieurs outils de développement de collecticiels selon la grille d'analyse de la partie précédente. Notre objectif n'est pas une revue exhaustive des outils existants, mais la présentation d'outils différents voire complémentaires selon notre grille.

Pour fixer un ordre de présentation, nous avons classé les outils selon le niveau d'abstraction des services offerts pour la coordination, en commençant par les services de plus bas niveau. Nous avons choisi de centrer la classification des outils sur les aspects de coordination, car la plupart des outils offrent des fonctions pour la coordination, constat que nous ne pouvons pas faire pour la production ou la communication. Néanmoins le dernier outil analysé est Corona, un outil dédié à la communication.

Pour l'ensemble des outils étudiés, les architectures présentées sont des architectures d'implémentation, à distinguer des architectures conceptuelles étudiées aux Chapitre III et Chapitre IV.

4.1. RENDEZ-VOUS

(a) Modèle ALV (Abstraction-Link-View)



(b) Architecture d'implémentation d'applications avec Rendez-Vous

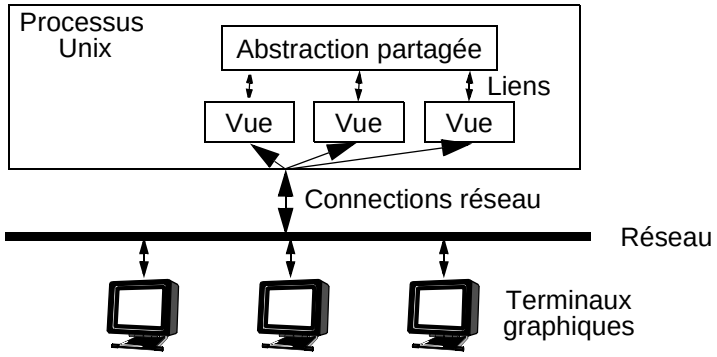


Figure 6
Architecture d'implémentation de Rendez-Vous
(a) Modèle ALV,
(b) Exemple d'architecture d'implémentation d'une application conçue avec l'infrastructure Rendez-Vous.

Rendez-vous [Hill 1994] est une infrastructure dédiée au développement de collecticiels graphiques synchrones. Elle repose sur une version étendue de l'interpréteur de langage Common Lisp et CLOS [Steele 1990], dont l'extension graphique permet la création d'interfaces sur X-Windows. Un des objectifs est de fournir une structure pour l'élaboration d'interfaces multi-ultilisateur partagées.

Comme le montre la Figure 6 (a), le développement d'applications avec cette infrastructure repose sur le modèle d'architecture ALV (*Abstraction-Link-View*, présenté au Chapitre III) qui distingue la présentation (*View*)

du Noyau Fonctionnel (*Abstraction*) séparés par un adaptateur (*Lien*). Tout comme le modèle de Arch avec son mécanisme de branches, il est possible de définir plusieurs vues pour une abstraction partagée. Le lien assure la cohérence entre les données et leurs présentations en fonction d'un ensemble de contraintes. Ces contraintes assurent que la représentation d'une donnée dans une vue est cohérente avec la représentation abstraite et permet ainsi de répercuter toutes modifications apportées à une vue sur l'abstraction. Ce mécanisme de contraintes est donc utilisé pour assurer la communication entre les vues et l'abstraction partagée.

L'implémentation d'une application avec Rendez-Vous repose sur un schéma d'exécution centralisé, comme le montre la Figure 6 (b). Notons néanmoins qu'avec le modèle d'architecture ALV, l'implémentation est réalisable selon un schéma totalement répliqué. Cette infrastructure offre de plus un service de gestion de sessions avec attribution de rôles pour chaque utilisateur. De plus, grâce au schéma d'exécution centralisé mettant en œuvre une abstraction partagée, l'application ainsi développée permet aux utilisateurs "tardifs" de rejoindre une session en cours.

Actions et couverture fonctionnelle

Par rapport à la première dimension de notre grille, nous constatons que les services proposés par Rendez-Vous sont d'assez bas niveau d'abstraction et le support à l'action collective est incomplet. En effet, les services sont centrés uniquement sur la coordination par le biais d'un système de gestion de sessions et d'attribution de rôles. L'action collective est considérée à gros grain : la majeure partie du travail de développement doit donc être assurée par le programmeur. Cependant, cette infrastructure offre des services de contrôle de la concurrence et un service permettant l'accès à une session en cours pour les utilisateurs tardifs. L'infrastructure Rendez-Vous est fortement axée sur l'aspect interface de par le modèle sous-jacent ALV, puisque le mécanisme central est d'avoir plusieurs vues pour une seule abstraction partagée.

Ressources du contexte

L'outil Rendez-Vous repose sur le modèle ALV. Par conséquent, il hérite des propriétés de ce modèle, notamment, celles relatives aux ressources du contexte. Cet outil rend explicites les ressources individuelles et les ressources collectives. Les ressources collectives sont les données partagées gérées par le serveur et sont la propriété de tous les utilisateurs. Les ressources individuelles sont les données relevant de la vue et sont la propriété exclusive de chaque utilisateur.

Observabilité des actions et des ressources

Cet outil met en œuvre des moyens grâce au mécanisme de liens pour propager les modifications de données. Il s'agit d'une forme de couplage *WYSIWIS* puisque le lien est défini par un ensemble de règles écrites en Lisp liant les données gérées par la vue avec les données partagées

correspondantes du serveur (abstraction partagée). En effet, tout changement apporté à la vue est automatiquement répercuté sur l'abstraction partagée et réciproquement.

La disponibilité des données repose sur un mécanisme très simple : les données partagées sont totalement accessibles. Les données privées relevant de la vue et qui ne sont pas associées à une donnée partagée sont privées et totalement inaccessibles pour les autres utilisateurs.

4.2. NSTP

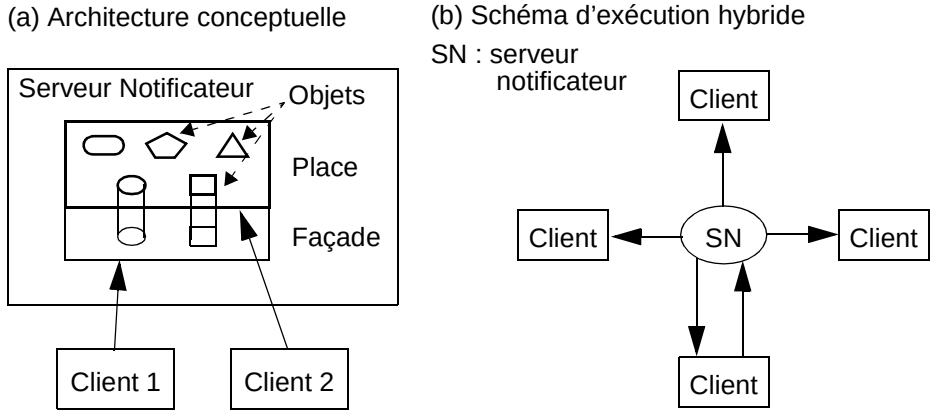


Figure 7
Architecture d'implémentation de NSTP
(a) architecture conceptuelle, (b) schéma d'exécution hybride.

NSTP (*Notification Service Transfer Protocol*) [Patterson 1996] est une infrastructure pour la réalisation de collecticiels de type client-serveur reposant principalement sur un service de notification de changements d'état partagé. NSTP a été développé en Java et offre un support complet pour la mise en œuvre du serveur et le développement de l'application cliente.

Comme le montre la Figure 7 (a), l'architecture conceptuelle repose sur quatre concepts : le concept de place, de façade, d'objets représentés sous forme de figures géométriques, et le serveur lui-même. Le concept de place (*Place*) a été utilisé pour désigner une session, métaphore similaire au concept de la pièce (*Room*). Une place représente l'état partagé par plusieurs clients, celle-ci étant constituée d'un ensemble d'objets partagés (*Things*). Ainsi, pour joindre une session, un client rejoint une place. De plus, un client peut joindre plusieurs places à la fois. Dès lors, il est possible de manipuler les différents objets en fonction des droits d'accès qui leur sont assignés. Enfin, le concept de façade (*Facade*) est la vue externe d'une place avec double fonction : premièrement, une façade permet de manipuler des objets rendus publics (formes géométriques dupliquées) sans forcément entrer dans une place, deuxièmement, c'est le point de passage obligatoire pour entrer dans une place. Pour rejoindre une place, c'est-à-dire une session, l'infrastructure repose sur le principe de requêtes : un participant émet une requête pour indiquer au groupe son souhait de rejoindre la session en cours (*calling-in*). Tous les clients d'une

session sont automatiquement avertis d'un changement d'état. Par le biais du service de notification, l'infrastructure offre donc un support à la rétroaction de groupe (*group awareness*) en indiquant tous les changements d'état.

L'architecture d'une application conçue à l'aide de cette infrastructure repose, comme le montre la Figure 7 (b), sur un schéma d'exécution hybride : des clients répartis pour un état partagé centralisé sur le serveur (SN à la Figure 7 (b)). Lors d'un changement d'état, le serveur a la charge de diffuser l'information aux clients. Enfin, le serveur doit gérer l'homogénéité et la cohérence des données partagées.

Actions et couverture fonctionnelle

L'outil NSTP distingue l'action individuelle de l'action collective et offre un support à l'action collective à différents niveaux d'abstraction.

Pour la coordination, le support est minimal mais néanmoins de haut niveau d'abstraction. En effet, la notion de groupe est restreinte à la notion de place : soit l'utilisateur est dans la place, soit il est en dehors. L'infrastructure offre des primitives pour créer, détruire, joindre ou quitter une place. A cela, s'ajoute la possibilité d'appartenir à plusieurs places. Par contre, il n'est pas possible d'avoir une structure de groupe au sein d'une même place, comme dans toutes organisations (par exemple, un laboratoire de recherche est constitué de plusieurs équipes). En ce sens, le support de l'aspect coordination est minimal.

Pour la production, l'infrastructure repose sur le concept d'objet décrit par le couple `<attribut, valeur>`. Il est possible de créer, détruire et de modifier ces objets en fonction des droits accès associés aux objets. Les droits sont définis de manière fixe à la création de l'objet. Aucune modification des droits d'accès n'est donc possible en cours de session. De plus les objets ne peuvent appartenir qu'à un créateur unique et non à un groupe d'utilisateurs. Néanmoins certains objets peuvent ne pas avoir de propriétaire : des mécanismes système pour verrouiller (*locking*) l'accès à ces objets sont alors fournis.

Enfin, la communication se limite au mécanisme système de diffusion de messages entre tous les clients. Aucun support explicite n'est fourni et c'est donc au développeur de mettre en place la structure de communication.

Outre ces fonctionnalités, l'infrastructure offre des services logiciels tels que l'arrivée dans une session en cours de route (*latecomers*) s'appuyant sur le fait que le serveur mémorise l'ensemble des données partagées. De plus, NSTP intègre un service de contrôle de cohérence de ces données.

Ressources du contexte

L'outil NSTP permet de distinguer les ressources individuelles et les ressources collectives en définissant la notion de propriété d'un objet. Un objet disposant d'un propriétaire est une ressource individuelle. Un objet

sans propriétaire est une ressource collective, commune à tous les utilisateurs. Par contre, comme nous venons de l'expliquer, un objet ne peut pas avoir plusieurs propriétaires.

Observabilité des actions et des ressources

Pour l'observabilité des actions et des ressources, cet outil met en œuvre un service de notification qui a la charge de diffuser l'information les changements d'états du système. Les changements d'états correspondent à des modifications des objets privés ou partagés.

La disponibilité des données repose sur un mécanisme de droits d'accès. Toutefois ce mécanisme est limité puisqu'il n'est pas possible de modifier les droits une fois l'objet créé. De plus, l'outil ne propose aucune solution afin de mettre en œuvre une politique de filtrage.

4.3. GROUPKIT

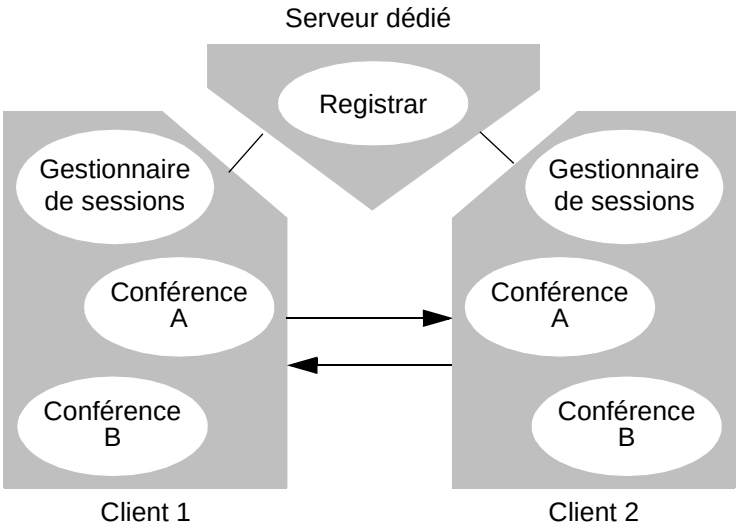


Figure 8 Architecture d'implémentation d'une application conçue avec GroupKit

GroupKit [Roseman 1992][Roseman 1996a] est une boîte à outils pour la programmation de collecticiels écrite dans une version étendue du langage interprété Tcl/Tk [Ousterhout 1994], incluant notamment le protocole UDP/Multicast, Tcl/DP. Du point de vue du programmeur, les applications développées avec GroupKit sont écrites en Tcl/Tk. Actuellement, l'interpréteur Tcl/Tk a été porté sur un très grand nombre de systèmes, ce qui permet aux applications ainsi développées d'être exécutées sur Unix, Macintosh (Classic et X), Microsoft Windows, et depuis peu sur PocketPC.

Cette boîte à outils est en fait une infrastructure puisqu'à l'exécution, comme le montre la Figure 8, l'architecture d'implémentation d'une application développée avec GrouKit est composée de trois types processus, le "*Registrar*", le gestionnaire de sessions et la conférence :

- Le processus "*Registrar*", instance unique de ce type, est centralisé sur un serveur dédié et se charge de maintenir une liste des conférences et des utilisateurs.

- Le processus de gestion de sessions (*session manager*), dont l'instance est répliquée au niveau de tous les clients, permet de créer et de détruire des conférences, ainsi que d'accéder aux différentes conférences. De plus, ce processus fait état de tous les changements dans une conférence auprès du "*Registrar*" qui se charge de propager ces changements aux autres instances du gestionnaire de sessions. L'outil est extensible, d'autres gestionnaires de sessions peuvent être développés par le programmeur.

- Le processus de conférence, instance répliquée au niveau de tous les clients, **est l'application collaborative écrite par le développeur**. Ces instances communiquent via un mécanisme RPC (*Remote Procedure Call*) et de propagation d'événements, et partagent des objets par le biais d'un environnement partagé (un ensemble de couples `<attribut, valeur>`).

De plus, GroupKit fournit des objets graphiques collaboratifs tels que les télépointeurs, les vues radar et les vues déformantes multi-utilisateurs.

Actions et couverture fonctionnelle

Selon la première dimension de notre grille, GroupKit ne distingue pas de façon explicite les actions individuelles des actions collectives. Néanmoins, Groupkit offre un support à l'action collective dédiée à la coordination, relativement de haut niveau, par le biais d'un système de gestion de conférences (i.e. les applications collaboratives) et du processus "*Registrar*". Des primitives sont disponibles pour gérer les conférences (créer, détruire, joindre et quitter une conférence) et pour gérer les droits d'accès. Ainsi, tous les changements au sein d'une conférence ou l'apparition d'une conférence sont automatiquement transmis à l'ensemble des clients par le biais du système de propagation de messages. Le processus "Registrar" maintient une base de donnée partagée des conférences, contenant l'ensemble des informations relatives aux utilisateurs et aux sessions.

L'action collective dédiée à la production est rendue explicite à travers cet environnement partagé. Cet environnement, un service système clé de l'outil GroupKit, est chargé de gérer un ensemble d'objets répliqués entre toutes les instances clientes. De plus, cet environnement dispose de services de gestion de la concurrence qui par défaut ne sont pas activés. Ainsi, lorsque des opérations sensibles sont réalisées, il est possible de sélectionner une instance répliquée de la conférence en cours pour traiter dans le bon ordre les différentes opérations (*serialization*).

Enfin, l'action collective dédiée à la communication repose sur des mécanismes logiciels d'appels de fonction *RPC* (*Remote Procedure Call*) et de propagation d'événements reposant sur le protocole Multicast.

Ressources du contexte

GroupKit considère les ressources collectives et individuelles de façon implicite. Les ressources individuelles sont les données gérées localement par un processus client. Les données gérées par l'environnement partagé sont, comme son nom l'indique, partagées par tous les utilisateurs : il s'agit alors de ressources collectives. Le service permet d'associer un utilisateur à une donnée partagée, utilisateur que l'on peut assimiler à un propriétaire.

Observabilité des actions et des ressources

Par le biais de l'environnement partagé, GroupKit offre un moyen pour diffuser les données (observabilité des ressources) et les changements d'état (observabilité des actions). Cet environnement contient l'ensemble des données partagées. De plus, cet environnement partagé participe à la mise en œuvre du couplage de l'interaction. Par exemple, le *widget* de pointeur partagé repose entièrement sur l'environnement partagé.

D'autre part, les mécanismes de communication par événements et appels de fonction *RPC* offrent une base minimale supplémentaire pour diffuser les données.

Enfin, l'outil GroupKit ne propose aucun moyen pour protéger les ressources du contexte puisque celles-ci sont totalement accessibles par le biais de l'environnement. Toutefois, les ressources individuelles locales au processus d'un utilisateur sont totalement privées et inaccessibles.

4.4. INTERMEZZO

Intermezzo [Edwards 1996a][Edwards 1996b] est une infrastructure reposant sur le modèle client-serveur et est écrit en C++ et Python. Intermezzo offre un support flexible pour la coordination à base de spécification de règles. La partie serveur contient différents services : stockage persistant des données de coordination, service de coordination, service de rendez-vous (*session management*), service de notification, chargement dynamique de code et exécution à distance (comparable au *RMI*, *Remote Method Invocation*, dans l'environnement Java). De plus, le serveur se charge de vérifier l'homogénéité des données partagées stockées sur le serveur mais aussi des copies stockées dans le cache de tous les clients.

Par le biais de règles, il est possible de définir les droits d'accès sur les différents objets manipulés. Cette infrastructure est basée sur le mécanisme de liste de contrôle d'accès (*ACL*, *Acces Control Lists*). La gestion de sessions repose aussi sur le même type de règles étant donné qu'une session est assimilée à un type d'objet particulier. Une règle est associée à un ensemble de ressources. Pour chaque attribut caractérisant

une ressource, il convient de spécifier un droit d'accès (lecture uniquement, écriture, etc). Un rôle est l'association d'un identifiant avec un ensemble de règles.

Actions et couverture fonctionnelle

Selon la première dimension de notre grille, Intermezzo met en place un support très évolué à l'action collective dédiée à la coordination. Les règles régissant la coordination entre utilisateurs sont spécifiées dans le langage Python sous forme de scripts chargés dynamiquement. Ces scripts définissent les rôles et les droits accordés.

Pour la production, Intermezzo dispose de deux services, un pour la gestion de la concurrence, l'autre pour le contrôle de l'homogénéité des données.

Le support de l'action collective dédiée à la communication est délégué au développeur chargé de mettre en œuvre une solution adaptée.

Ressources du contexte

L'outil Intermezzo utilise explicitement le terme de ressources pour identifier l'ensemble des données manipulées, qu'elles soient privées (ressources individuelles) ou partagées (ressources collectives). Cependant, une ressource ne peut disposer que d'un seul propriétaire et une ressource est partagée que si le propriétaire la rend observable et accessible.

Observabilité des actions et des ressources

Les données rendues accessibles sont protégées par des droits d'accès modifiables uniquement par le propriétaire. Grâce à la définition de droits d'accès, un propriétaire peut donc spécifier quel utilisateur est autorisé à consulter ou modifier une ressource. Le système offre donc un système de filtrage et de protection des données.

L'outil Intermezzo dispose aussi d'un service de notification, à base de propagation d'événements et de *trigger*, pour diffuser les changements d'état et les données partagées modifiées. Enfin, cet outil favorise la rétroaction de groupe en maintenant une base d'informations du type "qui manipule tel objet avec quelle application" sur toutes les activités en cours.

4.5. COCA

COCA [Li 1998][Li 1999] offre une infrastructure pour le développement d'applications collaboratives centrée sur la coordination. Cette infrastructure est écrite dans le langage Java et repose entièrement sur le protocole Multicast pour assurer la communication entre les différents clients et processus dédiés. Un interpréteur permet de définir des règles de coordination, spécifiées dans un langage de logique du premier ordre proche de Prolog.

Comme le montre la Figure 9, l'architecture d'implémentation d'une application conçue avec COCA est constituée de deux bus de

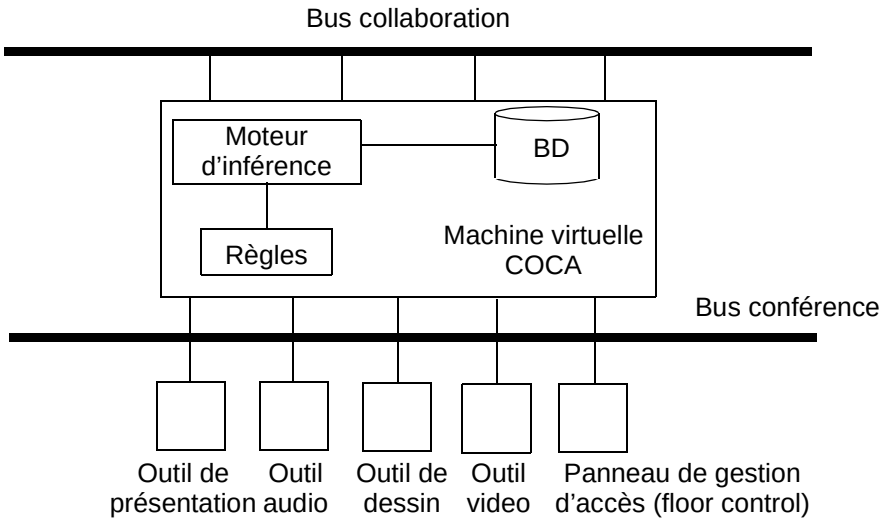


Figure 9
Architecture d'implémentation d'applications conçues avec COCA

communication, c'est-à-dire deux groupes multicast : un bus dédié à la collaboration (bus collaboration à la Figure 9) qui assure la liaison entre tous les utilisateurs et un bus assurant l'accès aux différents outils (bus conférence à la Figure 9) tels que les outils de présentation (*slideshow*) ou de dessin (*whiteboard*). Le système de gestion de la collaboration est un processus dédié, la machine virtuelle COCA, processus répliqué au niveau de tous les clients. La machine virtuelle COCA se charge de réguler les accès et les échanges de données entre tous les utilisateurs via les outils utilisés. Pour cela, elle intègre un moteur d'inférence qui analyse dynamiquement un ensemble de règles décrivant des rôles et les droits d'utilisation des différents outils. De plus, cette machine virtuelle inclut une base de données, sorte de cache contenant une vue éphémère de l'état de la collaboration pour la synchronisation entre les clients. Enfin, tous les outils sont exécutés localement à chaque client et communiquent avec la machine virtuelle via le bus dédié à la conférence. L'implémentation actuelle permet de réutiliser des outils conçus au départ pour des applications mono-utilisateur : il s'agit alors de l'approche par collaboration transparente expliquée au paragraphe 2.1.

Pour réguler la coordination entre les différents participants d'une conférence, les règles spécifient, en fonction des différents rôles, comment se réalisent les échanges de messages liés à une action au niveau de l'interface homme-machine. Ainsi la coordination est définie indépendamment de la sémantique de l'outil (Noyau Fonctionnel), puisque sa spécification ne s'appuie que sur des actions-utilisateur au niveau de l'interface graphique. De plus, le langage utilisé pour spécifier ces règles est constitué d'opérateurs, comme les transactions, qui facilitent la programmation :

- de services de gestion d'accès aux outils en spécifiant pour chaque rôle les droits accordés (*floor control et access control*),

- de services de gestion de sessions par la définition d'une règle de conférence (*session manager*) et,

- de services de gestion d'accès concurrent (*concurrency control*). Pour ce dernier service, la programmation est de très bas niveau puisque COCA offre des prédicats pour définir des zones de transactions (`begin-transaction` et `end-transaction`) et des prédicats pour le verouillage d'objets (`lock` et `unlock`).

Actions et couverture fonctionnelle

Tout comme Intermezzo, COCA se concentre uniquement sur l'aspect coordination de l'activité de groupe. L'objectif affiché est de proposer une plate-forme capable de contrôler la coordination entre utilisateurs indépendamment des outils utilisés [Li 1999]. La coordination est régie par un ensemble de règles programmées dans un langage déclaratif de logique du premier ordre. Avec ce langage, il est possible de spécifier précisément les rôles joués par les utilisateurs, de définir les règles d'utilisation des différents outils et les droits d'accès aux objets partagés. Ainsi, le support de l'action collective dédiée à la coordination est défini à haut niveau d'abstraction.

Par contre, cet outil ne traite pas de l'activité de groupe liée à la communication et à la production. Ces aspects de l'activité de groupe sont délégués aux outils manipulés comme les outils audio/video ou l'outil de dessin (*whiteboard*).

Ressources du contexte

La plate-forme COCA raisonne à très gros grain puisqu'une ressource collective est assimilée à un outil. Toutefois, nous avons évoqué l'existence d'objets partagés. Il s'agit d'objets au sens de la programmation orientée objet du type C++.

Observabilité des actions et des ressources

Il est difficile de parler d'observabilité des ressources du contexte sachant que les ressources sont des outils. Néanmoins, le modèle de coordination met en œuvre un système de droits d'accès et d'utilisation que l'on peut apparenter, par extension, à une forme de filtrage lié à l'observabilité.

Enfin, la plate-forme COCA met en œuvre des services systèmes à base de communication multicast et de gestionnaire d'événements (*event handling*) chargés de diffuser les informations sur les actions en cours.

4.6. DARE

DARE (Activités Distribuées dans un Environnement Réflexif) [Bourguin 2000] traite des aspects proches de ceux des infrastructures COCA et Intermezzo en proposant un modèle de coordination très élaboré et séparé des applications. Ce modèle repose sur des résultats issus de la théorie de l'activité. A la différence des deux infrastructures précédentes qui expriment la coordination selon un langage dédié (python et une forme de prolog), l'infrastructure DARE s'appuie sur la réflexivité des

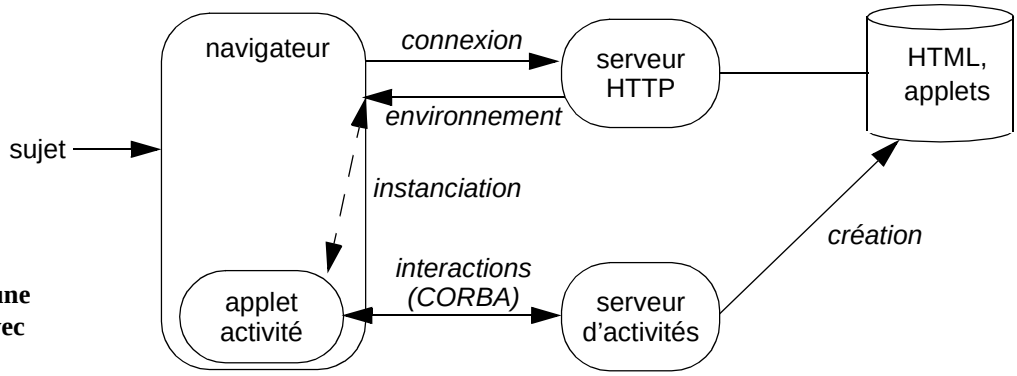


Figure 10
Architecture d'implémentation d'une application conçue avec DARE

langages à objet (Smalltalk, Java et Corba) en considérant l'aspect coordination sous forme de hiérarchie de méta-classes, une classe étant une instance d'une méta-classe. Cette approche est issue des recherches sur l'implémentation ouverte (*Open Implementation*, [Kiczales 1992] [Kiczales 1996] [Kiczales 1997]). L'infrastructure est développée en Java et Smalltalk tout en se reposant sur le protocole HTTP et la plate-forme CORBA.

La hiérarchie de méta-classes distingue les utilisateurs, les rôles et micro-rôles, les tâches et les activités. Les outils sont des instances de la méta-classe activité, comme le montre la Figure 10, et correspondent à des applets accessibles, via la plate-forme CORBA, sur un serveur d'activité. L'utilisateur manipule l'outil en fonction de son rôle accordé et du micro-rôle relatif à l'emploi de l'outil. Les connexions sont réalisées par le protocole HTTP via un serveur HTTP qui gère le chargement d'applets et de pages HTML stockées localement au serveur. L'outil mis à disposition peut être n'importe quelle applet, qu'elle soit conçue pour une utilisation multi-utilisateur ou non. Ce mécanisme autorise donc l'approche par collaboration transparente expliquée au paragraphe 2.1, puisque le système gère de manière externe à l'application la coordination. De plus il est possible de définir des scripts d'utilisation de l'interface de l'applet en définissant des droits d'exécution associés aux différentes méthodes. Par exemple, il est possible de définir un droit d'exécution d'une méthode d'un bouton graphique. Ce mécanisme est possible grâce à la réflexivité du langage Java.

Actions et couverture fonctionnelle

L'outil DARE, tout comme Intermezzo et COCA, met en œuvre un support à l'action collective de haut niveau, dédié uniquement à la coordination. Cet outil repose sur un modèle de l'activité développé sous forme de métaclasses Smalltalk. Le niveau de granularité est donc celui de l'activité. Une activité implique un ensemble de participants et est caractérisée par un but à atteindre (objet de la tâche commune). La réalisation de ce but repose sur l'utilisation d'outils mis à la disposition des participants. L'outil DARE définit des primitives qui permettent notamment à un participant de joindre ou quitter une activité en cours. La

coordination au sein de ces outils est régie en fonction des rôles joués par les participants.

Pour définir une forme d'activité, le développeur doit programmer différentes classes (activités, rôles et outils) résultant d'un instanciation des métaclasses définies au sein de l'outil DARE. A l'exécution, un moteur de coordination utilise les classes programmées par le développeur afin de réguler la coordination entre les différentes activités.

Ressources du contexte et observabilité

L'outil DARE ne rend pas explicites les éléments relatifs aux ressources du contexte, à l'observabilité des actions et des ressources.

4.7. CORONA

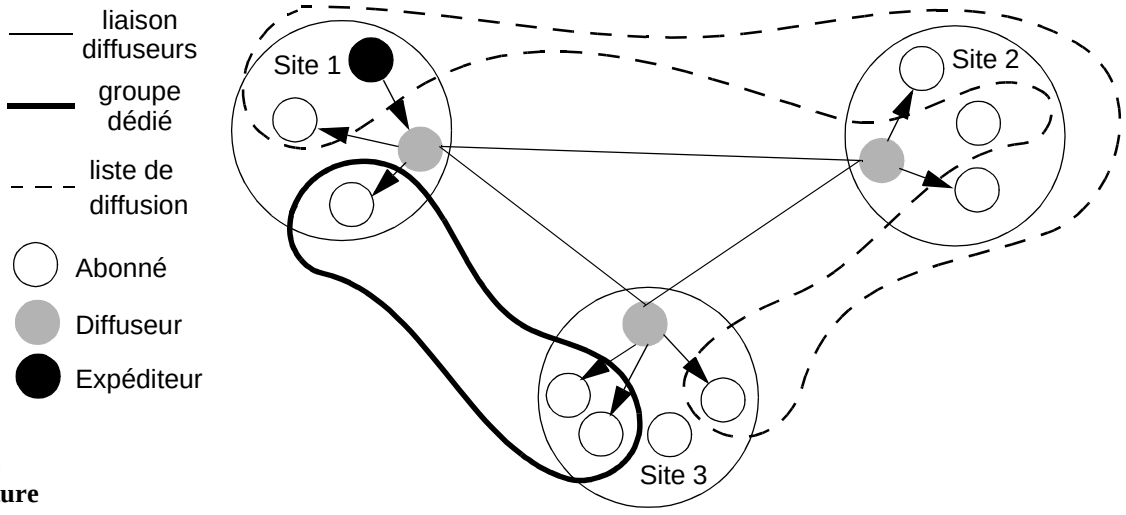


Figure 11 Architecture d'implémentation de CORONA

Corona [Hall 1996] est une infrastructure dédiée à la communication uniquement. L'objectif de cette plate-forme est d'offrir des services assurant l'acheminement fiable de l'information à destination, le passage à l'échelle et la conscience de groupe réduite à la connaissance des participants. Ces services s'articulent autour de deux paradigmes de communication : le paradigme "s'abonner/publier" (*publish/subscribe*) et le paradigme de groupe (*peer group*). Cette infrastructure est implémentée en Java et C++ ; les clients sont écrits en Java et les diffuseurs en C++.

Pour le premier service, reposant sur le paradigme "s'abonner/publier", la communication est anonyme : un utilisateur envoie un message (*publisher*) à l'ensemble des abonnés (*subscribers*) membres d'un groupe. Cette approche est comparable aux listes de diffusion (*mailing-lists*) dont les membres sont inconnus. Ainsi la conscience de groupe est très limitée : en effet l'expéditeur à l'origine du message ne connaît pas les

destinataires. Pour assurer la communication à grande échelle, l'infrastructure diffuse les messages localement à un site par le biais du protocole multicast et à tous les autres sites par le biais de serveurs dédiés appelés "diffuseurs" (*distributor*). Comme le montre la Figure 11, un expéditeur (cercle noir) envoie un message sur une liste de diffusion (délimitée par un trait en pointillé) et donc à tous les abonnés de cette liste (cercles vides inclus dans la liste de diffusion). La transmission du message est relayée par un diffuseur (cercle gris) qui se charge de transmettre l'information aux autres diffuseurs, qui, à leur tour, diffusent l'information localement. Cette infrastructure permet donc le passage à l'échelle entre différents sites, mais limite considérablement la conscience de groupe étant donné que les diffuseurs n'ont aucune connaissance des abonnés localisés sur les autres sites. En effet, un diffuseur dispose de la liste de ses abonnés qu'il gère localement, et de la liste des autres diffuseurs. Néanmoins un diffuseur ne connaît pas les abonnés que gèrent les autres diffuseurs. Ainsi lorsqu'un expéditeur envoie un message, celui-ci est distribué à tous les diffuseurs qui se chargent à leur tour de l'acheminer à leurs abonnés. De plus, pour faciliter ce passage à l'échelle, ce service gère la perte d'information en cours de route.

Le second service repose sur le paradigme de groupe : tous les membres d'un groupe se connaissent. Ce service favorise donc la conscience de groupe car l'expéditeur d'un message connaît les membres du groupe auxquels il s'adresse. De plus, cette notion de groupe s'accompagne de la notion de rôle. Deux rôles sont possibles. Le premier est le rôle principal et autorise toutes les opérations possibles. Le second est celui de l'observateur qui peut seulement consulter les messages échangés, sans pouvoir en modifier le contenu ou envoyer de nouveaux messages. Dans ce mode, les diffuseurs connaissent les différents membres d'un groupe et à quels diffuseurs ils sont rattachés. Ce mécanisme permet d'acheminer les messages directement aux diffuseurs sans avoir à propager le message comme pour le mode "s'abonner/publier". Néanmoins, ce mode de groupe supporte difficilement le passage à l'échelle sans perte de performance, puisque l'acheminement des messages est réalisé un-à-un ce qui est nettement plus coûteux.

Cette plate-forme dispose d'un service de gestion de cohérence des données liées aux groupes. Ce service assure le bon acheminement des données et l'ordre d'arrivée des messages. Ainsi, si des messages sont diffusés dans le désordre, le diffuseur se charge de les réordonner avant de les distribuer aux membres du groupe concernés. Ces diffusions sont réalisées par une communication TCP entre le client et le diffuseur.

Actions et couverture fonctionnelle

L'outil Corona est centré exclusivement sur l'action collective dédiée à la communication et raisonne à haut niveau d'abstraction.

Cette plate-forme distingue deux types de communication : la liste de diffusion et le groupe. Ainsi, pour chacun des types, des services sont définis par un ensemble de primitives. Pour la liste de diffusion, les primitives offertes permettent de créer ou de supprimer une liste de diffusion, de s'abonner ou de se désabonner (*subscribe/unsubscribe*) et de poster un message. Pour le groupe, les primitives permettent de créer, joindre ou quitter un groupe et de diffuser un message. De plus, le groupe distingue deux rôles : le rôle principal et l'observateur. Cependant, il n'est pas possible de définir d'autres rôles ou de changer de rôle au cours d'une session.

Ressources du contexte et observabilité

Parmi les différents outils étudiés, la plate-forme Corona est un cas particulier puisque cet outil porte uniquement sur la communication. Les données, c'est-à-dire la liste des participants et le contenu des messages, sont collectives. L'observabilité d'un message est liée à la liste des destinataires, différente selon le paradigme de communication. En effet, dans le cas de la liste de diffusion, les données sont accessibles à tous alors que dans le cas du groupe, les données sont restreintes à un ensemble de destinataires.

L'observabilité des actions repose sur un mécanisme de diffusion relatif à l'envoi de messages.

5. Synthèse et conclusion

5.1. SYNTHÈSE

Outils	Coordination	Production	Communication
Rendez-Vous	●	-	-
NSTP	◆●	◆●	■
Groupkit	◆●	●	■
Intermezzo	◆	●	■
COCA	◆	●	■
DARE	◆	-	-
Corona	-	-	◆

◆activité de groupe ● Services de la collaboration ■ Mécanismes logiciels
- Néant

Table 1
Tableau de synthèse relatif à l'action collective

Actions et couverture fonctionnelle

La Table 1 présente une synthèse des outils analysés. Pour chaque facette du modèle du trèfle, différents symboles indiquent à quel niveau d'abstraction se situent les fonctions offertes par l'outil correspondant : un losange (◆) pour les outils centrés sur l'activité de groupe, une pastille (●) pour les outils proposant principalement des services de collaboration, un carré (■) pour les outils n'offrant que des mécanismes logiciels. Un tiret (-) indique que l'aspect correspondant n'est pas couvert.

Les outils analysés se concentrent essentiellement sur l'action collective dédiée à la coordination, en dehors de Corona [Hall 1996] axé uniquement sur la communication. Tout d'abord, l'infrastructure Rendez-Vous [Hill 1994], comme indiqué par la Table 1, offre principalement des mécanismes centrés sur la coordination. Il offre des mécanismes pour maintenir un lien cohérent entre les différentes vues et les données partagées. De plus, cette infrastructure offre un service de gestion de sessions avec attribution de rôles pour chaque utilisateur.

NSTP [Patterson 1996] et Groupkit [Roseman 1996a] offrent un support de coordination de plus haut niveau que Rendez-vous. Notamment, la gestion des sessions repose sur des métaphores : la métaphore de la session/conférence pour Groupkit et la métaphore de la place pour NSTP. Ainsi, le gestionnaire de sessions permet de joindre ou quitter plusieurs sessions/conférences/places en fonction des différents droits d'accès. Du point de vue de l'activité de production, les deux infrastructures offrent

des services de partage d'objets par le biais d'environnements partagés, de contrôle de la concurrence et de notification des changements d'état. Cet aspect est un peu plus évolué dans NSTP que dans Groupkit, puisque, d'une part les objets sont mémorisés pour permettre la persistance des données, d'autre part, chaque objet est considéré comme étant un élément d'une place, avec un ensemble d'attributs définissant les droits de modification de celui-ci. Enfin, la communication est assurée par les mécanismes de bas niveau par diffusion de messages ou d'événements.

Intermezzo [Edwards 1996a], COCA [Li 1999], DARE [Bourguin 2000] sont clairement axés sur la coordination en offrant des modèles élaborés reposant fortement sur les langages sous-jacents. En l'occurrence, Intermezzo et COCA expriment la répartition des rôles et les droits associés sous formes de règles logiques, exprimées respectivement en python et dans une forme simplifiée de Prolog. Le modèle proposé par COCA est plus complet étant donné qu'il intègre de nombreux prédicats sur la réalisation des transactions et les modes de communication. Cependant, Intermezzo offre un service de gestion de sessions élaboré puisqu'il prend en compte les sessions initiées de manière explicite et implicite. DARE diffère des deux infrastructures précédentes puisque l'activité de coordination, définie à partir de la théorie de l'activité, est modélisée sous forme de méta-classes pour offrir une infrastructure la plus souple possible et laisser à l'utilisateur la possibilité de "programmer" par manipulation directe sa propre activité de coordination. Enfin, Intermezzo et COCA définissent des services pour l'activité de production incluant un service de gestion de concurrence et de cohérence de données avec stockage. Pour ces trois infrastructures, l'activité de production et de communication est assimilée à l'utilisation d'outils régulée par l'activité de coordination. Cependant, Intermezzo permet de définir la granularité des objets partagés et offrir un support plus fin pour l'activité de production.

Enfin, Corona [Hall 1996] est une infrastructure dédiée à l'activité de communication. Elle repose sur deux paradigmes de communication, deux formes de groupe de communication : la notion de publier/s'abonner et la liste de diffusion. Le premier paradigme permet un passage facile à l'échelle, mais avec une conscience de groupe très limitée. Le second permet au contraire de favoriser une conscience de groupe forte, mais n'autorise pas un passage à l'échelle sans perte de performance.

Ressources du contexte

Hormis les outils DARE et COCA qui raisonnent à un niveau de granularité centré sur les applications, nous constatons deux formes de gestion des ressources : soit les ressources sont uniquement collectives (GroupKit, RendezVous, Corona), soit les ressources sont individuelles et appartiennent à un propriétaire unique (NSTP, Intermezzo).

Observabilité des actions et des ressources

Dans le cas de gestion de ressources collectives, les données sont totalement accessibles et manipulables par les utilisateurs sans aucune politique de protection ou de filtrage. A l'opposé, pour les ressources individuelles, les plates-formes mettent en œuvre des mécanismes de protection des données individuelles à l'aide de droits d'accès. Selon les droits d'accès, ces données sont observables ou non. Ainsi, selon ces outils, rendre une donnée privée accessible est considéré comme la rendre partagée.

Toutes les plates-formes disposent de services de notification afin d'offrir un support à l'observabilité des actions reposant sur des mécanismes de diffusion d'événements en multicast, de trigger, d'appels distants de fonctions (RPC), etc.

Enfin GroupKit et RendezVous sont les deux outils à proposer des mécanismes de couplage de l'interaction par le biais de l'environnement partagé (GroupKit) et du lien reliant la vue à l'abstraction partagée (RendezVous).

5.2. CONCLUSION

La synthèse dressée dans le paragraphe précédent montre que les outils de développement traitent d'éléments de notre grille et donc d'éléments de conception de l'activité de groupe. Cependant pour chaque outil étudié, la couverture vis-à-vis de notre grille est partielle. Par exemple, l'analyse des outils met en évidence le rôle particulier de la coordination : en effet les outils proposent principalement des services de coordination au détriment de la production et de la communication. Or l'infrastructure Corona montre qu'un outil peut aussi proposer des services génériques pour la communication.

Dans le chapitre suivant, nous présentons notre plate-forme de développement, complémentaire aux outils existants. Cette plate-forme :

- permet le développement de collecticiels organisés selon notre modèle d'architecture conceptuelle Clover (Chapitre IV),
- offre des fonctions à haut niveau d'abstraction,
- traite les aspects de coordination, de production et de communication sur le même plan.

Chapitre VI La plate-forme Clover

1. Introduction

Appliquant la même démarche que pour l'étude des modèles d'architecture logicielle, nous avons étudié, dans le chapitre précédent, un ensemble d'outils de développement logiciel pour les collecticiels. A l'issue de cette étude, il est apparu que ces outils n'étaient pas entièrement satisfaisants vis-à-vis de notre grille d'analyse. En particulier, nous avons observé que la majorité des outils se concentrent principalement sur l'aspect coordination de l'action collective au détriment des aspects communication et production. Or ces deux derniers aspects sont aussi importants que l'aspect coordination, comme nous l'avons souligné par l'exposé de l'outil Corona [Hall 1996] à propos de l'aspect communication. De plus, peu d'outils proposent des fonctionnalités à haut niveau d'abstraction, centrées sur l'activité de groupe.

Aussi, s'appuyant sur notre modèle d'architecture conceptuelle Clover, nous avons réalisé une nouvelle plate-forme de développement logiciel pour les collecticiels, afin de proposer une approche complémentaire aux outils existants et qui soit satisfaisante par rapport à notre grille d'analyse. Notamment, nous visons à proposer un outil raisonnant à haut niveau d'abstraction et intégrant les trois aspects relatifs à l'action collective : production, communication et coordination. L'élaboration de cette plate-forme repose sur la définition d'un modèle de couverture fonctionnelle de l'activité de groupe à partir duquel nous avons déduit trois couvertures fonctionnelles dédiées respectivement à la production, à la communication et à la coordination. Le résultat obtenu est une plate-forme logicielle écrite en Java, la plate-forme Clover. Toutefois, il est à noter que ce travail n'a pas pour but de proposer ou de réinventer des services ou algorithmes logiciels mais d'identifier des fonctionnalités

génériques nécessaires pour mettre en œuvre l'activité de groupe comme définie dans notre grille d'analyse.

Ce chapitre détaille la mise en œuvre de la plate-forme Clover et est organisé de la façon suivante : la première partie présente et définit la couverture fonctionnelle de la plate-forme Clover. La seconde partie est dédiée à la mise en œuvre logicielle de cette plate-forme tandis que la troisième partie positionne notre réalisation logicielle vis-à-vis de la grille d'analyse. Nous concluons par un résumé des contributions exposées dans ce chapitre.

2. *Couverture fonctionnelle et activité de groupe*

La couverture fonctionnelle de notre plate-forme Clover se décompose en trois parties dédiées à la production, à la communication et à la coordination. Les trois couvertures fonctionnelles correspondantes sont déduites d'un modèle de couverture fonctionnelle de l'activité de groupe.

Tandis que le premier paragraphe présente ce modèle, le second paragraphe présente les trois couvertures fonctionnelles.

2.1. CONCEPTS ET COUVERTURE FONCTIONNELLE GÉNÉRIQUE

Le modèle de couverture fonctionnelle de l'activité de groupe s'articule selon trois concepts : les participants à l'activité de groupe (*qui*), le support de l'activité de groupe (*quoi*) et l'espace de l'activité de groupe (*où*). Ces concepts sont définis de la façon suivante :

- **Les participants à l'activité de groupe** (*qui*) : un participant représente un utilisateur impliqué dans une activité de groupe. Un participant peut mener une action de groupe ou une action individuelle. Cette dernière pouvant bien sûr contribuer à l'accomplissement d'une tâche commune.

- **Le support à l'activité de groupe** (*quoi*) : le support à l'activité de groupe est la matérialisation de l'activité de groupe, c'est-à-dire l'expression concrète du contexte partagé. Par exemple, dans un contexte de coordination, il s'agit d'un groupe formé de membres et régi par des lois. Dans un contexte de production, il s'agit du document produit par plusieurs auteurs. Dans un contexte de communication, il s'agit du cercle de discussion auquel participent des individus.

- **L'espace de l'activité de groupe** (*où*) : il s'agit du lieu dans lequel se déroule l'activité de groupe et, par conséquent, dans lequel évoluent les

participants à l'activité de groupe. Par analogie, l'espace de l'activité de groupe est comparable à la métaphore de la pièce (*room*) [Greenberg 1998] ou de la place [Harrison 1996] (*place*, employé notamment par l'outil NTSP [Patterson 1996], présenté au chapitre précédent).

Pour illustrer ces trois concepts, prenons par exemple le système IRC (*Internet Relay Chat*) [IRC 2002]. Ce système permet à des utilisateurs de communiquer de façon synchrone avec différents interlocuteurs à travers des forums de discussion définis par un thème (*channel*). Ce système met en œuvre une activité de groupe dédiée à la communication. Le système gère donc un espace de communication peuplé d'interlocuteurs et constitué de nombreux forums de discussion. Un forum de discussion est une forme de communauté dont il faut être membre pour pouvoir communiquer avec les autres membres. En ce sens, un forum de discussion est un support à l'activité de groupe et la communication à travers ce forum matérialise cette activité de groupe dédiée à la communication.

Nous avons retenu ces trois concepts élémentaires pour définir un modèle de couverture fonctionnelle de l'activité de groupe. Contrairement à des modèles tels que la théorie de l'activité [Kuuti 1991] ou la théorie de la coordination [Malone 1990], notre approche ne tend pas à analyser et à conceptualiser la mécanique de l'activité de groupe. En effet, ces théories visent à capter l'aspect structurel et organisationnel de l'activité de groupe. Notre approche est complémentaire puisque nous abordons le problème uniquement du point de vue fonctionnel.

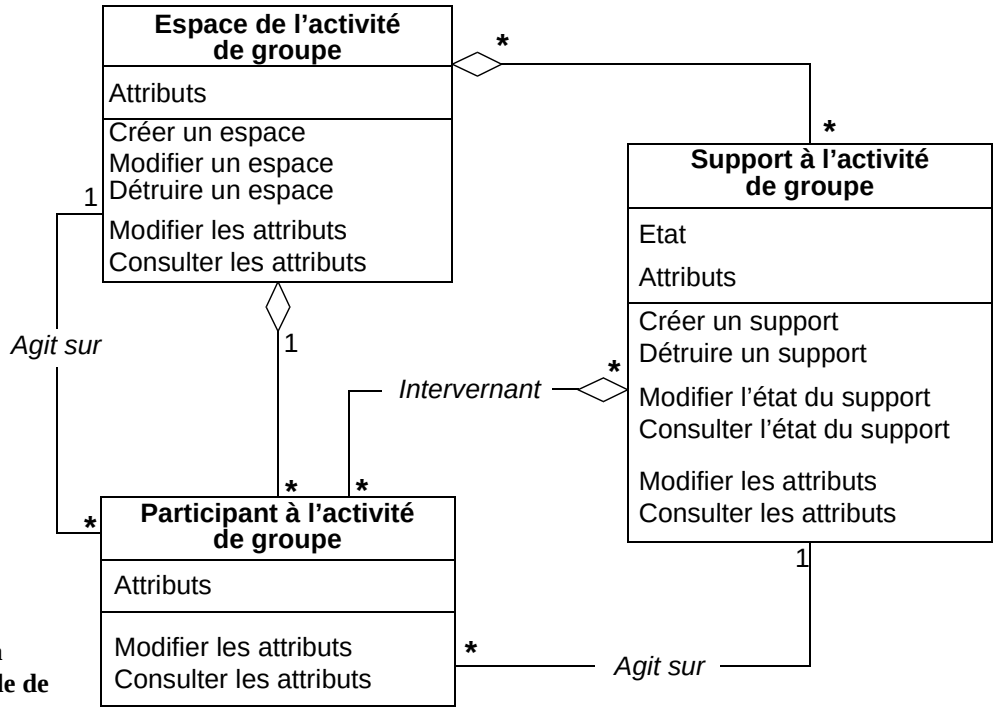


Figure 1
Modèle générique de la couverture fonctionnelle de l'activité de groupe

Ce modèle de couverture fonctionnelle de l'activité de groupe est décrit en *UML* (*Unified Modeling Language*) à la Figure 1. Ce modèle repose donc sur les trois concepts identifiés précédemment et considérés sous forme de classes *UML*. Le diagramme *UML* repose sur les fonctions usuelles : créer, détruire, consulter, modifier.

Les trois classes du modèle sont caractérisées de la façon suivante :

- **L'espace de l'activité de groupe** est peuplé de participants à l'activité de groupe (agrégation) et est composé de supports à l'activité de groupe (agrégation), sachant qu'un support peut être partagé par plusieurs espaces. Il est possible de créer et de détruire un espace. Un participant peut consulter les informations propres à un espace (attributs). L'espace est modifié lorsqu'un participant arrive ou quitte cet espace, ou lorsqu'un support est créé ou détruit.

- **Un participant à l'activité de groupe** est défini par un ensemble d'attributs comme, par exemple, des informations personnelles. Ces attributs, en fonction des règles de publication et de protection des informations (filtrage), peuvent être consultés et modifiés. Un participant peut être autorisé à intervenir au cours d'une activité de groupe en agissant sur le support à l'activité de groupe.

- **Un support à l'activité de groupe** est défini par un état et des attributs. L'état du support est modifiable et consultable : il traduit l'état de l'activité de groupe. Les attributs sont les données qui caractérisent ce support comme, par exemple, les droits d'accès. Le support change d'état suite à une action collective menée par les participants autorisés à intervenir au cours d'une activité de groupe (modifier).

A partir de ce modèle de couverture fonctionnelle de l'activité de groupe, nous avons déduit trois couvertures fonctionnelles dédiées à la production, à la communication et à la coordination. Le paragraphe suivant détaille ces trois couvertures fonctionnelles.

2.2. COUVERTURES FONCTIONNELLES

Nous présentons ici les trois couvertures fonctionnelles qui ont été déduites du modèle de couverture fonctionnelle de l'activité de groupe de la Figure 1. Pour chaque instance du modèle, nous avons identifié des concepts spécifiques à l'aspect de l'activité de groupe considéré.

La première section présente la couverture fonctionnelle dédiée à la production, la seconde section celle dédiée à la communication, et la troisième section celle dédiée à la coordination.

Couverture fonctionnelle pour la production

La couverture fonctionnelle décrite dans cette section est dédiée à l'aspect production de l'activité de groupe. Les trois classes *UML* du modèle de couverture fonctionnelle ont été spécialisées et sont présentées à la Figure 2.

- **L'espace de la production** regroupe les documents et les auteurs. Cet espace recense l'ensemble des auteurs présents et l'ensemble des documents crées (attributs). Par exemple, un système de bureau partagé du type *TeamRooms* [Roseman 1996b] (présenté dans le Chapitre II) peut mettre en œuvre différents espaces de production : un espace dédié à la production d'annotations par le biais de post-it électroniques, un espace dédié à la production de texte par le biais d'un éditeur partagé ou un espace dédié au *brainstorming* (production d'idées) par le biais d'un tableau blanc partagé.

- **Un auteur** est un participant à l'activité dédiée à la production. Ainsi, un auteur peut participer à l'élaboration d'un document commun (action collective) ou produire un document à titre personnel (activité individuelle).

- **Un document** est un support à l'activité de groupe dédiée à la production. Il s'agit de l'entité produite par plusieurs auteurs reflétant

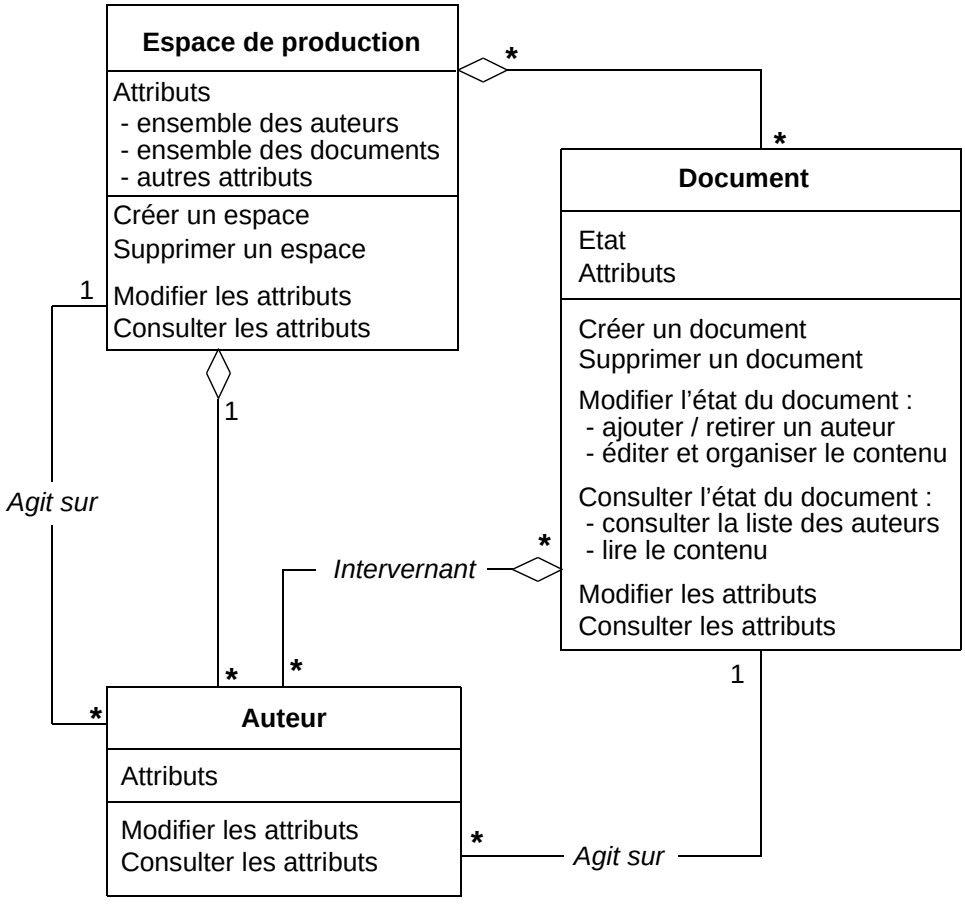


Figure 2
Couverture fonctionnelle dédiée à la production

l'état de l'activité de groupe. Le concept de document est, dans ce contexte, une métaphore pour symboliser le résultat d'une production : un document n'est pas nécessairement un texte ou un dessin.
Un document est caractérisé par un contenu et par une liste d'auteurs participant à son élaboration (état). Un auteur autorisé à accéder au contenu peut soit le modifier, soit le consulter. Les attributs sont les données caractérisant les documents tels que les droits accordés aux auteurs, la taille du document, etc. Les droits et les rôles attribués à un auteur (par exemple, auteur principal, lecteur, rédacteur, etc.) sont définis par rapport à un document. Ainsi, un auteur peut jouer différents rôles suivant le document.

Couverture fonctionnelle pour la communication

La seconde couverture fonctionnelle est dédiée à l'aspect communication de l'activité de groupe. Elle est présentée sous forme de diagrammes *UML* à la Figure 3 :

- **L'espace de communication** regroupe les abonnés et est constitué de listes de diffusion. Dans un espace donné, un abonné peut intervenir dans plusieurs listes de diffusion. Par exemple, le système CVE [Reynard 1998], qui combine un mediaspace et un système vidéoconférence dans un espace virtuel en 3D, met donc en place

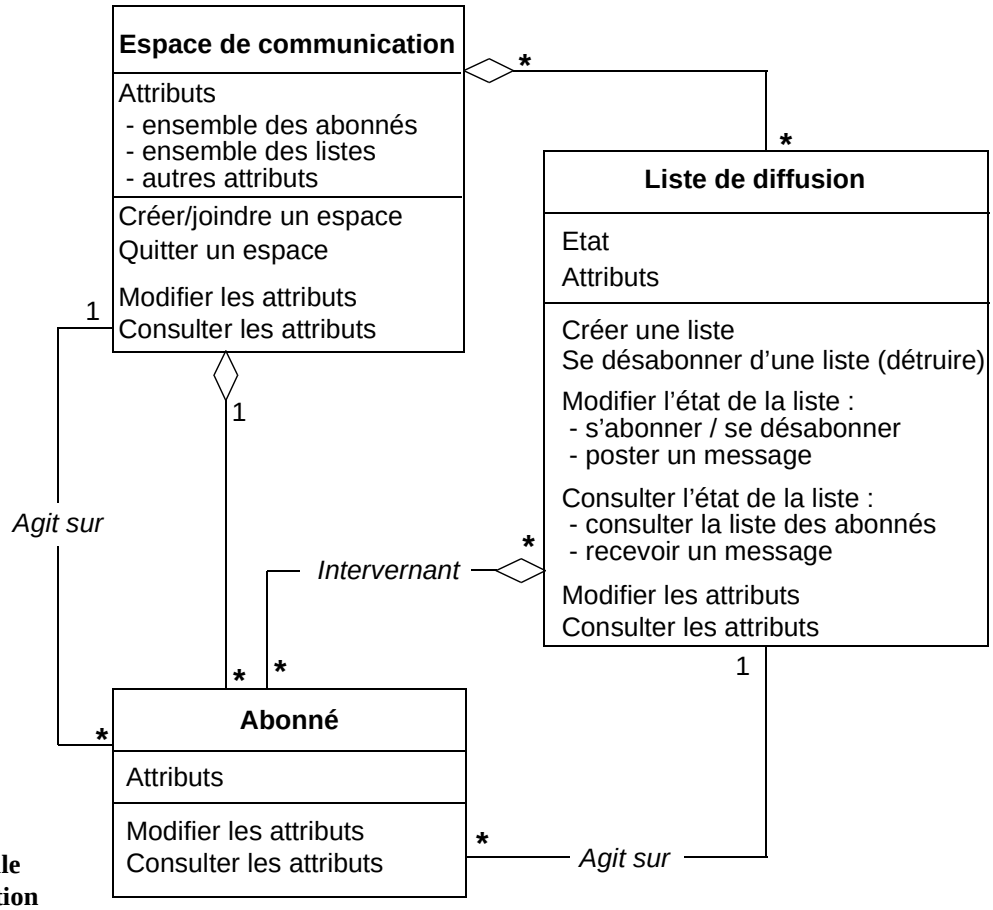


Figure 3
Couverture fonctionnelle dédiée à la communication

plusieurs espaces de communication : un espace dédié à la communication informelle par le biais d'un *mediaspace* et un espace dédié à la communication vidéo par le biais d'un système de vidéoconférence. Nous pouvons imaginer qu'il soit possible d'ajouter un espace de communication textuel par le biais des *email*.

- **Un abonné** est un participant à l'activité dédiée à la communication. Un abonné a la possibilité de joindre et de quitter un espace de communication ainsi que de s'abonner ou de se désabonner d'une liste de diffusion.

- **La liste de diffusion** est un support à l'activité dédiée à la communication. Ce concept s'inspire largement du principe "*publish/ subscribe*" (publier/s'abonner) abordé dans le chapitre précédent avec la présentation de l'outil Corona [Hall 1996]. La liste de diffusion est une métaphore illustrant cette notion de cercle de discussion. Un abonné peut s'abonner ou se désabonner d'une liste de diffusion, peut consulter ou envoyer des messages en fonction du rôle qui lui a été attribué. L'état de la liste de diffusion reflète l'état de la discussion en cours entre les abonnés, c'est-à-dire l'ensemble des messages échangés. Les droits et les rôles attribués à un auteur sont relatifs à une liste de diffusion. Par exemple, le système IRC [IRC 2002] définit trois types d'abonnés : l'administrateur d'un canal à l'origine de sa création, les *sysop* (*system operator*) qui disposent des mêmes privilèges que l'administrateur et qui ont la possibilité de conférer ces droits à d'autres utilisateurs (les *sysops* sont nommés par l'administrateur mais ce dernier peut perdre ses droits) et les simples abonnés.
  Par exemple, une liste de diffusion peut être une *mailing list*, un *newsgroup*, une vidéoconférence, une audioconférence, etc. Considérons le cas d'un système de conférence reposant sur des échanges textuels. Nous pouvons identifier deux listes de diffusion :

1. L'une est dédiée au conférencier pour qu'il puisse réaliser sa conférence à destination des auditeurs. Ces derniers n'ont pas le droit de communiquer sur cette liste, afin de ne pas "bruiter" le message du conférencier avec leurs discussions.
2. L'autre liste est dédiée aux auditeurs pour qu'ils puissent discuter entre eux (comme cela est courant lors d'une conférence en vraie grandeur) sans pour autant perturber le conférencier.

Couverture fonctionnelle pour la coordination

La troisième couverture fonctionnelle est dédiée à l'aspect coordination de l'activité de groupe. Les trois classes *UML* correspondantes sont présentées à la Figure 4. Elles sont déduites du modèle de couverture fonctionnelle de l'activité de groupe de la Figure 1 :

- **L'espace de coordination** regroupe les membres et est constitué de groupes. Un espace de coordination est assimilable à une session dans la terminologie de l'outil GroupKit [Roseman 1992] (présenté dans le chapitre précédent). Un utilisateur peut rejoindre cet espace et devenir membre d'un groupe ou créer un nouveau groupe. Réciproquement un membre peut quitter son groupe et quitter l'espace. Pour reprendre l'exemple de GroupKit, un espace de coordination est caractérisé par une session et un groupe est caractérisé par une conférence. Les membres sont les participants à une session (espace de collaboration) sachant qu'un membre peut intervenir dans plusieurs conférences (groupe).

- **Les membres** sont les participants à une activité dédiée à la coordination. Un membre est caractérisé par un ensemble d'attributs. Ces attributs sont consultables ou modifiables en fonction des règles de publication.

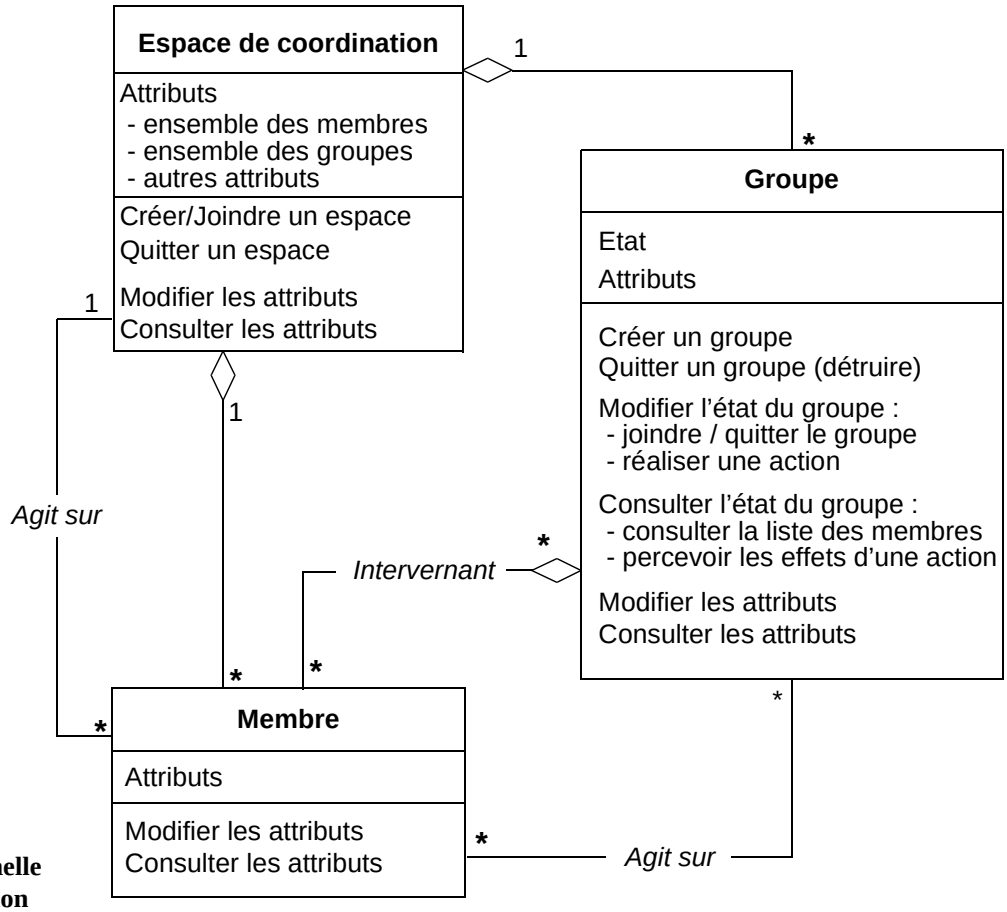


Figure 4
Couverture fonctionnelle dédiée à la coordination

- **Le groupe** est un support à l'activité dédiée à la coordination. L'état du groupe reflète l'assignation des tâches, l'organisation des actions à réaliser et l'assignation de droits à chaque membre ce qui revient à définir la notion de rôle. L'état du groupe est modifié suite à la réalisation d'une action collective ou lorsqu'un membre rejoint ou quitte le groupe. Consulter l'état du groupe revient à connaître comment sont répartis les rôles et quelles sont les tâches à réaliser, ainsi qu'à percevoir les changements d'états induits par la réalisation d'une action. Les attributs sont les données caractérisant le groupe comme sa politique d'organisation du travail. Par exemple, dans un système *workflow*, la politique d'organisation du travail consiste à définir à quel moment un utilisateur est autorisé à travailler sur un document.

Dans cette partie, nous avons présenté trois couvertures fonctionnelles correspondant aux trois aspects production, communication et coordination de l'activité de groupe. Ces couvertures fonctionnelles cernent les fonctions de la plate-forme : spécifications fonctionnelles. La plate-forme spécifiée, il convient maintenant d'en étudier sa réalisation logicielle.

3. Mise en œuvre de la plate-forme Clover

Cette partie détaille la mise en œuvre logicielle de notre plate-forme Clover. Le premier paragraphe présente une traduction des trois couvertures fonctionnelles en classes Java. Ces classes définissent l'interface de programmation de la plate-forme Clover. Le second paragraphe présente la mise en œuvre de la partie serveur de la plate-forme. Dans le troisième paragraphe, nous dressons un bilan de la qualité du logiciel développé.

3.1. COUVERTURE FONCTIONNELLE ET CLASSES JAVA

Nous présentons une traduction des trois couvertures fonctionnelles sous forme de classes Java. Les trois sections suivantes traitent respectivement des classes relatives à la production, à la communication et à la coordination.

Classes Java relatives à la production

Pour mettre en œuvre la couverture fonctionnelle dédiée à la production, nous avons défini, comme le montre le diagramme *UML* de la Figure 5, une classe et quatre interfaces Java. La traduction diffère légèrement de la couverture fonctionnelle car nous avons introduit le concept d'album qui est équivalent au concept de document dans la terminologie de la

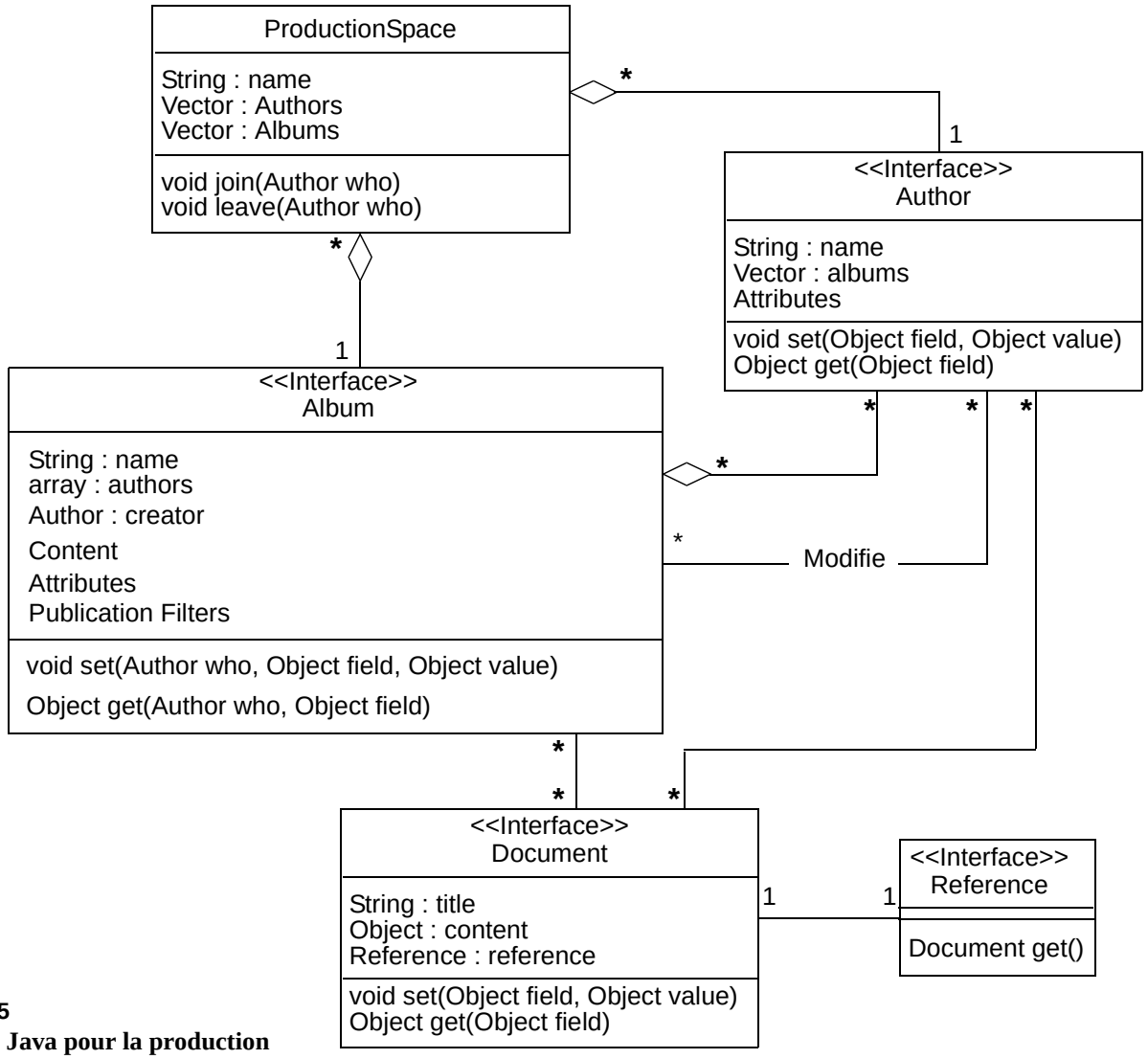


Figure 5
Classes Java pour la production

couverture fonctionnelle. De plus nous avons introduit le concept de référence de document, dans le sens référence à une unité documentaire. Ces classes et interfaces sont définies de la façon suivante :

- Classe `ProductionSpace` : cette classe implémente le concept d'espace de production. Elle maintient une liste des auteurs et des albums en cours de création. Cette classe dispose de deux méthodes permettant de joindre ou quitter l'espace.

- Interface `Author` : cette interface implémente le concept d'auteur défini par un nom et associé à la production d'un ensemble d'albums. L'interface `Author` dispose de deux méthodes pour modifier et consulter les attributs.

- Interface `Album` : l'interface `Album` implémente le concept de document (i.e. de la couverture fonctionnelle) qui peut être créé par un

ou plusieurs auteurs. Un album est identifié par un nom, un ensemble d'auteurs, un créateur et est doté d'un contenu. Il est aussi possible de modifier les attributs et de définir des filtres de publication. Cette interface met à disposition deux méthodes qui permettent de modifier et de consulter le contenu, de changer la liste des auteurs ou de modifier les attributs

- Interface `Document` : cette interface représente une unité documentaire (son, image, etc.) identifiée par un nom, un contenu et une référence.

- Interface `Reference` : l'interface `Reference` représente le concept de référence d'un document. Cette interface permet de faire le lien avec un document. Par exemple, une référence peut être le chemin d'accès à un fichier image, une référence de livre, une requête SQL ou une URL.

Classes Java relatives à la communication

Pour mettre en œuvre la couverture fonctionnelle dédiée à la communication, nous avons défini, comme le montre le diagramme *UML* de la Figure 6, une classe et deux interfaces Java :

- Classe `CommunicationSpace` : cette classe implémente le concept d'espace de communication. Cette classe est identifiée par un nom et

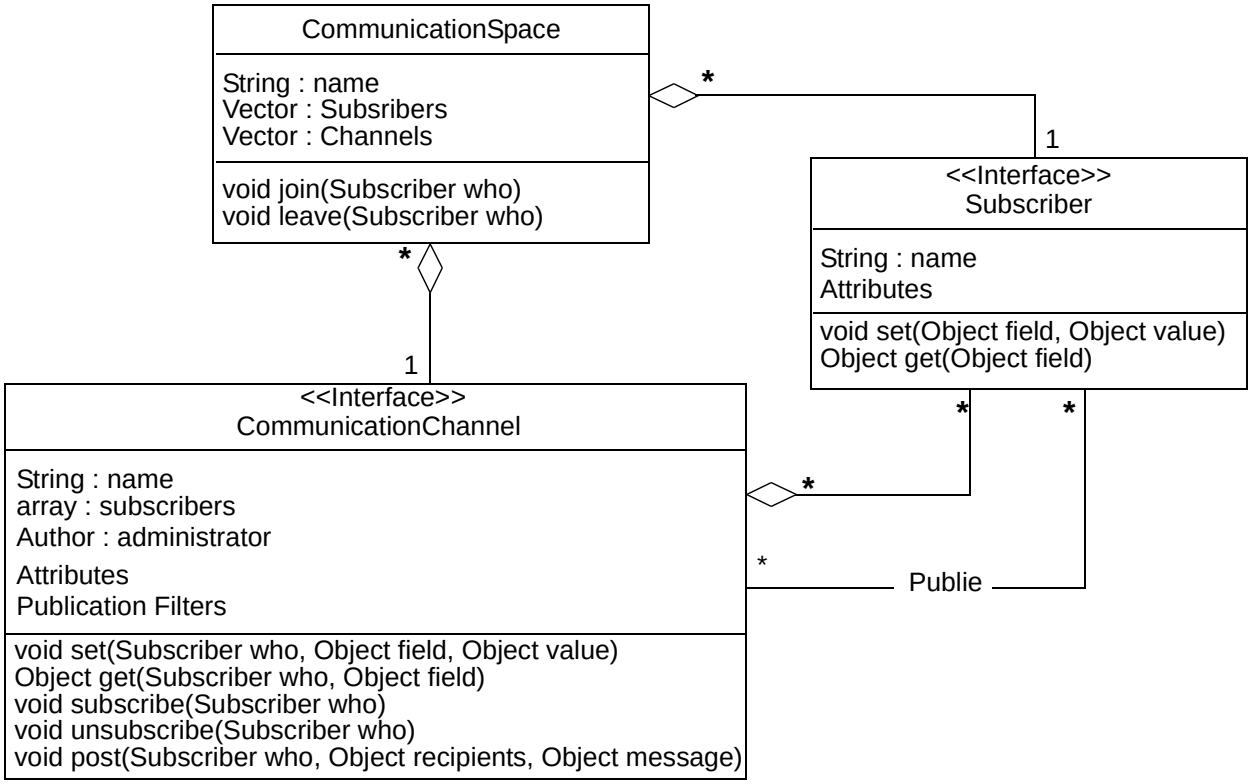


Figure 6
Classes Java pour la communication

gère un ensemble d'abonnés et de listes de diffusion. Un abonné peut joindre ou quitter un espace par le biais de deux méthodes.

- Interface `Subscriber` : l'interface `Subscriber` implémente le concept d'abonné identifié par un nom et des attributs qu'il est possible de modifier et de consulter par le biais de deux méthodes.

- Interface `CommunicationChannel` : cette interface implémente le concept de liste de diffusion. Une liste de diffusion est identifiée par un nom, une liste d'abonnés et un administrateur. Les méthodes disponibles permettent à un abonné de s'abonner, de se désabonner ou de poster un message. De plus, un membre de cette liste peut modifier les attributs et les filtres de publication.

Classes Java relatives à la coordination

Pour mettre en œuvre la couverture fonctionnelle dédiée à la coordination, nous avons défini, comme le montre le diagramme *UML* de la Figure 7, une classe et deux interfaces Java :

- Classe `CoordinationSpace` : cette classe implémente le concept d'espace de coordination identifiée par un nom, par la liste de ses membres et de ses groupes. Deux méthodes permettent à un membre de quitter ou joindre l'espace.

- Interface `User` : l'interface `User` implémente le concept de membre identifié par un nom, un mot de passe, des filtres de publication et des

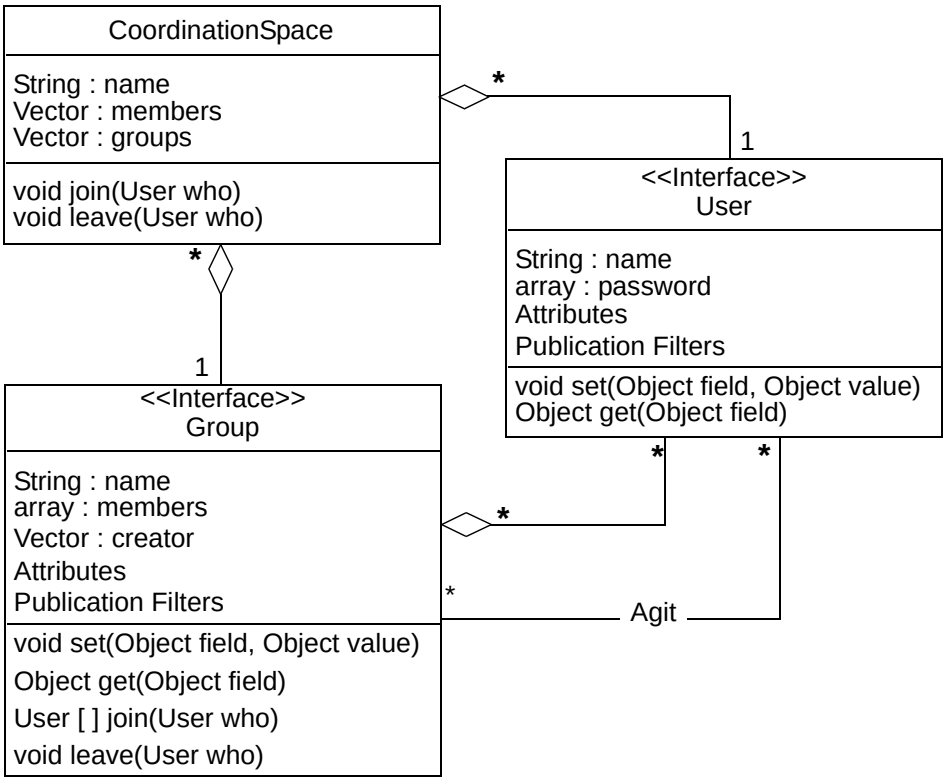


Figure 7
Classes Java pour la coordination

attributs qu'il est possible de consulter ou modifier par le biais de deux méthodes.

- Interface `Group` : cette interface implémente le concept de groupe identifié par un nom, une liste de ses membres, un créateur, des filtres de publication et des attributs. Trois méthodes permettent à un membre de joindre le groupe, de quitter le groupe ou d'exécuter une action suivant son rôle au sein du groupe. Deux autres méthodes permettent à un membre de modifier ou consulter les attributs et les filtres de publication.

Le développement d'un collecticiel à l'aide de notre plate-forme Clover nécessite donc la programmation de classes implémentant les différents concepts définis par la couverture fonctionnelle. Concrètement, il convient donc de programmer les classes implémentant les différentes interfaces présentées. Afin d'illustrer l'utilisation de ces classes et interfaces, le chapitre suivant présente la réalisation de deux collecticiels : le système CoVitesse et un système de tableau blanc partagé.

A l'exécution du collecticiel, ces classes sont instanciées sous forme d'objets qui vont être partagés avec la partie serveur mise en œuvre par la plate-forme. Le serveur prend connaissance de l'existence de ces objets uniquement à l'exécution par chargement dynamique des classes et des objets. En ce sens, la partie serveur est totalement indépendante de la sémantique (domaine d'application) du collecticiel. Cette indépendance totale rend la plate-forme Clover générique. Nous détaillons la réalisation logicielle de ce serveur dans le paragraphe suivant.

3.2. MODÈLE CLIENT-SERVEUR

La plate-forme, dans sa version actuelle, impose un schéma d'exécution hybride de type client-serveur. A terme, il est prévu de libérer le développeur de cette contrainte en lui permettant de choisir le schéma d'exécution désiré. Le serveur est constitué de quatre serveurs dont trois sont dédiés à l'activité de groupe. La première section, intitulée "Les serveurs dédiés à l'activité de groupe", détaille le rôle de ces quatre serveurs. La seconde section, intitulée "Diagramme d'états du serveur de coordination", décrit la machine à état qui régit le fonctionnement du serveur dédié à la coordination. Enfin, la troisième section, intitulée "Modèle multi-agent et protocole d'échanges", présente le modèle multi-agent que nous avons retenu pour développer un collecticiel et communiquer avec la partie serveur de notre plate-forme Clover.

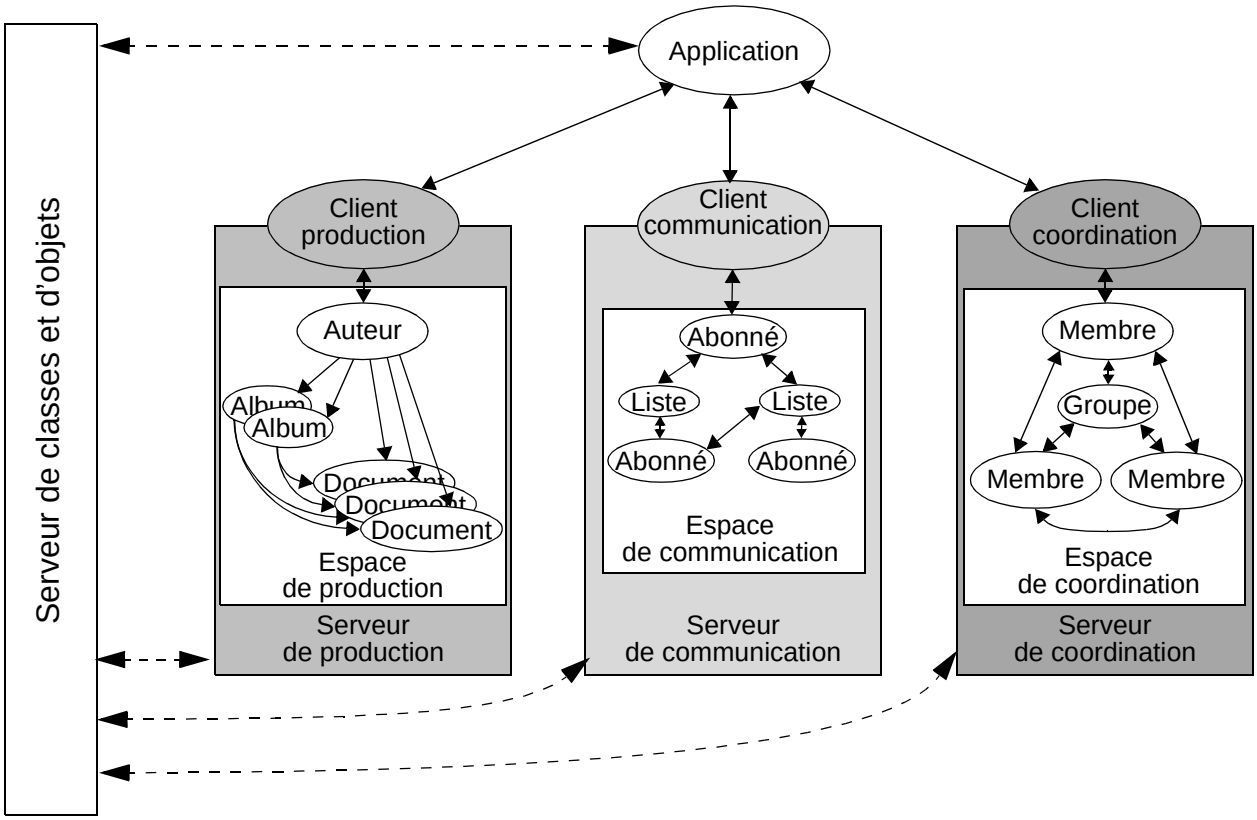


Figure 8
Architecture d'implémentation de la plate-forme Clover

Les serveurs dédiés à l'activité de groupe

La partie serveur de la plate-forme, comme le montre la Figure 8, est constituée de trois principaux serveurs offrant un support à l'activité de groupe et dédiés respectivement à la production, à la communication et à la coordination. A cela s'ajoute un quatrième serveur dédié à la migration de code Java et au chargement dynamique de classes et d'objets.

Ces quatre serveurs jouent les rôles suivants :

- **Serveur de production** : ce serveur gère l'aspect production de l'activité de groupe. Il manipule les concepts, instanciés sous forme d'objets, d'espace de production, d'auteur et de document. Un auteur représente un utilisateur dans un espace de production sachant qu'un utilisateur peut intervenir dans plusieurs espaces de production. Pour chaque espace de production nous associons un moyen de production comme un éditeur de texte ou un éditeur de dessin. Dans ce contexte, comme nous l'avons évoqué dans le paragraphe précédent, la terminologie diffère légèrement avec le vocabulaire de la couverture fonctionnelle. Du point de vue de la plate-forme, un album correspond au concept de document de la couverture fonctionnelle et un document correspond à une unité documentaire (élaboré et inaltérable) comme une image, une formule ou une page *WWW*. De plus, ce serveur se charge de mémoriser de manière persistante les albums créés et les documents afin de permettre la restauration des données d'une session

à l'autre. La partie cliente du collecticiel accède à ce serveur par le biais d'un client production (agent *Client production*) qui masque les échanges à travers l'infrastructure réseau. Ce point est détaillé dans la section intitulée "Modèle multi-agent et protocole d'échanges" de ce paragraphe.

- **Serveur de communication** : ce serveur gère l'aspect communication de l'activité de groupe. Il manipule les concepts, instanciés sous forme d'objets, d'espace de communication, d'abonné et de liste de diffusion. Un abonné représente un utilisateur dans un espace de communication sachant qu'un utilisateur peut s'abonner à plusieurs listes de diffusion. Il est aussi possible de communiquer avec un ensemble restreint de destinataires. A un espace de communication est associé un moyen de communication : mél, téléphone, vidéo, audio, etc. De la même façon, l'application accède au serveur via un client communication (agent *Client communication*).

- **Serveur de coordination** : ce serveur gère l'aspect coordination de l'activité de groupe. Il manipule les concepts, instanciés sous forme d'objets, d'espace de coordination, de membre et de groupe. Un membre représente un utilisateur dans un espace de coordination. Ce serveur a la charge de mettre en relation les différents membres et mettre en œuvre la coordination au sein des groupes. En fonction du groupe, des rôles sont attribués aux membres. Ces rôles définissent des actions qu'un membre peut effectuer. La politique de coordination est régie et définie par un type de groupe. L'espace de coordination est un lieu de rencontre entre différents utilisateurs. Comme pour les deux autres serveurs, la partie cliente du collecticiel accède au serveur via un client coordination (agent *Client coordination*).

- **Serveur de classes et d'objets** : ce serveur a un rôle particulier puisqu'il permet aux serveurs dédiés à l'activité de groupe et à la partie cliente du collecticiel de partager les mêmes définitions de classe et les même instances d'objets. L'apport de ce serveur est double car il permet :

  1. Le partage et la migration d'objets instanciés entre la partie cliente et la partie serveur du collecticiel. En effet, la partie serveur n'a aucune connaissance à la compilation des objets manipulés par la partie cliente. Ainsi, lors d'un échange à base d'objets entre la partie cliente et la partie serveur, le serveur de classe se charge de récupérer la définition de classe afin que la partie serveur puisse manipuler un objet dont la définition de classe est inconnue par défaut.

2 Le partage d'objets communs entre les trois serveurs sur lesquels ces derniers procèdent à des traitements spécifiques.

Ce serveur de classes et d'objets constitue l'abstraction d'un composant Clover (Chapitre IV) dont les trois sous-composants sont les trois serveurs dédiés à la production, à la communication et à la coordination.
La réalisation logicielle de ce serveur repose sur le mécanisme de chargement de classe mis en œuvre par la machine virtuelle Java. Ainsi, lorsqu'un des trois serveurs doit manipuler un objet dont il ne connait pas la description, celui-ci envoi une requête au serveur de classe qui se charge de la récupérer au niveau de l'application. Ceci est transparent pour le développeur car cette requête est émise automatiquement dès qu'un objet est échangé entre l'application et les trois serveurs à travers le concept de *stream* (`ObjectInputStream` et `ObjectOutputStream`).

Diagramme d'états du serveur de coordination

Dans cette section, nous détaillons l'un des serveurs, le serveur de coordination. Pour cela, nous présentons la machine à état du serveur. Ce serveur est le seul, contrairement aux serveurs de communication et de production, à disposer d'une machine à état ayant la charge de synchroniser les membres et les groupes. Cette responsabilité revient donc à la partie serveur du collecticiel. Au contraire, dans le cas de la production et de la communication, la responsabilité de synchroniser les auteurs, documents, abonnés et listes de diffusion est déléguée à la partie cliente du collecticiel.

Le diagramme d'état, représenté par la Figure 9, est constitué de trois principaux états et de deux états transitoires. Pour chaque transition entre deux états, nous indiquons le type de l'événement à l'origine de la transition. Ce mécanisme de transition par événement repose sur une communication par message selon notre modèle multi-agent. La communication par message est détaillée dans la section suivante.

Les trois états principaux sont le mode *non connecté* (i.e. l'utilisateur n'a pas rejoint un espace de collaboration), l'*évolution dans un espace de collaboration dans un mode individuel* et l'*évolution dans un espace de collaboration en tant que membre d'un groupe*.

Le premier état transitoire est l'attente d'une réponse pour rejoindre un espace de collaboration à partir de l'état *non connecté*. Le passage à cet état fait suite à la soumission d'une requête de demande de connexion (`CONNECTION`, `REQ`). Si la connexion est refusée ou si la requête est annulée, le serveur retourne dans l'état *non connecté*. Il est possible de retourner dans l'état *non connecté* à partir de tous les autres états suite à une demande de déconnexion (`DECONNECTION`), d'annulation de connexion (`CONNECTION`, `CANCEL`) ou de refus de connexion

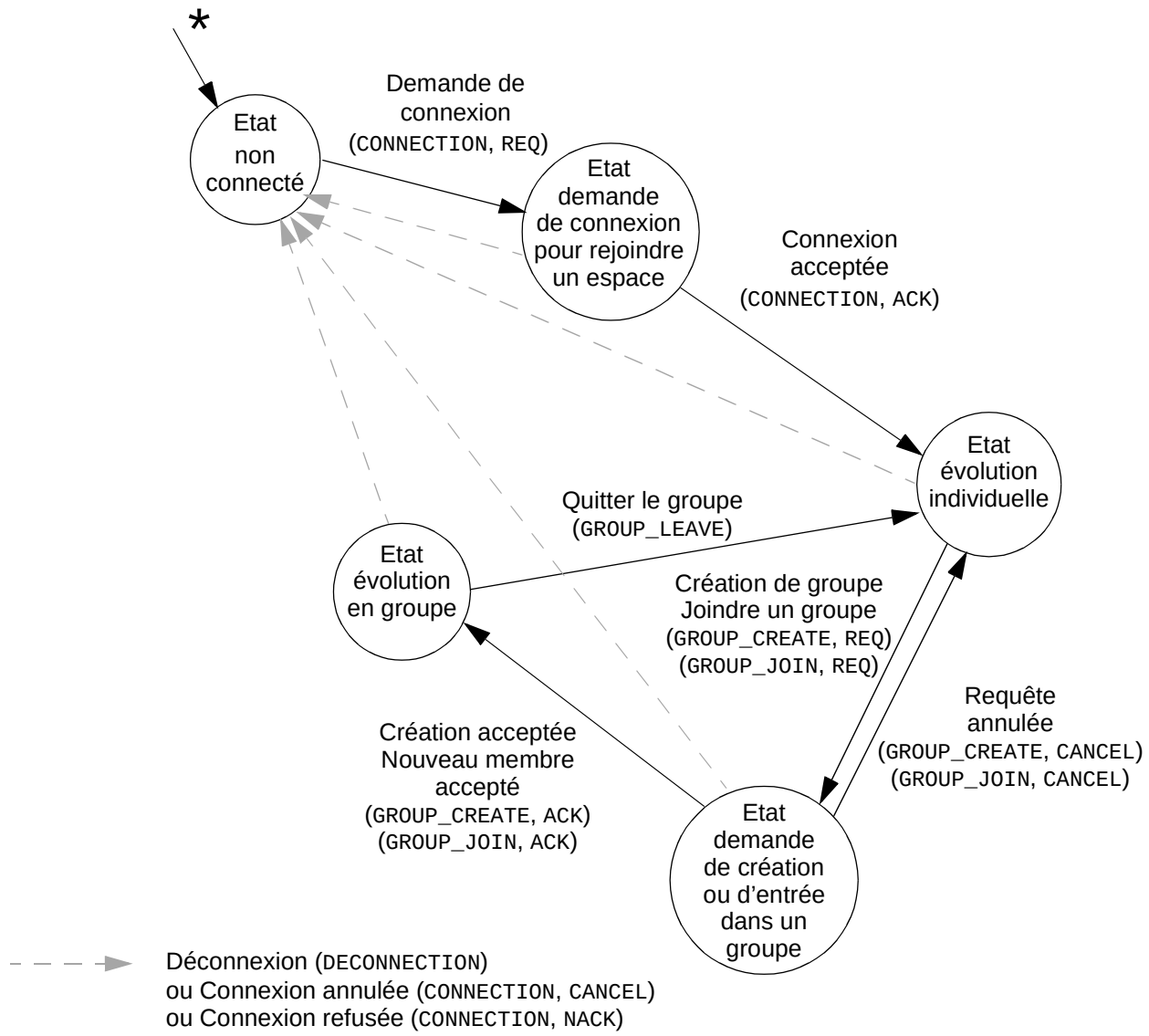


Figure 9
Diagramme d'état du serveur de coordination

(CONNECTION, NACK). Ce changement d'état est symbolisé par une flèche grise et en pointillé dans la Figure 9. Si la connexion est acceptée (CONNECTION, ACK), le serveur passe à l'état *évolution individuelle*.

Le second état transitoire correspond au passage à un état *évolution en groupe*, c'est-à-dire lorsque l'utilisateur souhaite créer ou joindre un groupe. Le passage à cet état à partir de l'état *évolution individuelle* fait suite à l'envoi d'une requête de création de groupe (GROUP_CREATE, REQ) ou de demande d'entrée dans le groupe (GROUP_JOIN, REQ). Dans le cas d'un refus ((GROUP_JOIN, NACK), (GROUP_CREATE, NACK)) ou d'une annulation de requête ((GROUP_JOIN, CANCEL), (GROUP_CREATE, CANCEL)), le serveur retourne à l'état *évolution individuelle*, sinon, le serveur passe dans l'état *évolution en groupe* ((GROUP_JOIN, ACK), (GROUP_CREATE, ACK)). Dans cet état, il est

possible de quitter le groupe pour retourner dans l'état *évolution individuelle* (GROUP_LEAVE).

Modèle multi-agent et protocole d'échanges

Nous avons adopté une approche multi-agent pour développer la plate-forme et pour assurer les échanges entre la partie cliente et la partie serveur du collecticiel. Ce modèle multi-agent est aussi préconisé pour développer un collecticiel avec notre plate-forme. Nous illustrons ce point avec le développement de deux collecticiels au chapitre suivant.

Comme nous l'avons expliqué lors de la description des trois serveurs, la partie cliente du collecticiel accède aux serveurs par le biais de trois agents : l'agent *Client production*, l'agent *Client communication* et l'agent *Client coordination*. Ces trois agents sont programmés sous forme de trois classes : `ProdClient`, `CommClient` et `CoordClient`. Ils encapsulent la partie communication à travers le réseau avec le serveur dédié correspondant. Cet agent client, localisé dans la partie cliente du collecticiel, interagit avec un agent *Proxy* localisé dans la partie serveur. Cet agent *Proxy* joue exactement le même rôle que l'agent client : il a la charge de masquer les aspects réseaux du côté serveur et réalise les échanges avec son homologue de la partie cliente. L'agent *Proxy* est programmé sous la forme d'une classe, la classe `RemoteClient`. Grâce à ce mécanisme, nous rendons ainsi l'aspect client-serveur entièrement transparent, en masquant l'infrastructure réseau. Enfin, du côté serveur, l'agent `RemoteClient` communique avec un agent qui représente l'utilisateur en fonction du serveur : un agent `Auteur` pour la production,

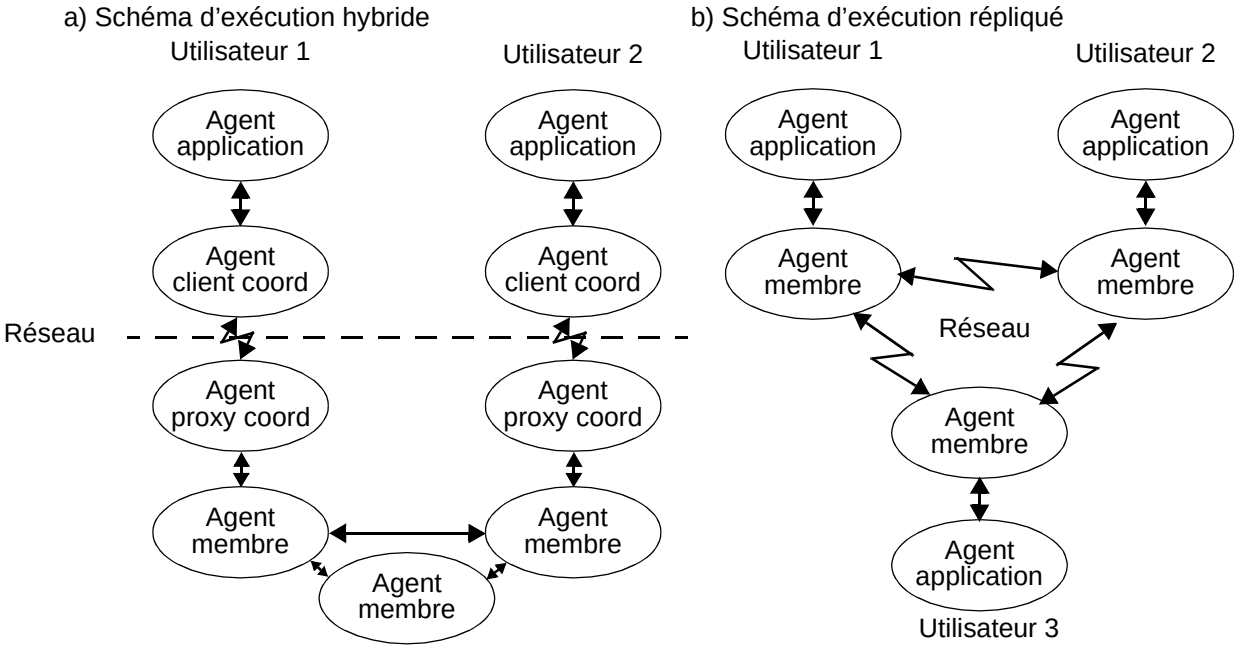


Figure 10
Communication dans un modèle à agents du point de vue de la coordination

un agent `Abonné` pour la communication et un agent `Membre` pour la coordination.

L'utilisation d'un agent *Client* et *Proxy* est une technique courante qui correspond, entre autre, aux *stub* et *skeleton* des java *RMI* (*Remote Method Invocation*). Ces derniers se chargent de masquer les aspects réseaux pour l'invocation à distance de méthodes sur un objet géré par un processus réparti.

Pour illustrer le rôle de ces agents, considérons l'exemple de la Figure 10 (a) : il s'agit d'une hiérarchie d'agents constituant un collecticiel selon le point de vue de la coordination. Dans cet exemple, la partie cliente est constituée de deux agents : un agent application relié à un agent *Client coordination*. Ce dernier communique avec le serveur de coordination à travers le réseau en s'adressant à son homologue, l'agent *Proxy*. L'agent *Proxy* est relié à un agent *Membre* qui représente l'utilisateur dans un espace de coordination. Au sein de cet espace de coordination, l'agent *Membre* communique directement avec les autres agents *Membre*. Ainsi, tous les clients sont mis en relation de façon transparente.

Dans la version actuelle de la plate-forme, illustré par l'exemple de la Figure 10 (a), l'élaboration d'un collecticiel repose nécessairement sur un schéma d'exécution hybride selon un modèle client-serveur. Cependant, avec cette approche par agent, comme le montre l'exemple de la Figure 10 (b), il est tout à fait possible d'élaborer un collecticiel selon un schéma entièrement répliqué : les agents *Client* et *Proxy* sont supprimés et la communication entre agents *Membre* est réalisée à travers le réseau. Cette approche a l'avantage de ne pas avoir à modifier le code de l'application puisque, par le biais de l'approche multi-agent, celui-ci est indépendant du code de la gestion du réseau.

La programmation d'un agent repose, comme le montre la portion de code de la Figure 11, sur l'implémentation de l'interface `Messenger`. Cette

```
public interface Messenger
{
 public boolean sendRecvMessage(Messenger sender,
                                Message message);
}


public class Message
{
 public String type;
 private Hashtable args;

 public void set(Object field, Object value)
 { [...] }

 public Object get(Object field) { [...] }

 public Object [] fields() { [...] }
}
```

Figure 11
Classes `Messenger` et `Message`

interface se limite à la déclaration de la méthode `sendRecvMessage` qui prend en paramètre un message et l'expéditeur du message. Ce dernier est du type `Messenger`. Un message est une instance de la classe `Message` définit par un type (chaîne de caractères identifiant la nature du message). Cette classe maintient un ensemble de couples `<attribut, valeur>` contenus dans une table de hachage (`Hashtable`). Il est possible d'accéder à cette table par le biais de deux méthodes, `set` et `get`, qui permettent respectivement d'ajouter ou de modifier la valeur d'un attribut, et de consulter la valeur d'un attribut. Une troisième méthode, la méthode `fields`, permet l'obtention de l'ensemble des attributs véhiculés par un objet de la classe `Message`.

```
Messenger agentPere;
Messenger agentCoord;
Message message = new Message(CONNECTION);

message.set(USER_NAME, "yann");
message.set(PASSWORD, cryptage("toto"));
message.set(USER_EMAIL, "yann@imag.fr");

if (! agentCoord.sendRecvMessage(this, message))
{
 /*
 ** problème de connexion
 */
 Message error = new Message(ERROR);

 error.set(ERROR_VALUE, CONNECTION_FAILED);
 error.set(ERROR_MSG, "connection impossible");
 agentPere.sendRecvMessage(this, error);
}
```

Figure 12
Exemple d'utilisation de la classe `Messenger`

La Figure 12 propose un exemple d'utilisation des classes `Messenger` et `Message`. Cette portion de code illustre la phase de connexion entre l'application et le serveur de coordination. Dans cet exemple, il s'agit du code d'un agent relié à un agent *Père* (agent `agentPere`) et à l'agent *Client coordination* (agent `agentCoord`). Ce dernier assure la communication avec le serveur de coordination. Le message échangé est un message de connexion et le type est `CONNECTION`. Ce message est composé de trois champs, le nom de l'utilisateur (`USER_NAME`), son mot de passe crypté (`PASSWORD`) et son adresse mél (`USER_MAIL`). Le message est envoyé à l'agent coordination (`agentCoord`) par l'appel de la méthode `sendRecvMessage`. En cas de problème de connexion, un message d'erreur est retourné à l'agent père (`agentPere`) en employant la même méthode de communication.

L'utilisation d'une approche multi-agent autorise une forte modularité du code et, comme nous l'avons souligné précédemment, facilite le passage d'un schéma d'exécution à un autre. De plus, la classe `Messenger` met en place un protocole d'échanges évolutif, comme l'ajout de nouveaux champs pour un type de message donné. En effet, cette approche permet aux systèmes existants de traiter des messages même s'ils ignorent

l'existence de nouveaux champs. Néanmoins, cette technique pourrait être améliorée en codant les champs (couples `<attribut, valeur>`) dans le formalisme *XML* (*eXtended Markup Language*) qui a l'avantage de permettre à des applications écrites dans un autre langage à pouvoir interagir avec notre plate-forme Clover. Actuellement, les messages qui transitent entre la partie cliente et la partie serveur sont codés (*serialization*) dans un format propriétaire à Sun Microsystems [Sun 2002]. Seules des applications écrites en Java peuvent interopérer avec notre plate-forme Clover.

3.3. RÉALISATION ET QUALITÉS LOGICIELLES

Pour conclure sur la réalisation logicielle de la plate-forme, nous dressons un bilan qualité du code de la plate-forme.

La plate-forme Clover vérifie la propriété de modularité. D'une part, la plate-forme met en œuvre trois serveurs dédiés à l'activité de groupe relatifs à la production, à la communication et à la coordination. Ces trois serveurs ont été développés et testés séparément. Ceci permet donc le développement d'un collecticiel en programmant indépendamment des portions de code spécifiques aux trois serveurs. Ceci a été vérifié lors du développement du collecticiel CoVitesse, exposé au chapitre suivant. D'autre part, l'approche multi-agent adoptée pour mettre en œuvre la plate-forme Clover renforce cette propriété de modularité. Notamment, cette approche permet le développement d'un collecticiel indépendamment de l'infrastructure réseau, entre autre, grâce aux agents *Client* et *Proxy*. De plus, nous avons pu observer que, par le biais de cette technique, il est aisé de mettre en œuvre différents schémas d'exécution. Pour l'instant, nous avons opté uniquement pour le schéma d'exécution hybride pour plusieurs raisons :

1. comme nous l'avons expliqué dans le Chapitre V, le schéma hybride est un bon compromis entre le schéma centralisé et répliqué puisqu'il permet à la fois de concilier une complexité réduite du développement et de bonnes performances,
2. un serveur est toujours plus sûr qu'un client [Patterson 1996] et semble indispensable pour un passage à l'échelle [Hall 1996].

Les deux autres propriétés logicielles de cette plate-forme sont la généricité et la flexibilité. La propriété de généricité est d'une part héritée de la généricité de la couverture fonctionnelle mise en œuvre à travers les différentes classes Java. D'autre part, l'utilisation du système d'interface du langage Java renforce cette propriété de généricité puisque la partie serveur est totalement indépendante du domaine de l'application. C'est le serveur de classes et d'objets qui se charge d'augmenter les trois serveurs dédiés à l'activité de groupe à l'exécution pour que ceux-ci puissent manipuler les concepts du domaine d'application. De plus, ce mécanisme

augmente la propriété de flexibilité de la plate-forme étant donné que le chargement dynamique de nouvelles classes augmente les capacités fonctionnelles de la partie serveur à l'exécution.

Dans cette partie, nous avons présenté les principaux aspects de la mise en œuvre logicielle de notre plate-forme Clover. Nous avons déjà dressé un bilan sur la qualité logicielle de la plate-forme dans le dernier paragraphe. Il convient maintenant de positionner ce travail vis-à-vis de notre grille d'analyse.

4. Plate-forme Clover et grille d'analyse

Comme nous l'avons fait dans le chapitre précédent pour des outils existants, nous analysons notre plate-forme Clover à la lumière de notre grille d'analyse. Nous considérons successivement les quatre dimensions de notre grille, dans les paragraphes suivants.

4.1. ACTIONS COLLECTIVES ET INDIVIDUELLES

La plate-forme Clover intègre explicitement les deux notions d'actions collectives et individuelles :

- **Actions collectives** : les trois aspects de l'action collective (production, communication et coordination) sont traités de façon explicite par la plate-forme Clover. En effet, nous avons défini un modèle de couverture fonctionnelle de l'activité de groupe à partir duquel nous avons déduit trois couvertures fonctionnelles dédiées à la production, à la communication et à la coordination. Ces trois couvertures ont ensuite été traduites sous forme de classes et interfaces Java.

- **Actions individuelles** : le modèle de couverture fonctionnelle de l'activité de groupe considère le concept de participant à l'activité de groupe. Comme nous l'avons expliqué, cette notion de participant réserve un espace à l'action individuelle. Cependant, la plate-forme n'offre pas de fonctions particulières à cet espace individuel, puisque celles-ci dépendent entièrement du domaine d'application du collecticiel.

4.2. RESSOURCES DU CONTEXTE

Les deux types de ressources du contexte, collectives et indiviudelles, sont présentes dans la plate-forme Clover :

- **Ressources collectives** : les ressources collectives sont traduites dans la plate-forme par le concept de support à l'activité de groupe.

- **Ressources individuelles** : les ressources individuelles relèvent essentiellement de la partie cliente. Cependant, à travers le concept de participant, la plate-forme réserve un espace aux ressources individuelles.

4.3. OBSERVABILITÉ DES ACTIONS

Par rapport à cette dimension, notre plate-forme est partiellement satisfaisante :

- **Actions collectives** : la rétroaction de groupe est assurée. D'une part, la partie serveur met en œuvre un service de notification qui propage les changements d'états des supports à l'activité de groupe. D'autre part, des fonctions ont été conçues pour consulter l'état de ces supports. Par contre, cette plate-forme n'offre aucun support pour mettre en œuvre le couplage et tout le développement est laissé à la charge du développeur. Nous illustrons ce cas avec le développement de CoVitesse au chapitre suivant : pour un des types de navigation, la visite guidée, un couplage fort entre les participants a du être développé.

- **Actions individuelles** : la conscience de groupe est mise en œuvre par le service de notification qui diffuse les informations relatives aux actions individuelles. De plus, des fonctions permettent de consulter les données relatives à un participant.

4.4. OBSERVABILITÉ DES RESSOURCES

L'observabilité des ressources collectives et individuelles est assurée par l'introduction du concept de filtre qui définit une politique de filtrage. Ainsi, des données peuvent être observables à un moment donné puis inaccessibles suite à une modification de la politique de filtrage. Cependant, la spécification de la notion de filtrage au sein de la plate-forme est incomplète et il convient de l'améliorer. En effet, une trop grande partie de la mise en œuvre d'une politique de filtrage est laissée à la charge du développeur. La plate-forme pourrait proposer un ensemble de filtres prédéfinis.

4.5. CONCLUSION

Tandis que le paragraphe précédant qualifiait la plate-forme en termes de propriétés logicielles, ce paragraphe met l'accent sur l'utilisabilité des collecticiels produits avec la plate-forme, en analysant les fonctions de la plate-forme vis-à-vis des dimensions de notre grille. Cette analyse montre qu'un ensemble d'éléments liés à la conception du collecticiel ainsi que des propriétés d'utilisabilité comme la rétroaction de groupe sont pris en compte dans la plate-forme. Néanmoins nous avons identifié des améliorations possibles de la plate-forme, comme les filtres de publication.

5. *Résumé des contributions*

Dans ce chapitre, nous avons présenté les spécifications fonctionnelles puis la réalisation logicielle de la plate-forme Clover. Un bilan qualité en termes de propiétés logicielles et d'utilisabilité a ensuite été établi. Nous retenons de ce chapitre les points contributifs suivants :

- **Définition, description et réalisation d'un modèle de couverture fonctionnelle de l'activité de groupe :** le modèle repose sur trois concepts clés : l'espace de l'activité de groupe, les participants à l'activité de groupe et le support à l'activité de groupe. Le modèle, décrit en UML, a ensuite été instancié aux trois aspects de l'activité de groupe : production, communication et coordination. Les trois modèles UML obtenus ont enfin été traduits en un ensemble de classes et interfaces Java qui constituent notre plate-forme Clover.

- **Réalisation logicielle de la plate-forme Clover** qui met en œuvre la couverture fonctionnelle ainsi définie. La plate-forme, écrite en Java, repose sur notre modèle d'architecture conceptuelle Clover. En particulier, on retrouve dans l'interface de programmation les trois couvertures fonctionnelles identifiées. La plate-forme intègre trois serveurs pour gérer l'activité de groupe : un serveur dédié à la production, un autre à la communication et un troisième à la coordination. La modifiabilité, la généricité et la flexibilité sont les principales propriétés de qualité logicielle de la plate-forme. Les propriétés d'utilisabilité des collecticiels produits avec la plate-forme ont été abordées lors de l'analyse de la plate-forme au regard de notre grille.

Pour illustrer l'utilisation de la plate-forme, nous présentons dans le chapitre suivant deux exemples de réalisation : le système CoVitesse et un système de tableau blanc partagé.

Chapitre VII *Illustration : le système CoVitesse et un tableau blanc partagé*

1. Introduction

A l'aide du modèle d'architecture Clover (Chapitre IV) et de la plate-forme Clover, présentée dans le chapitre précédent, nous avons mis en œuvre deux exemples de collecticiel : le système CoVitesse, système de navigation collaborative sur le *World Wide Web* (*WWW*), et un tableau blanc partagé. Une première version du système CoVitesse a été développée dans le cadre de travaux de DEA [Laurillau 1999a][Laurillau 1999c] et nous avons décidé de le reconcevoir puisqu'il fournit un exemple complet pour illustrer l'utilisation de la plate-forme Clover. Ce redéveloppement a permis, notamment, de prendre en compte un ensemble de recommandations issues d'une évaluation empirique de la première version du système, d'améliorer son utilisabilité et de réviser les différents types de navigation collaborative. L'exemple du tableau blanc partagé est beaucoup plus simple puisqu'il n'a pas fait l'objet d'une conception approfondie. Ce dernier est présenté pour illustrer la facilité de mise en œuvre à l'aide de la plate-forme et pour montrer l'interopérabilité simultanée de cette plate-forme avec différents collecticiels.

Ce chapitre est organisé de la façon suivante : la première partie est consacrée au système CoVitesse et nous détaillons son utilisation, son architecture logicielle et sa programmation. La seconde partie est consacrée au système de tableau blanc partagé et présente les principaux points de sa réalisation logicielle. La troisième partie présente les tests qui ont été menés pour évaluer la robustesse de la plate-forme Clover. La dernière partie présente un résumé des contributions de ce chapitre.

2. *CoVitesse*

Dans cette partie, nous présentons le système CoVitesse sous divers angles. Dans le premier paragraphe, nous présentons son utilisation à travers ses principales fenêtres afin de mieux comprendre la mise en œuvre logicielle. Dans le second paragraphe, nous présentons l'architecture logicielle du système. Le troisième paragraphe expose les points principaux de sa programmation.

2.1. DESCRIPTION DU SYSTÈME

Le système CoVitesse [CoVitesse 2002][Laurillau 1999a] est un système de navigation collaborative synchrone sur le *World Wide Web* (*WWW*). Ce système repose sur le système Vitesse [Vitesse 2002][Nigay 1998] qui propose différentes modalités en sortie pour la représentation d'une grande quantité d'information. L'espace d'information considéré est l'espace des réponses d'une requête (un ensemble de mots clés) soumise à un moteur de recherche. Le système CoVitesse est la version collaborative du système Vitesse et permet ainsi de mettre en relation simultanément plusieurs utilisateurs partageant un même espace d'information.

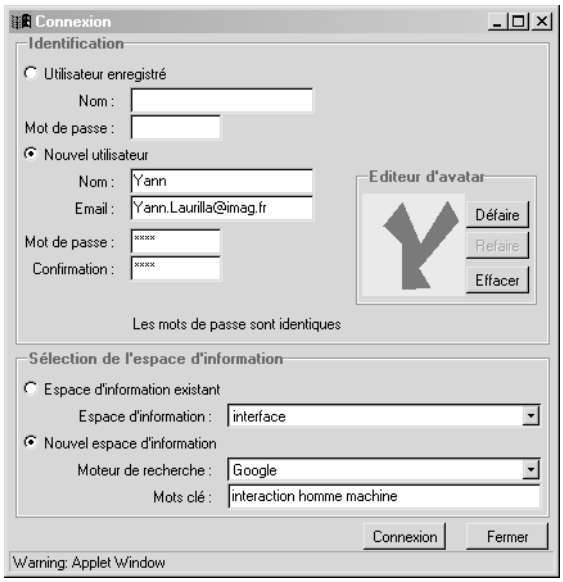


Figure 1
Fenêtre de connexion du système CoVitesse
Pour se connecter, un utilisateur a la possibilité de joindre une session rapidemment en donnant son nom et son mot de passe s'il est déjà enregistré. Dans le cas d'une première inscription, l'utilisateur doit donner un nom, éventuellement une adresse électronique pour recevoir les résultats collectés au cours d'une session, un mot de passe (deux fois pour confirmation) et un avatar dessiné à l'aide d'un petit éditeur de dessin. Ensuite, l'utilisateur sélectionne un espace d'information soit en choisissant un espace d'information existant, soit en créant un nouvel espace en soumettant une requête à un moteur de recherche sur le *WWW*. Dès lors, l'utilisateur peut joindre la session en cours.

Accès à un espace d'information

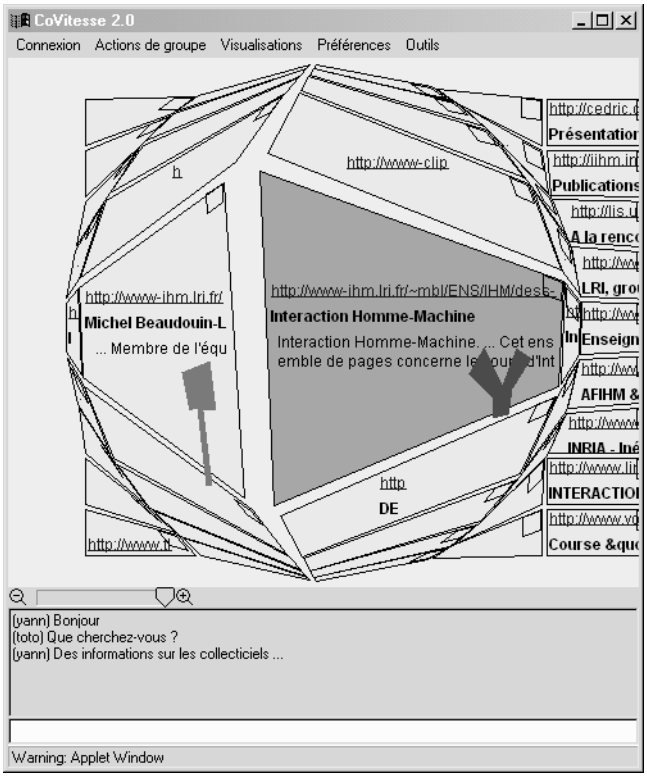
Au démarrage du système, comme le montre la Figure 1, l'utilisateur dispose d'une fenêtre de connexion nécessitant un nom d'utilisateur et d'autres informations s'il n'a pas encore été enregistré. Pour le représenter dans l'espace d'information, l'utilisateur doit dessiner un avatar. De plus, il doit préciser l'espace d'information dans lequel il désire naviguer, soit en spécifiant une requête soumise à un moteur de recherche, soit en sélectionnant un espace pré-existant. Dès qu'un nouvel espace d'information est créé, celui-ci est mémorisé de manière persistante sur un serveur : cela permet d'accélérer l'accès à l'espace d'information pour les futurs arrivants ; ainsi, il n'est pas nécessaire de

soumettre à nouveau une requête au moteur de recherche et d'analyser les résultats pour les mettre en forme. Les données collectées au cours d'une session sont aussi mémorisées et récupérables pour les sessions suivantes. Pour déterminer quels sont les espaces d'information disponibles, le système interroge le serveur sur lequel sont stockées les données.

Navigation dans l'espace d'information

Dès que la phase de connexion est terminée, l'utilisateur pénètre dans l'espace d'information sélectionné. L'espace d'information, comme le montre la Figure 2, organise spatialement les résultats du moteur de recherche sous forme d'un agencement de polygone (un polygone équivaut à une page web). Dans cet espace, un utilisateur est représenté par un avatar qui se déplace dès que l'on sélectionne une page. C'est dans cette zone que sont représentés tous les avatars : le mouvement des avatars traduit ainsi l'activité de navigation, qu'elle soit individuelle ou en groupe. Par défaut, les utilisateurs n'appartiennent à aucun groupe ; ceux-ci sont identifiables par la couleur bleu de l'avatar (par exemple, le T de thomas dans la Figure 2). Les membres d'un groupe sont identifiés par une autre couleur (unique) comme, par exemple, le Y rouge de Yann dans la Figure 2. Dans la partie située sous l'espace d'information, l'utilisateur dispose d'un forum de discussion pour pouvoir échanger avec les autres utilisateurs. Cet agencement permet aux utilisateurs à la fois de continuer leur activité de navigation et de percevoir l'activité de communication de façon périphérique, et réciproquement. Suite à une évaluation empirique de la première version du système CoVitesse, il est apparu indispensable

Figure 2
Fenêtre principale du système CoVitesse
L'espace d'information, qui occupe une grande partie de la surface de la fenêtre, est constitué d'un ensemble de polygones. Chaque polygone représente une page web en affichant plusieurs types d'informations : l'adresse, le titre et un court résumé. Cet espace est construit à partir des résultats issus d'une requête soumise à un moteur de recherche. Lorsque l'utilisateur clique deux fois sur un polygone, la page web correspondante est affichée avec le navigateur web associé : l'avatar de l'utilisateur est automatiquement déplacé sur le polygone pour indiquer sa position courante dans l'espace d'information. Ainsi, le déplacement de tous les avatars indique le taux d'activité dans l'espace. A tout moment, l'utilisateur peut cocher cette page (case située en haut à droite de tous les polygones) pour ajouter celle-ci à la liste des références collectées.
Le système propose trois modalités de visualisation : la vue à plat (*bird eye view*), la vue en sphère et la vue en colline. A tout moment, l'utilisateur peut changer de modalité par l'intermédiaire du menu "visualisation" sachant que chaque utilisateur dispose de sa propre vue. Certaines vues peuvent être ajustées à l'aide d'un zoom réglable situé juste en dessous de l'espace d'information.



de placer ce forum de discussion sous l'espace d'information pour réduire les discontinuités au cours de l'interaction.

Dans le menu principal, le système met à disposition un ensemble d'actions et d'outils. Il existe deux sortes d'actions et d'outils :

- *des actions et outils individuels* :
  * changer de visualisation,
  * réorganiser des références collectées,
  * modifier les préférences personnelles.
- *des actions et outils de groupe* :
  * créer/joindre un groupe,
  * quitter un groupe,
  * réorganiser les références collectées par le groupe,
  * modifier les préférences du groupe,
  * exécuter une action spécifique à un groupe et à un rôle.

Création d'un groupe

Un utilisateur, s'il n'est pas déjà membre d'un groupe, peut créer son propre groupe en spécifiant, comme le montre la Figure 3, un nom, une couleur et un type de navigation. Nous avons identifié quatre types de navigation définis comme suit [Laurillau 2000] :

- **Visite guidée** : Inspirée de la visite guidée d'un musée ou d'un site, les utilisateurs membres du groupe cèdent le contrôle de l'interaction à un

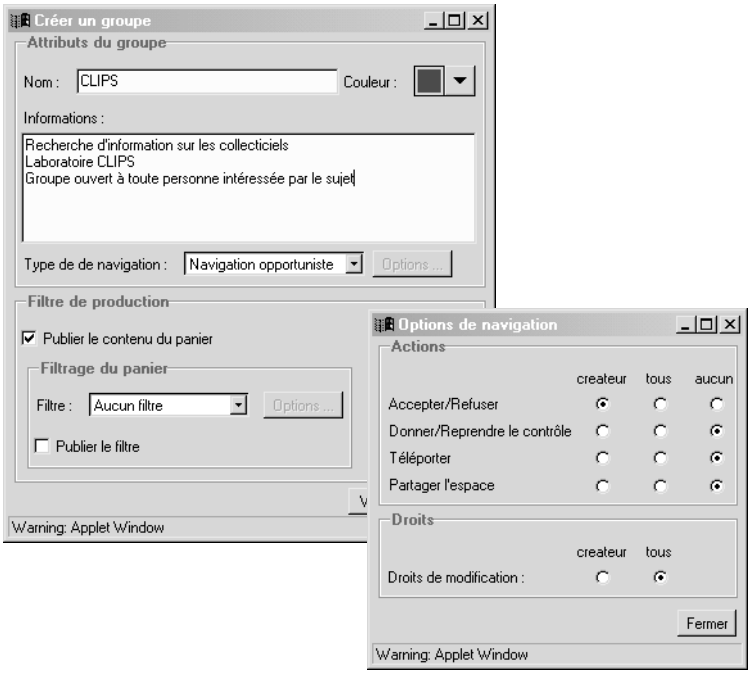


Figure 3
Fenêtres de création de groupe et de paramétrage de la navigation
Pour créer un groupe, un utilisateur doit impérativement choisir un nom, une couleur et un type de navigation. En outre, le créateur peut préciser la nature du groupe en donnant des informations complémentaires et choisir une politique d'observabilité en choisissant et en paramétrant des filtres de publication. De plus, il est possible de rendre observable le choix de filtrage.
La navigation personnalisée est paramétrable à l'aide d'un panneau de configuration activé dès que l'on clique sur le bouton "option". Ce panneau permet au créateur de réaliser plusieurs combinaisons pour l'attribution de droits d'exécution d'action et de modification des données du futur groupe. De plus, il est possible, en fonction des droits définis au démarrage, de modifier ces attributions dynamiquement au cours de la session.

guide qui les emmène découvrir l'espace d'information. A tout moment un utilisateur peut joindre ou quitter la visite guidée.

- **Navigation opportuniste** : La rencontre est informelle, et chacun navigue indépendamment les uns des autres, mais dispose des résultats collectés par les membres du groupe. N'importe qui peut devenir un membre et, à n'importe quel moment, il est possible de donner le contrôle de la navigation à un autre.

- **Navigation coordonnée** : Les membres du groupe travaillent séparément, dans un espace restreint ou non, sur un sujet commun, mais sans forcément avoir exactement les mêmes buts. Tous ont les mêmes droits, et n'importe qui peut décider si un nouveau venu peut devenir ou non un membre.

- **Navigation coopérative** : Un responsable de groupe oriente la recherche. Il lui incombe la responsabilité de définir les objectifs et les stratégies à suivre, de manière autoritaire ou concertée, et de répartir, si cela est nécessaire, les sous-espaces à explorer. Il est le seul à décider si un nouveau venu peut devenir membre. Il a de plus la possibilité de déplacer ("téléporter") l'ensemble du groupe dans un endroit qu'il peut juger intéressant.

Ces quatre types de navigation ont été identifiés à partir de différentes études ethnographiques et caractérisées à l'aide du modèle Denver (présenté dans le Chapitre II). Tous les détails concernant ces types de navigation sont donnés en annexe.

Pour rendre le système plus flexible, nous avons défini un cinquième type de navigation, la navigation personnalisée, qui permet la création de son propre type de groupe. Le paramétrage s'articule, comme le montre la Figure 3, sur l'allocation :

- d'actions de groupe :
  * accepter/refuser un nouveau membre,
  * donner/reprendre le contrôle de la navigation
  * téléporter tous les membres du groupe dans une zone quelconque de l'espace
  * imposer le partage automatique de l'espace,

- de droits d'accès aux données appartenant au groupe :
  * modifier les préférences du groupe,
  * modifier le panier du groupe des références collectées.

Dans la version actuelle, nous considérons deux rôles uniquement, le chef de session assimilé au créateur du groupe et le membre du groupe. Pour chaque type d'action et de droit d'accès, il est donc possible de choisir de restreindre le droit au chef de session ou de l'étendre à tous les membres.

Enfin, durant la phase de création, il est possible de spécifier une politique d'observabilité des données en choisissant de publier ou non les données collectées. De plus, il est possible de publier ou non les informations relatives à la politique de filtrage.

action individuelle et action collective

Le système CoVitesse gère à la fois l'action individuelle et l'action collective. Cependant, l'espace réservé à l'action individuelle peut varier au cours d'une session ce qui se traduit par des variations du couplage de l'interaction. Ce degré s'applique uniquement lorsqu'un utilisateur est membre d'un groupe. Ainsi, lorsqu'un utilisateur effectue une navigation individuelle, le degré de couplage est nul.

Par exemple, dans le cas de la visite guidée, le degré de couplage est relativement fort puisque le guide contrôle la navigation de tous les membres. Cependant, un membre peut toujours modifier la vue courante sur l'espace d'information en changeant de modalité, discuter à travers le forum de discussion et réorganiser sa liste personnelle de résultats collectés. Il en est de même lorsqu'un membre cède le contrôle de la navigation à un autre membre comme, par exemple, dans le cas de la navigation opportuniste. En l'occurence, le fait de céder le contrôle de sa propre navigation modifie dynamiquement le degré de couplage de l'interaction. Dans les autres cas, le degré de couplage est peu élevé : il

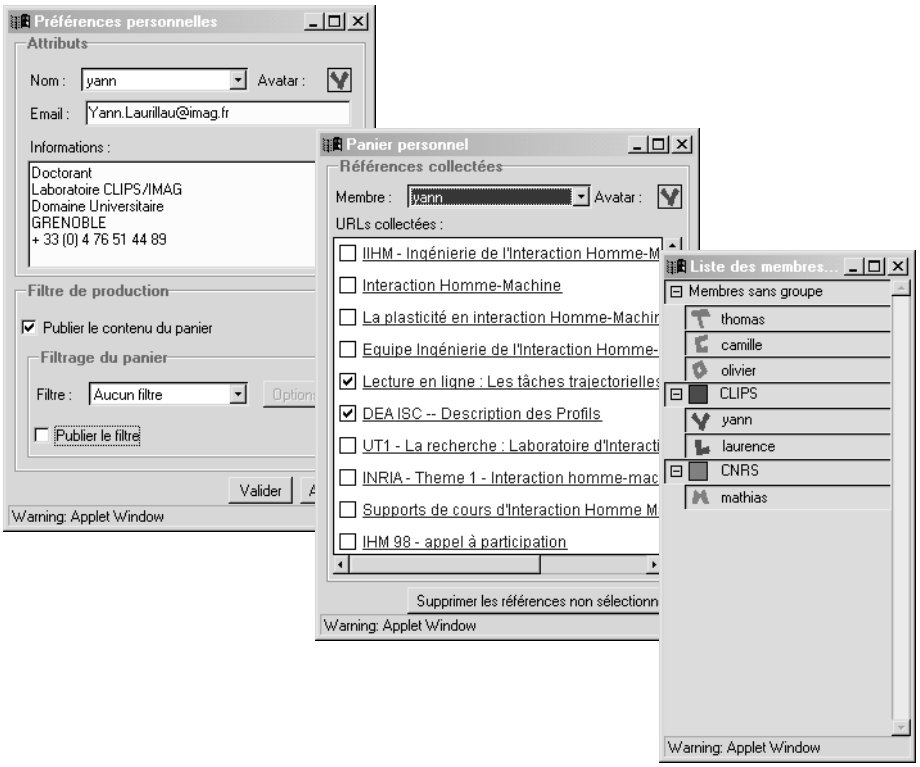


Figure 4
Fenêtres de paramétrage des préférences personnelles, du panier personnel et de la liste des membres et des groupes
L'utilisateur dispose d'une fenêtre pour effectuer le réglage de ses préférences personnelles. Il dispose d'une autre fenêtre pour éditer la liste des références collectées pour éventuellement en réorganiser le contenu (suppression de référence). Enfin, une troisième fenêtre lui permet de connaître la liste des groupes et utilisateurs présents dans l'espace d'information : il est possible de masquer ou afficher (filtrage) certains avatars ; il suffit de cliquer sur l'avatar correspondant pour le faire apparaître ou disparaître.

peut être ponctuellement fort lorsque le chef de session, dans le cas d'une navigation coordonnée, "téléporte" l'ensemble des membres dans une zone de l'espace d'information.

En terme d'action individuelle, hormis la navigation dans l'espace d'information et la communication à travers le forum de discussion, un utilisateur peut, comme le montre la Figure 4, modifier ses préférences, réorganiser les références collectées (personnelles) ou changer sa politique d'observabilité. De plus, il est possible de consulter les données relatives aux autres utilisateurs.

La réorganisation des données collectées consiste à les retirer de son "panier". Dans la version actuelle, il n'est pas encore possible d'opérer des regroupements par thème par exemple. Cependant, il est possible de consulter la liste des résultats collectés par les autres utilisateurs, en fonction des droits de publication.

En terme d'action collective, en dehors, de la création de groupe et des actions spécifiques à un groupe, un utilisateur peut toujours accéder aux actions permettant de joindre, quitter un groupe et modifier les données relatives au groupe en fonction des droits accordés à son rôle.

Ayant présenté l'aspect utilisation du système, nous présentons dans le paragraphe suivant l'architecture logicielle du système.

2.2. ARCHITECTURE LOGICIELLE

Dans ce paragraphe, nous détaillons l'architecture logicielle du système. Dans la première section, nous présentons l'architecture globale de CoVitesse puis nous détaillons, dans les sections suivantes, l'architecture de la partie cliente du système (Noyau Fonctionnel, Adaptateur du Noyau Fonctionnel et Contrôleur de Dialogue).

Architecture logicielle globale

La Figure 5 présente l'architecture logicielle du système CoVitesse. Pour élaborer cette architecture logicielle, nous avons appliqué notre modèle d'architecture Clover, présenté au Chapitre IV, composé des Trèfles Fonctionnels publics et privés. Au niveau implémentationnel, CoVitesse est écrit en Java et son architecture implémentationnelle est de type client/ serveur. Le Trèfle Fonctionnel public est centralisé sur le serveur et le Trèfle Fonctionnel privé est décomposé en deux parties, une au niveau du serveur et l'autre au niveau du client.

Nous décrivons l'architecture en partant de la couche la plus haute vers la couche la plus basse. Au plus haut niveau d'abstraction, le Trèfle Fonctionnel public, est composé d'un NF dédié à la production (NF_{prod}), d'un NF dédié à la communication (NF_{comm}), d'un NF dédié à la coordination (NF_{coord}) et d'une interface logicielle notée méta-serveur.

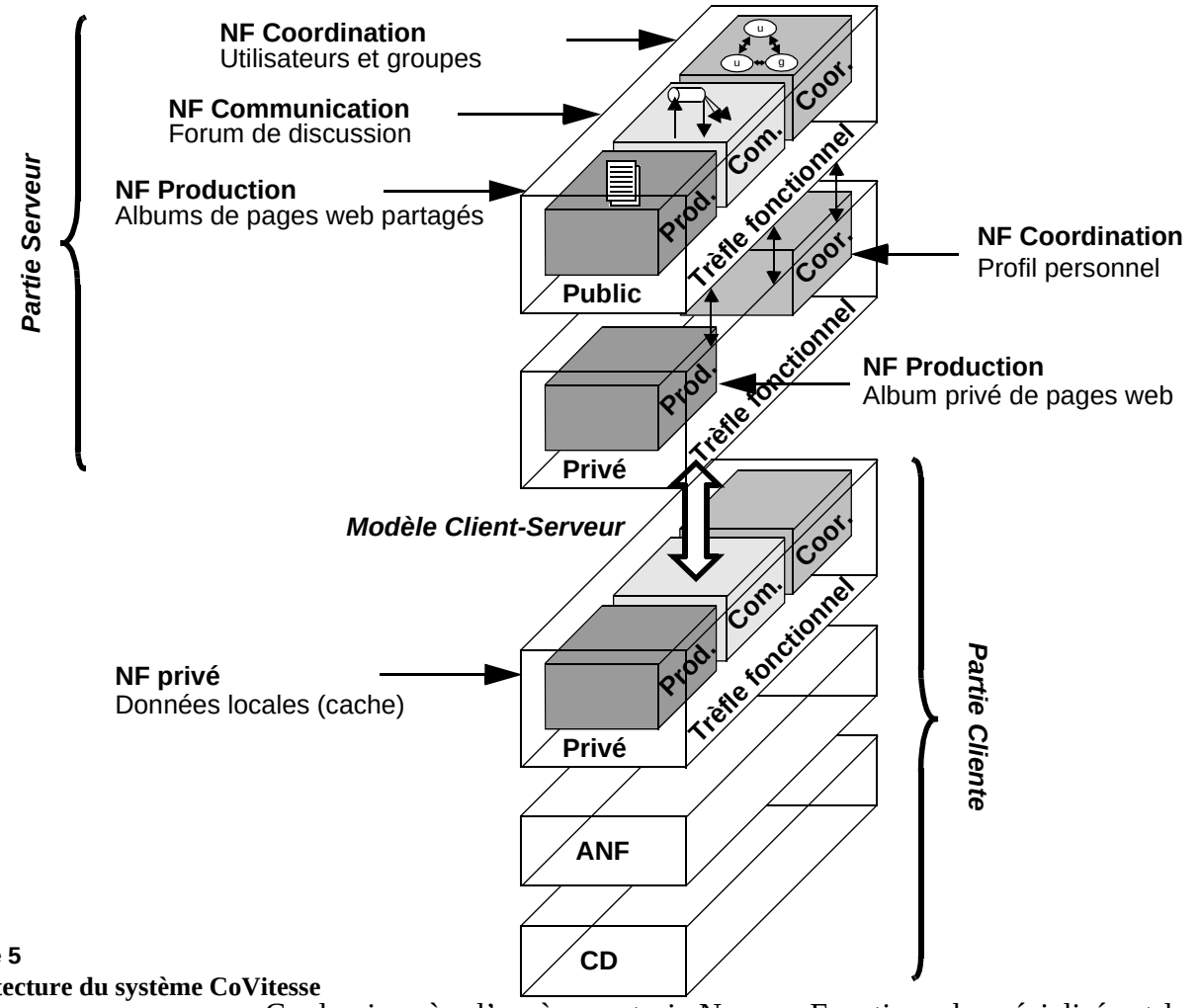


Figure 5
Architecture du système CoVitesse

Ce dernier gère l'accès aux trois Noyaux Fonctionnels spécialisés et les appels à l'infrastructure réseau.

- **Composant public dédié à la production (NFprod)** : ce composant maintient l'ensemble des documents partagés, représentés par des pages à la Figure 5. Les principaux concepts manipulés sont les documents, les références, les albums et les catalogues. Un document, associé à une référence, est l'entité d'information élémentaire manipulée par les utilisateurs. Un album, créé et géré par des auteurs, contient des documents et des références. Un catalogue est une collection de références. De nombreuses fonctions sont disponibles telles que créer ou récupérer un catalogue, créer un album, ajouter ou retirer un auteur d'un album, ajouter un document dans l'album, modifier le contenu d'un album. Dans CoVitesse, une référence est une URL et un document, le contenu de la page web, c'est-à-dire un document HTML. Un catalogue est un ensemble d'URL construit à partir de l'ensemble des réponses d'une requête soumise à un moteur de recherche. L'album est aussi un ensemble d'URL.

- **Composant public dédié à la communication (NFcomm)** : ce composant gère différents canaux de communication, représentés par un tube à la Figure 5. Le canal de communication est l'unique concept manipulé par le NF_{comm}. Un client peut s'abonner ou se désabonner à un canal de communication et peut y poster des messages. Dans CoVitesse, l'unique canal de communication est celui du forum de discussion textuel, comparable à une liste de diffusion : quand un utilisateur poste un message sur un canal, celui-ci est diffusé à l'ensemble des abonnés.

- **Composant public dédié à la coordination (NFcoord)** : Ce composant a la responsabilité de coordonner les utilisateurs et les groupes. Il dispose de mécanismes gérant les accès concurrents aux données partagées comme les préférences d'un groupe et notifie tous les clients des changements d'état. Les principaux concepts manipulés par NF_{coord} sont les concepts d'utilisateur et de groupe :

  * *Utilisateur* : deux fonctions permettent respectivement de lire et de modifier les informations personnelles d'un utilisateur, tandis que deux autres fonctions sont dédiées à la lecture et à la modification des droits d'accès à ces informations. La réalisation actuelle offre la possibilité de définir son propre concept d'utilisateur en fonction du collecticiel. Dans CoVitesse, un utilisateur est identifié par un nom et un mot de passe. La description d'un utilisateur contient aussi une forme géométrique, des préférences de filtrage, une adresse mél, etc.

  * *Groupe* : un groupe est défini par un ensemble de membres, des droits d'accès au groupe et un ensemble minimal de fonctions. Ces fonctions permettent de modifier ou d'obtenir les informations relatives à un groupe, de créer et de joindre un groupe (un utilisateur soumettant une requête à un des membres pour une acceptation éventuelle), de quitter un groupe. Dans CoVitesse, un groupe est défini par un nom, une couleur, des préférences de filtrage et un type de navigation.

Comme le montre Figure 5, le Trèfle Fonctionnel public est empilé au-dessus du Trèfle Fonctionnel privé. Ce dernier est, au niveau implémentationnel, divisé en deux parties, l'une au niveau client et l'autre au niveau du serveur. Le Trèfle Fonctionnel privé localisé sur le client maintient un état local à l'application et une copie de l'état privé située sur le serveur. Un mécanisme de cache permet d'optimiser les performances à l'exécution. Le Trèfle Fonctionnel privé situé sur le serveur est constitué de deux sous-composants, dédiés à la production et à la coordination (NF_{prod} et NF_{coord}). Ces composants maintiennent les données privées d'un utilisateur. L'implémentation actuelle du système permet de conserver ces données de manière persistante sur le serveur. D'une part, le

NF_{prod} privé gère l'album privé de l'utilisateur contenant les URL qu'il a collectés au cours de la session. De plus ce composant a accès à l'album du groupe, maintenu et protégé par le NF_{prod} public. D'autre part, le NF_{coord} privé maintient les préférences personnelles de l'utilisateur. Les deux NF_{coord}, privés et publics, manipulent les mêmes données relatives à l'utilisateur. Cependant, le composant privé est l'unique instance autorisée à modifier les données privées comme le mot de passe. Le composant public peut uniquement lire ces données quand celles-ci sont publiées.

Dans la version actuelle de CoVitesse, il n'y a pas de sous-composant dédié à la communication dans le Trèfle Fonctionnel privé. Par conséquent, l'interface logicielle de ce Trèfle Fonctionnel transfère directement les événements dédiés à la communication entre le Trèfle Fonctionnel public et l'Adaptateur du Noyau Fonctionnel. Cependant, des développements sont en cours pour intégrer la facette manquante. Ce composant gérera par exemple des listes de destinataires (*aliases*). De plus, nous avons prévu d'étendre le sous-composant dédié à la communication du Trèfle Fonctionnel public pour permettre la discussion uniquement entre membres d'un même groupe.

Les autres couches de l'architecture de CoVitesse ne sont pas décomposées selon les trois facettes du trèfle des collecticiels. Ces couches sont au nombre de quatre : l'Adaptateur du Noyau Fonctionnel (ANF), le Contrôleur de Dialogue (CD) et les deux composants Interaction Logique et Physique. Ces deux derniers composants ne sont pas dessinés à la Figure 5, ils se situent en dessous du Contrôleur de Dialogue.

Architecture de la partie cliente

La partie cliente, comme le montre la Figure 5, est composée d'une partie du Trèfle Fonctionnel privé, de l'Adaptateur du Noyau Fonctionnel et du Contrôleur de Dialogue. Les Composants Logique et Physique existent mais ne jouent pas un rôle essentiel dans ce contexte. Pour développer la partie cliente, nous avons adopté l'approche multi-agent telle que nous l'avons définie dans le paragraphe 3.1 du chapitre précédent. L'architecture conceptuelle de cettte partie cliente, comme le montre la Figure 6, est constituée d'une hiérarchie d'agents sachant que l'agent *Adaptateur du Noyau Fonctionnel* relie le Noyau Fonctionnel, via l'agent *Noyau Fonctionnel*, avec le Contrôleur de Dialogue, via l'agent ciment *CoVitesse* (racine de la hiérarchie d'agent constituant le Contrôleur de dialogue). Cette architecture a été élaborée dans l'esprit du modèle PAC (voir Chapitre II).

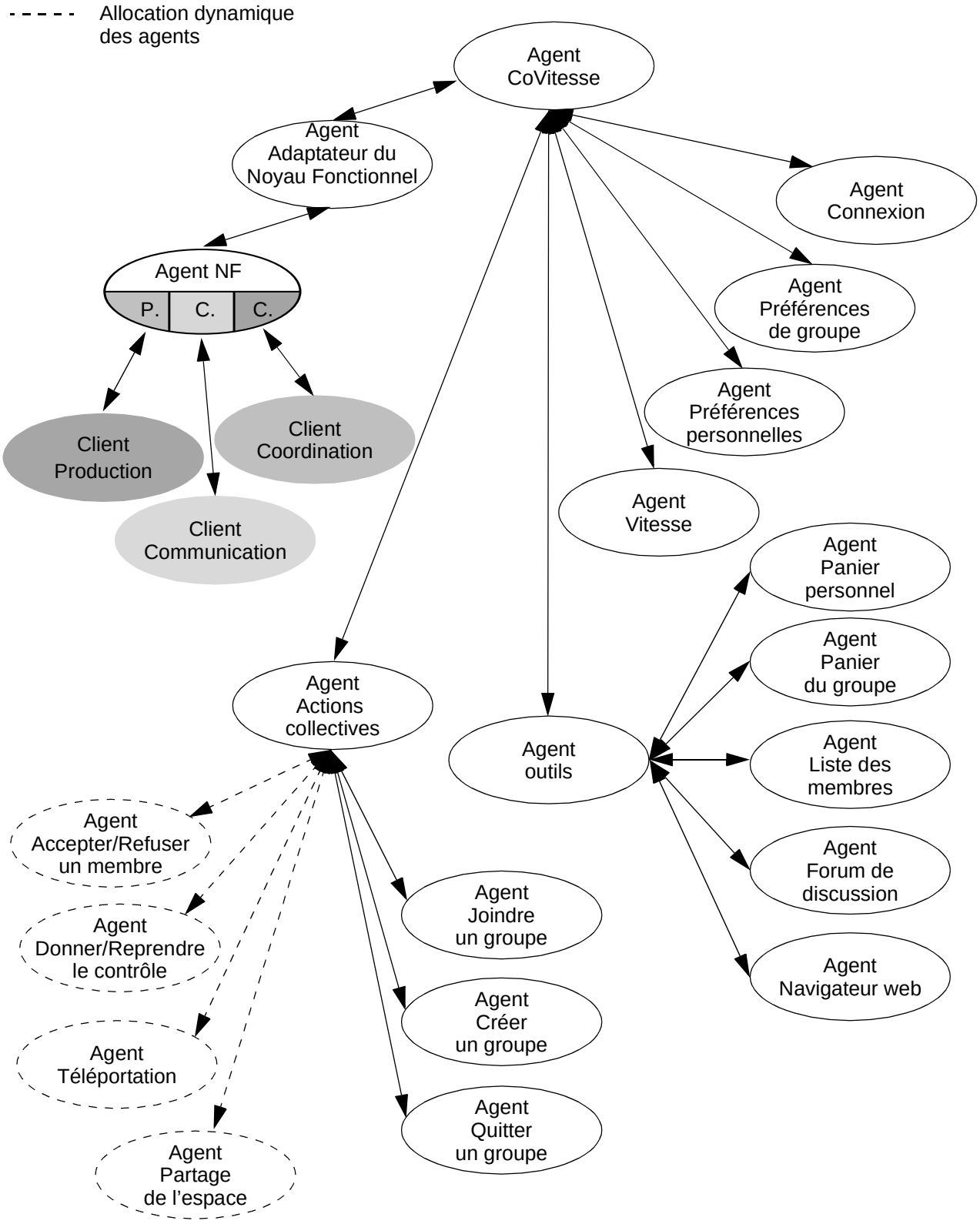


Figure 6
Hiérarchie d'agents constituant l'application cliente CoVitesse
Détail du Noyau Fonctionnel, de l'Adaptateur du Noyau Fonctionnel et du Contrôleur de dialogue.

Noyau Fonctionnel

Le Noyau Fonctionnel de la partie cliente est composé de quatre agents : l'agent *Noyau Fonctionnel* et trois agents *Clients* dédiés à la production, à la communication et à la coordination permettant l'accès aux serveurs associés. Ces agents clients donnent l'illusion à l'agent *Noyau Fonctionnel* de communiquer directement avec son agent fils situé sur le serveur, c'est-à-dire l'agent *Membre*, l'agent *Auteur* et l'agent *Abonné*. Les agents *Clients* encapsulent la communication par le réseau et rendent transparent l'accès aux agents centralisés sur les serveurs.

L'agent *Noyau Fonctionnel*, comme le montre la Figure 6, est un composant Clover tel que nous l'avons défini dans le Chapitre IV (modèle d'architecture Clover) : l'agent est constitué de trois sous-composants dédiés à la production, à la communication et à la coordination. De plus, ces trois sous-composants communiquent directement avec l'agent *Client* correspondant. Enfin, une abstraction encapsule ces trois composants. Celle-ci a la charge de faire le lien avec la partie dialogue de l'application.

La sémantique de l'application, à savoir la navigation collaborative, est définie par la partie cliente du système CoVitesse. L'interopérabilité avec la plate-forme Clover est assurée par le serveur de classes et d'objets qui se charge d'acheminer les définitions des classes implémentant les concepts relatifs à la navigation collaborative. Ceci permet de plus à différentes applications de reposer sur les mêmes instances de serveurs à condition qu'il n'y ait pas de conflit de nom de classe (par exemple, conflits liés à l'utilisation simultanée de différentes versions d'une même classe).

Adaptateur du Noyau Fonctionnel

Le rôle de l'agent *Adaptateur du Noyau Fonctionnel* est très simple. Celui-ci est chargé de faire la correspondance entre le concept d'utilisateur employé par le Noyau Fonctionnel et sa représentation graphique sous forme d'avatar utilisé par le Contrôleur de Dialogue.

Contrôleur de Dialogue

Le Contrôleur de Dialogue est constitué d'une hiérarchie d'agents organisée en trois catégories : les agents dédiés aux actions collectives, les agents dédiés aux outils et les autres agents : un agent *Vitesse* pour la navigation dans l'espace d'information, deux agents pour la gestion des préférences personnelles et du groupe (agents *Préférences personnelles* et *Préférence du groupe*), et un agent *Connexion* pour joindre ou créer une session.

Les agents dédiés à l'action collective sont gérés par un agent ciment, l'agent *Actions collectives*. Comme le montre la Figure 6, le nombre d'agents fils de l'agent ciment évolue dynamiquement au cours de l'interaction par ajout et suppression d'agents. Le nombre d'agents varie en fonction du rôle joué par un utilisateur au sein d'un groupe. Ainsi, lorsqu'un utilisateur rejoint un groupe, de nouvelles actions sont

disponibles et cela se traduit par l'apparition de nouveaux agents fils. Réciproquement, lorsque l'utilisateur quitte le groupe ou change de rôle, des agents fils sont détruits et retirés de la hiérarchie. Néanmoins, les trois agents suivants ne peuvent être supprimés : l'agent *Créer un groupe*, l'agent *Joindre un groupe* et l'agent *Quitter un groupe* qui correspondent aux actions collectives créer/joindre/quitter un groupe.

Les agents dédiés aux outils sont eux aussi gérés par un agent ciment, l'agent *Outils*. Les cinq agents fils sont : l'agent *Panier personnel*, l'agent *Panier du groupe*, l'agent Liste des membres, l'agent *Forum de discussion* et l'agent *Navigateur web*. Les agents *Panier* permettent la gestion des résultats collectés au cours de la navigation. Pour l'instant l'unique fonctionnalité disponible permet la suppression d'un ensemble de références. Il est prévu à terme d'ajouter d'autres fonctionnalités comme le regroupement de références. Toutefois, il est possible de consulter, en fonction des droits de publication, le panier des autres utilisateurs et des autres groupes.

L'agent *Navigateur web* est une encapsulation du navigateur web sous-jacent. En particulier, ce navigateur est celui qui a permis de télécharger l'applet CoVitesse. Toutefois, cette encapsulation autorise l'utilisation du système CoVitesse sous forme d'une application disposant de son propre module d'affichage de pages web.

L'agent *forum de discussion* gère un forum de discussion en mode texte. Cet agent dispose de deux facettes présentation : une est de taille fixe et réduite, située sous l'espace d'information ; l'autre est dans une fenêtre indépendante capable d'être agrandie afin de bénéficier d'un plus grand espace de dialogue.

Enfin, les agents *Préférences* gèrent les préférences personnelles et collectives. L'agent *Vitesse* gère la présentation et la navigation au sein de l'espace d'information. L'agent *Connexion* gère l'accès aux espaces d'information.

Dans ce paragraphe nous avons détaillé l'architecture du système. Le paragraphe suivant est consacré à l'aspect programmation et en présente les points principaux.

2.3. PROGRAMMATION LOGICIELLE

Dans ce paragraphe, nous présentons les points importants relatifs à la programmation du système CoVitesse. Ainsi, les premières sections détaillent la mise en œuvre des concepts pour la coordination et la production. La méthode consiste à programmer des classes Java qui implémentent les interfaces génériques définies par la plate-forme Clover. Afin d'illustrer le développement de ces concepts, nous présentons dans la suite la programmation des concepts relatifs à la coordination et à la production :

- *concepts pour la coordination* : utilisateur CoVitesse, groupe CoVitesse et type de navigation collaborative,

- *concepts pour la production* : album CoVitesse et référence de page web.

Dans la suite, les sections traitent dans l'ordre la programmation de ces concepts.

La dernière section détaille le fonctionnement de la machine à état du système CoVitesse pour l'aspect coordination.

```
public class CoVitesseUser  implements  User, java.io.Serializable
{

 public static final String INFO = "info";
 public static final String EMAIL = "email";
 public static final String AVATAR = "avatar";
 public static final String RESULTS_PUBLISHED = "ppub";
 public static final String FILTER_PUBLISHED = "fpub";
 private static final String [] fields = { NAME, PASSWORD, EMAIL,
                                           AVATAR, RESULTS_PUBLISHED,
                                           FILTER_PUBLISHED, INFO };


 public void set(Object field, Object value)
 {
  if (field != null)
  {
   if (value != null) { cache.put(field, value); }

  /* Champs à conserver : valeur booléenne */
   else if (cache.containsKey(field) && (! field.equals(NAME))
               && (! field.equals(RESULTS_PUBLISHED))
               && (! field.equals(FILTER_PUBLISHED)))
     { cache.remove(field); }
  }
 }
 public Object get(Object field, boolean filtered)
 {
  Object ret = null;

  if (field != null) && ((! filtered) || (! field.equals(PASSWORD)))
  {
    ret = cache.get(field);
  }

  return ret;
 }
[...]
}
```

Figure 7
Implémentation du concept d'utilisateur de CoVitesse

Concept d'utilisateur

Le concept d'utilisateur CoVitesse, comme le montre la portion de code Java dans la Figure 7, est défini par la classe `CoVitesseUser` et implémente l'interface `User`. Un utilisateur CoVitesse est caractérisé par :

- un nom d'utilisateur contenu dans le champ `NAME`. Ce champ est déclaré au niveau de l'interface `User`.
- un mot de passe identifiant l'utilisateur contenu dans le champ `PASSWORD`
- une adresse électronique pour contacter l'utilisateur et contenue dans le champ `EMAIL`.
- un ensemble d'informations complémentaires contenues dans le champ `INFO`.
- un avatar représentant l'utilisateur dans l'espace d'information et contenu dans le champ `AVATAR`. Cet avatar est une forme géométrique codée sous forme d'image. Le type du champ est un tableau d'entiers à deux entrées.
- un drapeau indiquant si les résultats collectés sont filtrés. Ce drapeau est codé sous la forme d'un booléen contenu dans le champ `RESULTS_FILTERED`.
- un drapeau indiquant si les informations de filtrage sont publiées. Ce drapeau est codé sous la forme d'un booléen contenu dans le champ `FILTER_PUBLISHED`.

Ces champs sont identifiés par des constantes de type chaine de caractères et les valeurs associées sont stockées dans une table de hachage (`Hastable`). Pour consulter ou modifier un champ, il suffit d'utiliser les fonctions `get` et `set`.

La fonction `set` prend deux arguments, le champ et la valeur correspondante. Si la valeur est nulle, le champ est supprimé de la table de hachage. Cependant, il n'est pas possible de supprimer le champ correspondant au nom de l'utilisateur et aux champs booléens relatifs à l'état du filtrage et aux règles de publications des résultats collectés.

La fonction `get` prend deux arguments, le champ dont on veut connaître la valeur et un booléen indiquant si la valeur retournée doit être filtrée ou non. Dans notre cas, seul le mot de passe est caché ; il ne peut être consulté que par l'utilisateur. A partir de l'identificateur de champ, la

fonction récupère et retourne la valeur correspondante stockée dans la table de hachage.

A cela s'ajoute un tableau, le tableau `fields`, qui contient la liste des champs spécifiques à cette classe. La fonction `fields` permet ainsi de questionner n'importe quel type d'objet héritant de la classe `User` et d'en connaître les champs qui le caractérisent. Cette technique permet la manipulation d'objets sans avoir à connaitre son type exact à la compilation. C'est cette mécanique qui est exploitée au sein de la plate-forme Clover et qui lui permet d'être indépendante des applications développées.

Enfin, ces objets sont transformables en une suite d'octets par un processus de "*serialisation*" (`Serializable`). Par conséquent, il est facile de faire migrer ces objets à travers une communication réseau ou dans un fichier sur disque. Néanmoins, le gros désavantage de ce mécanisme impose que les clients qui ne sont pas écrits en Java puissent lire ce format qui est un format propriétaire à Sun MicroSystems. A terme, il est envisagé de transcrire ces objets dans un format standard comme par exemple *XML* (*eXtended Markup Language*) pour permettre une plus grande interopérabilité avec les applications développées avec les autres langages.

```
public abstract class   CoVitesseGroup  implements  Group
{
 public static final String INFO = "info";
 public static final String TYPE = "type";
 public static final String COLOR = "color";
 public static final String FILTER_PUBLISHED = "fpub";
 public static final String RESULTS_PUBLISHED = "ppub";
 public static final String WRITABLE = "writable";
 public static final String FREEZE = "freeze";
 public static final String SHARED = "shared";
 protected static final String [] fields = { TYPE, COLOR, INFO,
                                             NAME, CREATOR,
                                             RESULTS_PUBLISHED,
                                             FILTER_PUBLISHED,
                                             WRITABLE, FREEZE, SHARED };
 protected CoVitesseGroup(String name)
 {
  cache.put(TYPE, name);
  /* initialisation des champs booléen avec une valeur par défaut */
  cache.put(RESULTS_PUBLISHED, Boolean.FALSE);
  cache.put(FILTER_PUBLISHED, Boolean.TRUE);
  cache.put(WRITABLE, Boolean.TRUE);
  cache.put(SHARED, Boolean.FALSE);
 }
[...]
}
```

Figure 8
Implémentation du concept d'un groupe CoVitesse générique

Concept de groupe et de navigation

Le concept de groupe CoVitesse est implémenté sous forme d'une classe abstraite (`CoVitesseGroup`) et de cinq classes implémentant les différentes navigations : la classe `OpportunisticNavigation`, la classe `SharedNavigation`, la classe `CoordinatedNavigation`, la classe `GuidedTour` et la classe `CustomizedNavigation`. Ces classes implémentent le même mécanisme de gestion de champ que la classe `CoVitesseUser`. Un groupe CoVitesse est caractérisé par, comme le montre la portion de code dans la Figure 8, les champs suivants :

- un nom de groupe contenu dans le champ `NAME`.
- un créateur (un utilisateur CoVitesse) à l'origine du groupe référencé par le champ `CREATOR`.
- un type de navigation contenu dans le champ `TYPE`.
- des informations complémentaires contenues dans le champ `INFO`.
- une couleur représentative du groupe codée au format RGB codé sur un entier et contenu dans le champ `COLOR`.
- un drapeau indiquant si les résultats collectés sont filtrés. Ce drapeau est codé sous la forme d'un booléen contenu dans le champ `RESULTS_FILTERED`. Par défaut, les données collectées ne sont pas publiées.
- un drapeau indiquant si les informations de filtrage sont publiées. Ce drapeau est codé sous la forme d'un booléen contenu dans le champ `FILTER_PUBLISHED`. Par défaut, le filtre utilisé pour la publication des données collectées est publié.
- un drapeau indiquant si un utilisateur est autorisé à modifier les différents champs. Ce drapeau est codé sous la forme d'un booléen contenu dans le champ `WRITABLE`. Par défaut, tous les membres du groupe ont le droit de modifier les données relatives au groupe.
- un drapeau indiquant si le contrôle de la navigation de tout nouveau membre est aussitôt donnée au créateur. Ce drapeau est codé sous la forme d'un booléen contenu dans le champ `FREEZE`.
- un drapeau indiquant si l'espace de travail est automatiquement partagé dès qu'un nouveau membre rejoint le groupe. Ce drapeau est codé sous la forme d'un booléen contenu dans le champ `SHARED`.

A la création du groupe, par le biais du constructeur, les différents champs sont assignés avec une valeur par défaut. En l'occurrence, les résultats ne sont pas publiés et l'espace n'est pas partagé. Par contre, les informations relatives au filtrage sont publiées et tous les champs sont modifiables par

```
public class  CoordinatedNavigation  extends  CoVitesseGroup
{
 public void set(User user, Object field, Object value)
 {
  if (field != null)
  {
   if (   field.equals(CREATOR)
       || ((user != null) && user.equals(cache.get(CREATOR))))
   {
    super.set(user, field, value);
   }
  }
 }
 public Object get(User user, Object field)
 {
  if (field != null)
  {
   if (field.equals(WRITABLE))
  {
    if ((user != null) && (user.equals(cache.get(CREATOR)))) { return Boolean.TRUE; }
    else { return Boolean.FALSE; }
  }
   else if (field.equals(FREEZE)) { return Boolean.FALSE; }
   else if (field.equals(SHARED)) { return Boolean.FALSE; }
   else { return cache.get(field); }
  }
  return null;
 }
 public User [] join(User user, boolean req)
 {
  if (req) { return new User[] { (User) cache.get(CREATOR) }; }
  else if ((user != null) && (users.indexOf(user) == -1)) { users.addElement(user); }
  return null;
 }
 public void leave(User user)
 {
  if (user != null)
  {
    int index;

    if (user.equals(cache.get(CREATOR))) { users.removeAllElements(); }
    else if ((index = users.indexOf(user)) != -1) { users.removeElementAt(index); }
  }
 [...]
}
```

Figure 9
Implémentation du groupe CoVitesse de navigation coordonnée

tous les membres du groupe. Les fonctions définies par l'interface `Group` (`set`, `get`, `join`, `leave`) sont implémentées au niveau des classes programmant les différents types de navigation collaborative.

Nous illustrons la mise en œuvre de ces fonctions en détaillant la programmation de la navigation opportuniste par le biais de la classe `OpportunisticNavigation` dont un extrait du code est présenté dans la Figure 9. Dans cette navigation, le chef de session, c'est-à-dire le créateur du groupe, est le seul à disposer de privilèges pour modifier les préférences et pour accepter un nouvel utilisateur au sein du groupe.

La fonction `set` se contente de mettre à jour les champs avec les nouvelles valeurs passées en paramètre. Ainsi, la fonction vérifie que l'utilisateur qui est à l'origine de la modification est bien le chef de session. Dans le cas contraire, l'action n'est pas réalisée.

La fonction `get` retourne la valeur contenue dans le champ correspondant. S'il ne s'agit pas d'un champ particulier, le traitement est assuré par la fonction de la classe mère. Dans les autres cas, par exemple pour le cas du champ `WRITABLE`, la fonction vérifie qui est à l'origine de l'appel de fonction. Ainsi, la fonction renvoie vrai dans le cas du créateur et faux dans les autres cas. En effet, le chef de session est le seul à pouvoir modifier les valeurs des champs protégés.

La fonction `join`, dans le cas d'une requête pour rejoindre le groupe, retourne une liste d'utilisateurs à contacter et autorisés à accepter ou non un nouvel utilisateur. Ainsi, la liste est réduite à un seul utilisateur, à savoir le chef de session. Dans le cas où l'utilisateur est accepté, le nouvel utilisateur est ajouté à la liste des membres.

La fonction `leave` est appelée lorsqu'un utilisateur désire quitter le groupe. Si l'utilisateur est en l'occurrence le chef de session, alors tous les membres sont automatiquement exclus du groupe.

Concept de document web et d'album

Un document CoVitesse est une page *HTML*, c'est-à-dire une chaîne de caractères, et une référence d'un document CoVitesse est une adresse web (*URL*), c'est-à-dire une chaîne de caractères du type "http:// www.etc.com".

Un album CoVitesse est une collection de références et est géré par un ou plusieurs auteurs. Dans la version actuelle de CoVitesse, un auteur est identifié par une simple chaîne de caractères contenant le nom de l'auteur. Cependant, le mécanisme mis en œuvre au sein de la plate-forme Clover est plus complet puisqu'il peut gérer des objets implémentant l'interface `Auteur`.

A travers un album CoVitesse, un auteur peut réaliser les actions suivantes, comme le montre la portion de code de la Figure 10 :

```
public class CoVitesseAlbum  implements java.io.Serializable,   Album
{
 public static final String GET_REF = "GET_REF";
 public static final String ADD_REF = "ADD_REF";
 public static final String SET_FILTER = "SET_FILTER";
 public static final String REMOVE_REF = "REMOVE_REF";
 public static final String ADD_AUTHOR = "ADD_AUTHOR";
 public static final String CONTAINS_REF = "CONTAINS_REF";
 public static final String MAIL_CONTENT = "MAIL_CONTENT";
 private Filter filter = null;
 private Vector authors = new Vector();
 private Vector references = new Vector();

 private static final Object [] actions = { GET_REF, ADD_REF, SET_FILTER,
                                            SET_FILTER, REMOVE_REF, ADD_AUTHOR,
                                            CONTAINS_REF, MAIL_CONTENT };


 public Object set(Object who, Object field, Object value)
 {
  if (action != null)
  {
   if (action.equals(GET_REF))
   {
    Vector ret = (Vector) references.clone();
    if ((filter != null) && (authors.indexOf(who) == -1)) { ret = filter.filter(ret);
    return ret;
   } else [...]
  }

  return null;
 }
[...]
}
```

Figure 10
Implémentation d'un album contenant les résultats collectés par un utilisateur ou un groupe

- Ajouter (`ADD_REF`), supprimer (`REMOVE_REF`) et récupérer une référence (`GET_REF`). Dans le cas de cette dernière action, comme le montre la portion de code de la Figure 10, si l'utilisateur à l'origine de la requête n'est pas un auteur, alors le contenu retourné est automatiquement filtré (si le filtre existe).

- Vérifier la présence d'une référence (`CONTAINS_REF`).

- Ajouter (`ADD_AUTHOR`) et retirer un auteur (`REM_AUTHOR`).

- Ajouter ou modifier un filtre de publication (`SET_FILTER`). Pour mettre en œuvre les filtres, nous avons défini une classe `Filter` dotée d'une fonction `filter` qui prend en paramètre un vecteur de

références et qui retourne la liste filtrée. L'exemple de la Figure 10 montre l'utilisation de ce filtre.

- Envoyer par courrier électronique le contenu de l'album (`MAIL_CONTENT`).

Nous venons de présenter la mise en œuvre logicielle des principaux concepts au sein du système CoVitesse. La section suivante présente en détail le fonctionnement de la machine à états du système relatif à l'aspect coordination.

Machine à état du Noyau Fonctionnel

La machine à état du système CoVitesse relative à l'aspect coordination est cohérente avec la machine à état du serveur de coordination de la plate-forme Clover (cf. Chapitre VI) puisque nous retrouvons les mêmes états reliés par les mêmes transitions. Cependant, cette machine à états

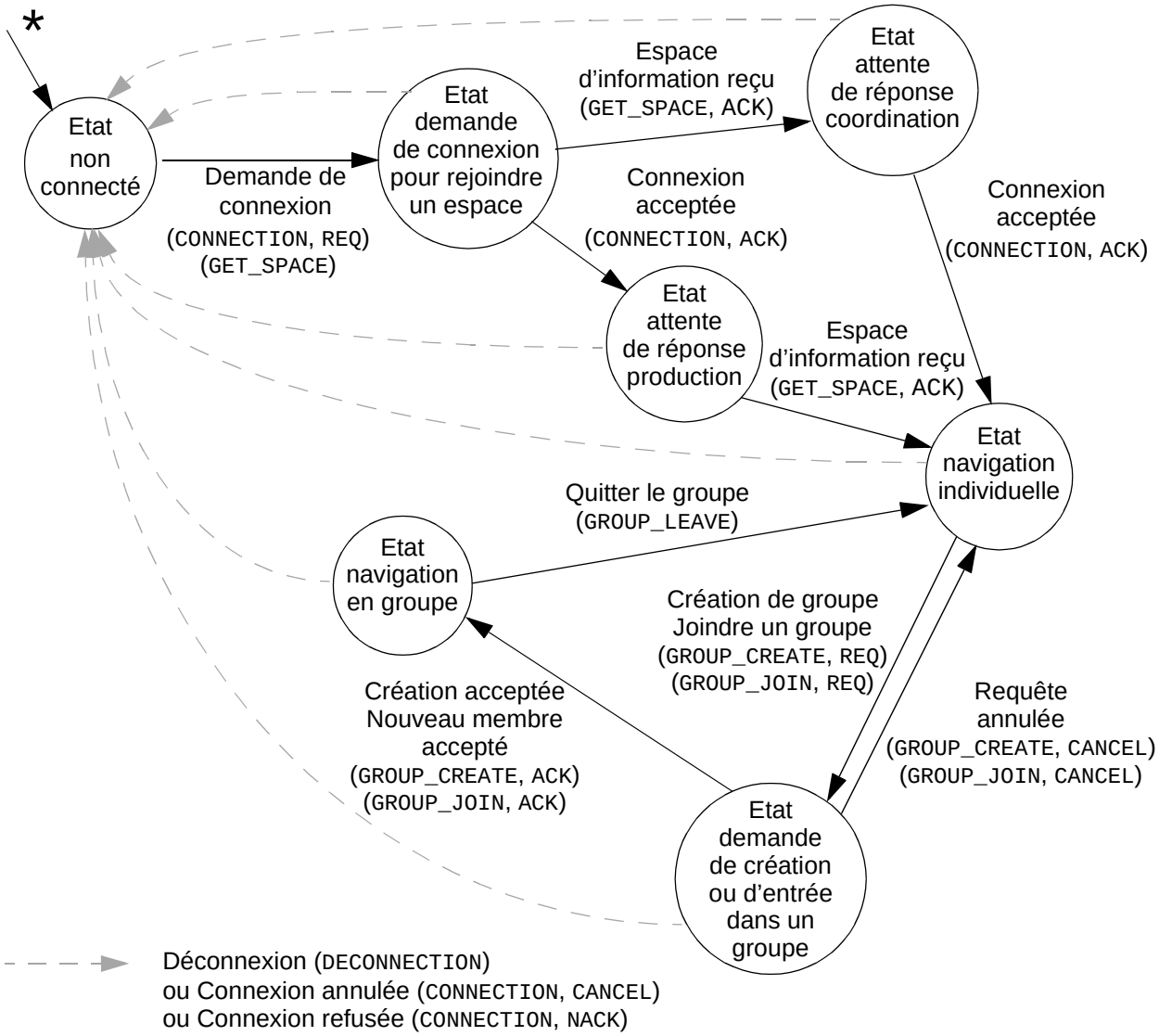


Figure 11
Diagramme d'états du système CoVitesse

comporte deux états supplémentaires comme le montre la Figure 11. En l'occurence, il s'agit de deux états transitoires attendant la confirmation d'une entrée dans l'espace de collaboration et l'espace de production pour passer dans l'état de *navigation individuelle*. Le passage à un de ces états transitoires fait suite à l'envoi de requête de connexion pour rejoindre l'espace de coordination (`CONNECTION`, `REQ`) et pour rejoindre l'espace de production (`GET_SPACE`, `REQ`). Ainsi, pour arriver dans l'état *navigation individuelle*, en passant par l'un des deux états transitoires supplémentaires, il est nécessaire de recevoir une double confirmation autorisant l'accès à l'espace de coordination et à l'espace de production ((`CONNECTION`, `ACK`) et (`GET_SPACE`, `ACK`)). Dès que l'on a reçu confirmation, une demande d'abonnement à l'espace de communication "forum de discussion" est ensuite envoyée au serveur de communication (`SUBSCRIBE`, `CHANNEL`). Dans l'état actuel du système, la demande est acceptée automatiquement, ce qui justifie l'absence d'état transitoire dans le diagramme d'états. Dans le cas d'un refus, le système retourne dans l'état *non connecté*.

Dans cette partie, nous avons présenté la mise en logicielle du système CoVitesse. La partie suivante illustre un autre exemple plus simple : le développement d'un tableau blanc partagé à l'aide de la plate-forme Clover.

3. Tableau blanc partagé

Dans cette partie, nous présentons un second exemple de mise en œuvre d'un collecticiel à l'aide de notre plate-forme Clover. Il s'agit d'un système de tableau blanc partagé. Le premier paragraphe présente une description de l'utilisation du système. Le second paragraphe présente les aspects techniques de mise en œuvre logicielle du système.

3.1. DESCRIPTION DU SYSTÈME

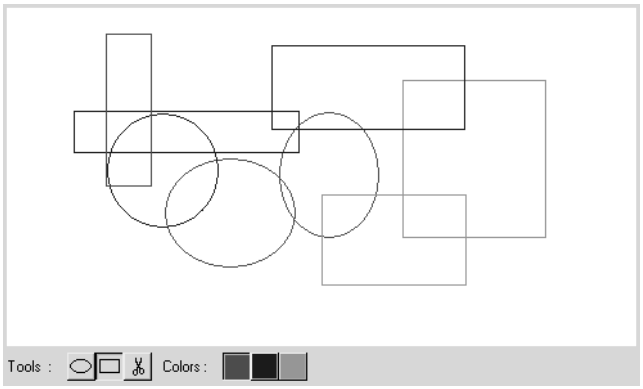


Figure 12
Fenêtre du tableau blanc partagé

Le système de tableau blanc partagé, comme son nom l'indique, met à disposition une surface de dessin accessible par plusieurs utilisateurs. Ce système de tableau blanc partagé est très simple dans sa conception puisqu'il ne met à disposition, comme le montre la Figure 12, que quelques outils : deux formes géométriques (ellipse et rectangle), trois couleurs (rouge, bleu et vert) et les ciseaux pour supprimer une forme géométrique. Pour dessiner une figure, il suffit de cliquer dans deux endroits de la zone de dessin. Au cours du dessin, la forme géométrique se déforme en suivant le curseur de la souris. Dès que le second point est sélectionné, la figure est dessinée et l'apparition d'une nouvelle figure est aussitôt transmise aux autres clients. Il en est de même pour la disparition d'une figure lorsque celle-ci est supprimée à l'aide de l'outil ciseaux.

Dans ce système, tout nouvel utilisateur accède automatiquement à l'espace de coordination et de production. Contrairement au système précédent, un utilisateur n'a pas besoin de donner un nom et un mot de passe. Ceux-ci sont générés automatiquement. L'aspect production se reflète à travers le concept de tableau et le concept de figures. Le tableau est vu comme un album partagé par tous les utilisateurs. Cependant, il n'y a pas de notion de propriété puisque tous les utilisateurs ont tous les mêmes droits, y compris d'effacer ce qui a été dessiné par d'autres.

De plus, ce système mémorise l'état du tableau et autorise ainsi le travail asynchrone. Le contenu est stocké au niveau du serveur dédié à la production.

3.2. MISE EN ŒUVRE LOGICIELLE

Dans ce paragraphe, nous présentons les principaux points de mise en œuvre logicielle du système de tableau blanc partagé. La première section présente l'architecture logicielle du système et la seconde section présente la programmation des concepts de tableau et de forme géométrique.

Architecture logicielle

L'architecture logicielle, contrairement au système CoVitesse, ne dispose pas de composant dédié à la communication. Le système est principalement axé sur la production. La partie consacrée à la coordination est très limitée puisqu'il s'agit juste de mettre en relation les différents utilisateurs. Ainsi, comme le montre la Figure 13, le composant du Trèfle Fonctionnel public et le Trèfle Fonctionnel privé (client) sont constitués de deux composants, un est dédié à la production, l'autre à la coordination. Le Trèfle Fonctionnel privé localisé sur le serveur est une

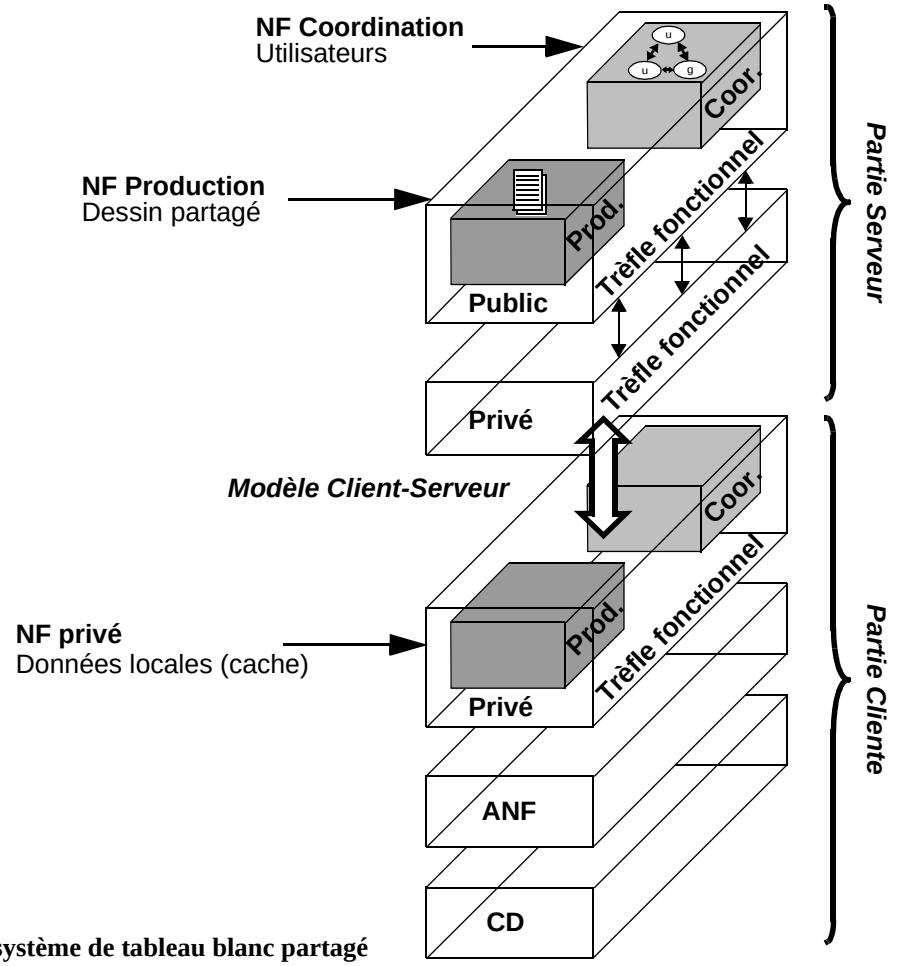


Figure 13
Architecture du système de tableau blanc partagé

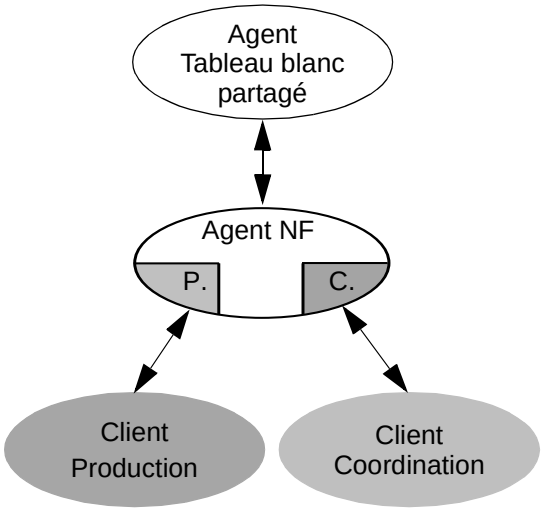


Figure 14
Architecture de l'application cliente du tableau blanc partagé

simple coquille vide qui assure le lien entre le Trèfle Fonctionnel public et la partie cliente du Trèfle Fonctionnel privé.

Le reste de l'architecture est constitué de la partie dialogue du tableau blanc partagé résultant, comme le montre la Figure 14, d'une fusion de l'Adaptateur du Noyau Fonctionnel avec le Contrôleur de Dialogue sous la forme d'un unique agent *Tableau blanc partagé.*

L'agent *Noyau Fonctionnel*, comme le montre la Figure 14, implémente la partie cliente du Trèfle Fonctionnel privé. Puisque le système ne

```
public class ShapeReference implements Reference
{
 public static final int ELLIPSE = 0;
 public static final int RECTANGLE = 1;

 private int type = RECTANGLE;
 private Color color = Color.red;
 private Rectangle bounds = new Rectangle();

 public ShapeReference(int type, int x, int y, int w, int h, Color color)
 {
  if ((type == ELLIPSE) || (type == RECTANGLE))
  {
   this.type = type;
   bounds.x = x;
   bounds.y = y;
   bounds.width = w;
   bounds.height = h;

   if (color != null) { this.color = color; }
  }
 }
[...]
}
```

Figure 15
Implémentation d'une figure géométrique

propose pas de moyen de communication, les deux seuls fils de l'agent *Noyau Fonctionnel* sont les agents clients pour la production et la coordination. La machine à état est plus simple que le système précédent puisqu'elle n'est constituée que de deux états : le mode non connecté et le mode connecté.

Concepts de tableau et de forme géométrique

Les principaux concepts manipulés par ce système sont les concepts de tableau, implémenté sous forme d'album, et de forme géométrique, implémenté sous forme de référence. Le concept d'utilisateur est très simple dans cet exemple puisqu'il se limite à un nom généré automatiquement par le système. Du point de vue de l'utilisateur, le concept de groupe est inexistant.

Le concept de forme géométrique, comme le montre la portion de code dans la Figure 15, est implémenté par la classe `ShapeReference`

```
public class WhiteBoard implements java.io.Serializable, Album
{
  public static final String GET_SHAPES = "GET_SHAPES";
  public static final String ADD_SHAPE = "ADD_SHAPE";
  public static final String CLEAR = "CLEAR";
  public static final String REMOVE_SHAPE = "REMOVE_SHAPE";
  public static final String CONTAINS_SHAPE = "CONTAINS_SHAPE";

  private Vector references = new Vector();

  private static final Object [] fields = { GET_SHAPE, ADD_SHAPE, CLEAR,
                                            REMOVE_SHAPE, CONTAINS_SHAPE };

  public Object set(Object who, Object field, Object args)
  {
   if (field != null)
   {
     if (field.equals(ADD_SHAPE && (value != null)) { references.addElement(value);
     else if (field.equals(CLEAR))
    {
      references.removeAllElements();
    }
    else if (field.equals(REMOVE_SHAPE) && (value != null))
    {
      int index = references.indexOf(value);
      if (index != -1) { references.removeElementAt(index); }
    }
    else if (field.equals(GET_SHAPES))
    {
     return references.clone();
    } else [...]
   }
  }

  [...]
}
```

Figure 16
Implémentation du tableau blanc partagé sous forme d'album

héritant de l'interface `Reference`. Théoriquement, le concept de forme de géométrique aurait dû être implémenté sous forme de document, mais nous avons simplifié le développement en utilisant uniquement une référence.

Une forme géométrique est identifiée par son type, un rectangle ou un cercle, une couleur et une taille (coordonnées de la boîte englobante, longueur et largeur). Ces formes géométriques sont manipulées par le tableau blanc partagé.

Le concept de tableau est implémenté sous forme d'album par la classe `WhiteBoard` dont un extrait du code est donné dans la Figure 16. Les différentes actions réalisables sont :

- Ajouter (`ADD_SHAPE`) et supprimer (`REM_SHAPE`) une forme géométrique,
- Nettoyer le contenu (`CLEAR`),
- Récupérer le contenu (`GET_SHAPES`),
- Vérifier l'existence d'une forme géométrique (`CONTAINS_SHAPE`).

Ces actions sont réalisables par le biais de la fonction `set` qui prend en argument l'auteur de l'action (inutilisé dans cet exemple), le type d'action et les arguments.

4. Robustesse de la plate-forme Clover

Nous avons montré à travers les systèmes CoVitesse et tableau blanc partagé comment programmer un collecticiel à l'aide de la plate-forme Clover. Aussi, à l'aide de ces deux systèmes, nous avons mené une série de tests afin d'évaluer la robustesse de la plate-forme en concevant des robots. Le premier paragraphe présente une série de tests réalisée avec l'aide du système CoVitesse. Le second paragraphe présente une série de tests réalisée avec le système de tableau blanc partagé.

4.1. COVITESSE

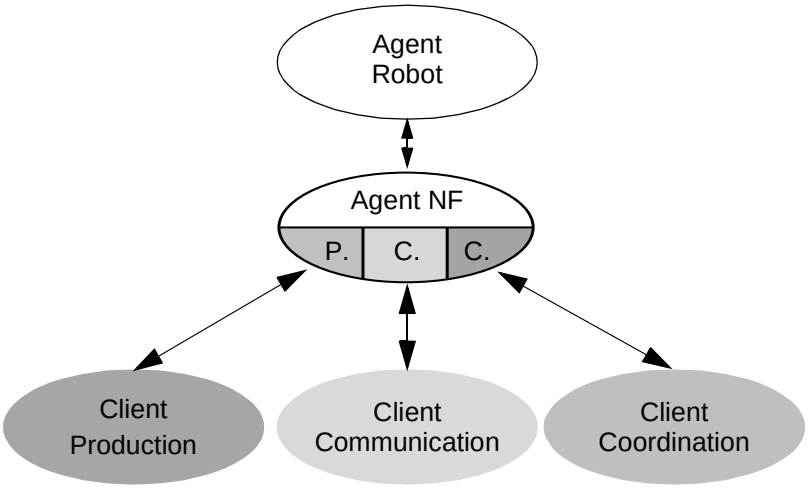


Figure 17
Architecture de CoVitesse avec un agent robot

La première série de tests menée avec CoVitesse est essentiellement axée sur la résistance de la partie coordination, de la partie production et, dans une moindre mesure, de la partie communication. Plus précisemment, nous avons testé la résistance de la machine à état de la coordination et la robustesse de la gestion de la concurrence (coordination et production). Les tests menés sur la partie communication se contentaient de vérifier que le serveur de communication résistait à une production massive de messages. Néanmoins, nous ne disposons pas de résultats numériques : la capacité de résistance de la plate-forme a été mesurée à l'utilisation de l'application du système CoVitesse avec les robots (effets sur l'interaction, surcharge du processeur côté serveur et côté client).

Ainsi pour mettre en place ces tests, nous avons développé un agent robot "émulant" le comportement d'un utilisateur. Comme le montre la Figure 17, cet agent robot réutilise entièrement le Noyau Fonctionnel de CoVitesse sans y apporter une quelconque modification. Cette facilité de mise en œuvre valide la propriété de modifiabilité et de réutilisabilité véhiculée par notre approche par agent (voir chapitre précédent). En effet, la mise en œuvre de ce robot résulte d'un assemblage d'agents développés pour le système CoVitesse et réutilisés pour construire le robot.

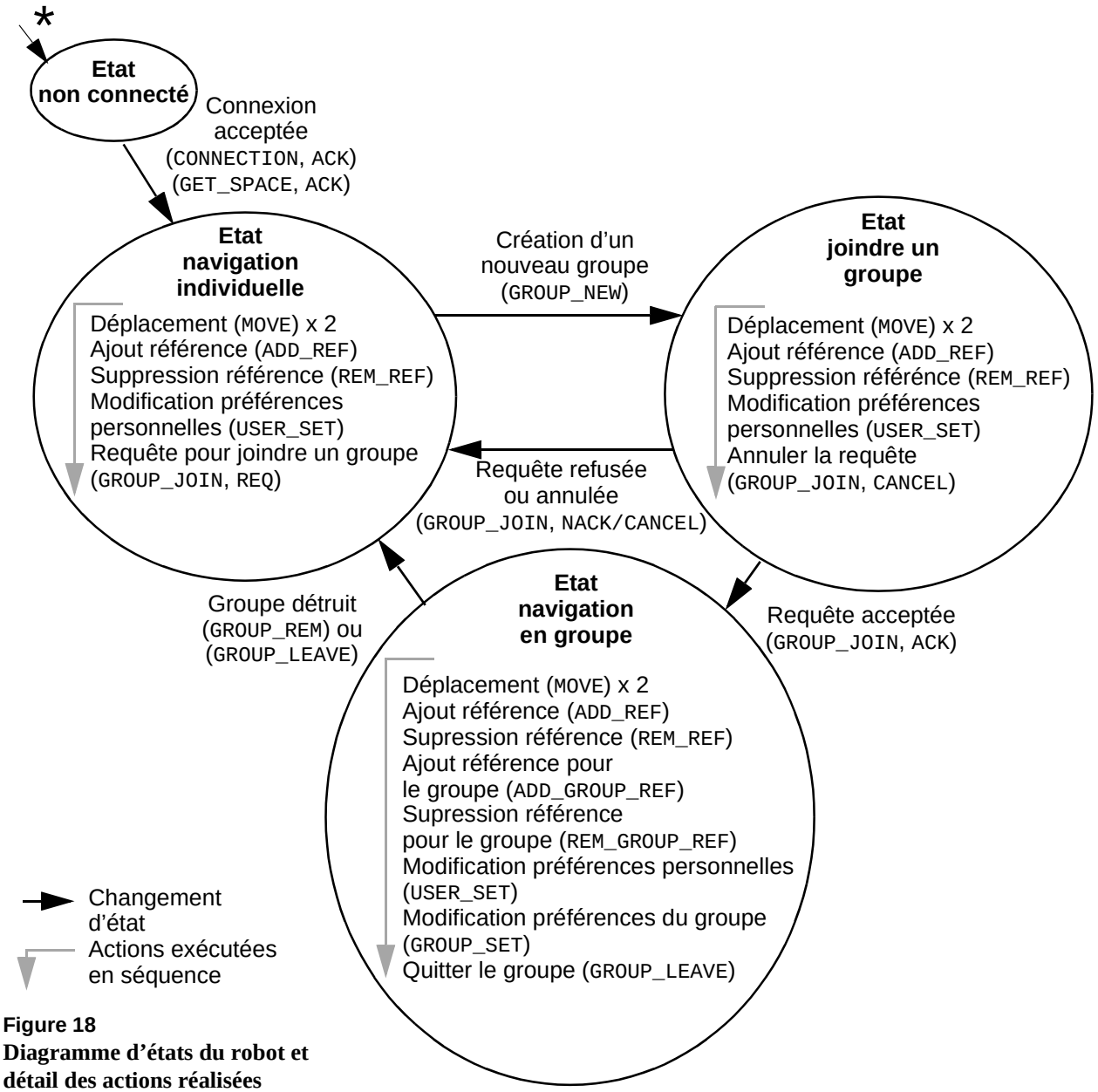


Figure 18
Diagramme d'états du robot et détail des actions réalisées

Ces robots sont réglés par une machine à état dont le diagramme est donné par la Figure 18. Cette machine est constitué de trois états (*navigation individuelle*, *joindre un groupe* et *navigation en groupe*) et d'un état correspondant à l'initialisation (*Etat non connecté*). Pour chaque état, les robots exécutent une séquence d'action pour tester les différentes fonctionnalités. Le temps de latence entre l'exécution de deux actions est de 200ms plus ou moins un temps aléatoire compris entre 0 et 100 ms. Ainsi, les différents robots exécutent les mêmes actions dans un même ordre de grandeur de temps de latence mais avec une composante aléatoire pour générer le plus grand nombre d'actions concurentes. Le passage à l'état *Joindre un groupe* est possible dès que les robots reçoivent l'information qu'un groupe a été créé (GROUP_NEW). Ensuite, la

machine à état boucle en passant d'un état à l'autre en exécutant l'une de ces actions :

- Joindre un groupe (état *navigation individuelle*),
- Annuler la requête (état *joindre un groupe*),
- Quitter le groupe (état *navigation en groupe*).

Cependant, le passage d'un état à un autre peut être imposé lorsqu'un groupe est supprimé (passage immédiat à l'état *navigation individuelle*) dans le cas d'une navigation coordonnée ou lorsqu'une requête pour joindre un groupe est refusée.

Le résultat des tests est positif puisque nous avons observé une faible surcharge des processeurs, à la fois côté client et serveurs, et un temps de réponse, du point de vue de l'interaction, tout à fait honorable. En effet, la présence de dizaines d'"utilisateurs" perturbe très peu l'interaction qui reste dans l'ensemble très fluide (peu d'effets du type "écran gelé" à cause d'une surcharge). Cependant, nous avons pu noter que la présence d'une grande quantité d'utilisateurs réduit la qualité de la conscience de groupe (*awareness*) à cause des déplacements intempestifs et trop rapides. Enfin, il apparaît que le contenu de l'album du groupe est cohérent puisqu'il contient, après vérification, les bonnes valeurs.

4.2. TABLEAU BLANC PARTAGÉ

La seconde série de tests menée avec le système de tableau blanc partagé a débouché sur un ensemble de mesures. De la même manière, nous avons développé un agent robot avec un comportement prédéfini. Ces tests ont permis de mesurer la robustesse de la partie production pour la gestion du tableau et de la partie coordination pour sa capacité de diffusion des informations. Ces tests ont été effectués sur un réseau local. Il conviendrait de réaliser une série de tests sur de longues distances.

Cette série de tests a été menée en deux étapes. La première a consisté à mesurer le temps mis par un utilisateur expert pour dessiner le plus rapidemment possible un ensemble de figures et à mesurer le temps de propagation des informations. L'enchainement des dessins a été enregistré de sorte que les robots reproduisent le même comportement que l'utilisateur mais avec des temps de latence variables. L'intervalle de temps entre le dessin de deux figures est de 360 ms pour un utilisateur expert. Cette valeur est confirmée par l'utilisation de la loi de Fitts [Fitts 1954], en fonction des valeurs données par le modèle du processeur humain [Card 1983], en calculant la valeur moyenne des temps de latence connaissant la liste des figures dessinées. Puis, dans la seconde étape, nous avons réalisé trois tests avec le robot avec des temps de latence de 360 ms, 200 ms et 100 ms.

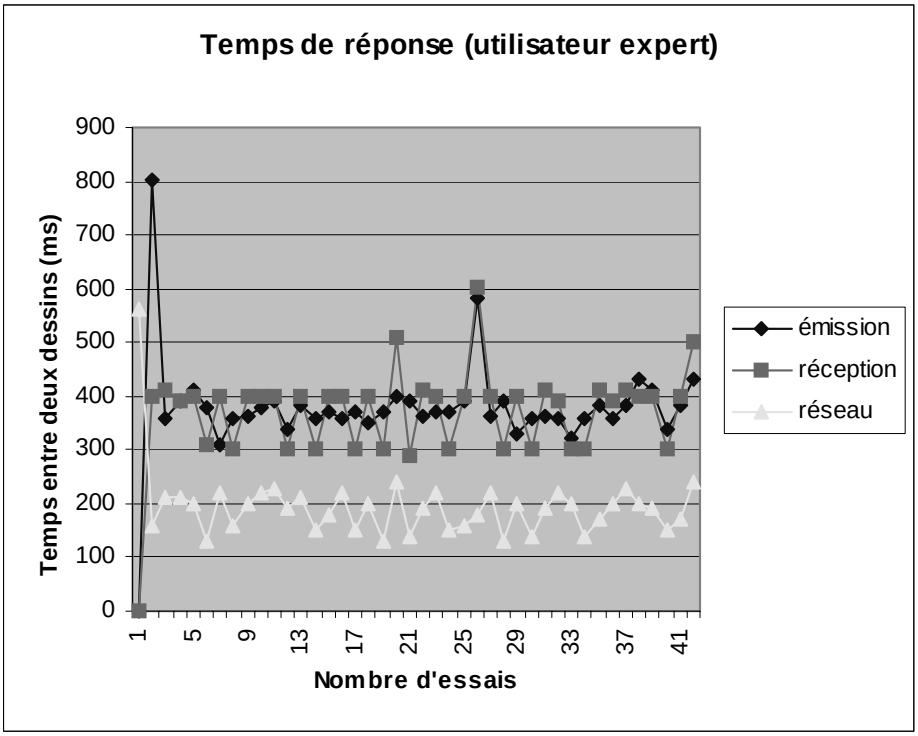


Figure 19
Temps réalisés par un utilisateur expert

La première série de tests a été réalisée par un utilisateur expert. Comme le montre la Figure 19, la réception des informations (courbe en violet) suit le même mouvement que l'émission des données (courbe en bleu). Cependant, la réception est nettement plus saccadée ce qui est potentiellement lié au retard induit par la communication réseau. Comme le montre la courbe jaune, ce temps de latence est borné et oscille autour d'une valeur moyenne de 200ms.

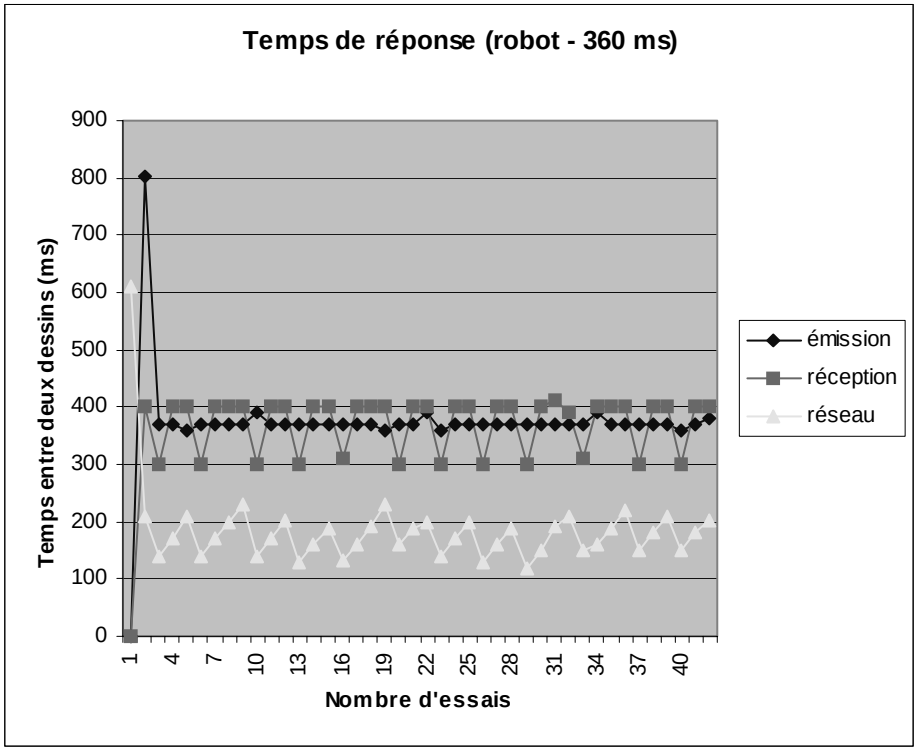


Figure 20
Temps réalisés par un robot configuré avec un intervalle de 360 ms

Avec la seconde série de tests, nous avons reproduit des résultats identiques avec un robot programmé selon un temps de latence de 360ms. Cette série de tests, comme le montre la Figure 20, confirme le bon temps de réponse de la plate-forme et montre que la diffusion des informations est du même ordre de grandeur que l'émission des données, avec un retard induit par le réseau oscillant toujours autour de 200ms.

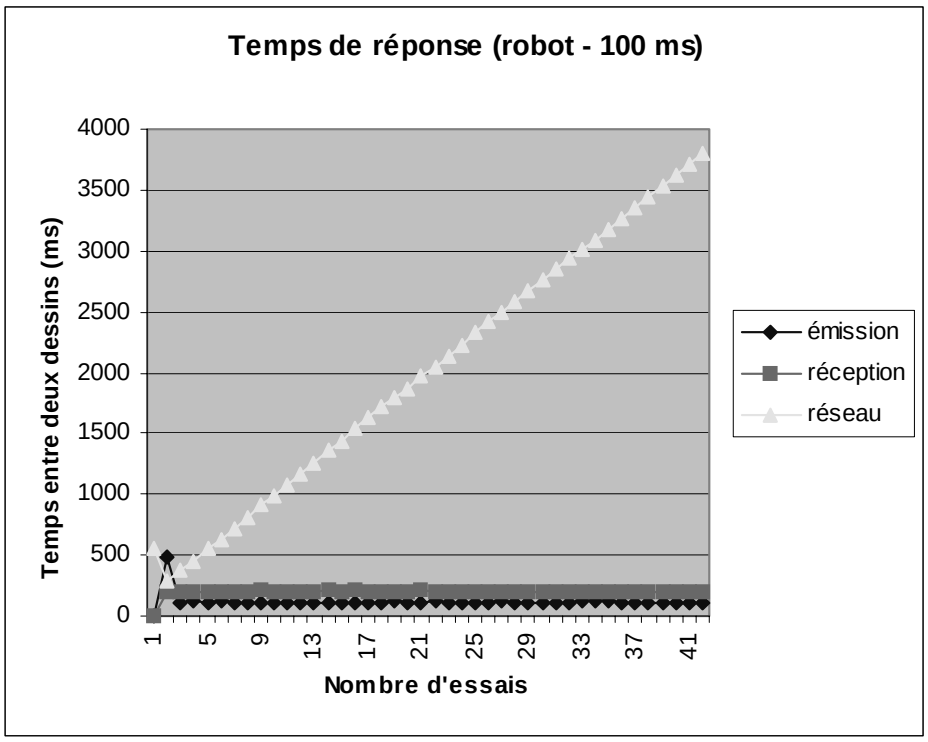


Figure 21
Temps réalisés par un robot configuré avec un intervalle de 100ms

A partir d'un temps de latence inférieur à 200ms, le retard induit par le réseau ne cesse d'augmenter comme le montre la Figure 21. En effet, cette troisième série de tests a été réalisée avec un robot programmé selon un temps de latence de 100ms. Le retard induit par le réseau progresse linéairement selon un pas de 200ms.

Nous avons pu observer que la plate-forme donne de bon résultats en terme d'interaction mais montre des limitations du fait des communications à travers le réseau et de sa forme hybride. Cependant, nous avons pu constater que la plate-forme résiste très bien à un envoi massif de messages et finit par les diffuser à l'ensemble des applications clientes.

5. Résumé des contributions

Dans ce chapitre nous avons montré que :

- **La plate-forme Clover est générique, fonctionnelle et utilisable** pour développer des collecticiels à travers le développement du système **CoVitesse** et d'un **tableau blanc partagé**. Ceci nous a permis de vérifier la qualité de la couverture fonctionnelle et la généricité de cette plate-forme. De plus, cette qualité de généricité a été vérifiée par l'utilisation simultanée du système CoVitesse et du système de tableau blanc partagé.

- **La plate-forme Clover est extensible, modifiable et favorise la réutilisation** : ceci a été montré à travers une utilisation simultanée de deux types de documents (tableau blanc et album de pages web), une utilisation de plusieurs types de groupes avec modification dynamique des règles régissant ces groupes (navigation personnalisée), une réutilisation de composants pour élaborer les robots afin de mener des tests de robustesse.

- **La plate-forme Clover est robuste** : nous avons pu vérifier la robustesse de la plate-forme en menant des séries de tests concluants.

Cependant, la plate-forme a montré ses limites en terme de temps de réponse. De plus, la partie communication est incomplète et des développements sont en cours pour que la plate-forme intègre entièrement le modèle de couverture fonctionnelle dédié à la communication. Dès que ces développements seront achevés, il est prévu de tester l'intégration de nouveaux moyens de communication tels que la vidéo.

Chapitre VIII *Conclusion*

Motivés par le constat que le développement d'un collecticiel constitue une tâche complexe, les travaux menés dans cette thèse sont dédiés aux outils de conception logicielle et de développement des collecticiels.

Nous proposons ici, en conclusion, un résumé de nos contributions en soulignant leur originalité. Une prise de recul par une analyse critique des résultats permet d'envisager pour nos travaux de multiples perspectives que nous organisons en deux parties : les extensions et les prolongements à plus long terme.

1. Résumé de la contribution

Nos travaux de recherche contribuent au domaine de l'ingénierie logicielle des collecticels selon deux formes complémentaires : un modèle d'architecture conceptuelle et une plate-forme de développement.

Démarche de travail centrée sur l'activité de groupe

La démarche adoptée pour mener ces travaux s'appuie sur la notion d'activité de groupe en termes d'actions, individuelles ou collectives, et de contexte des actions. Nous raisonnons sur l'observabilité des actions des membres d'un groupe ainsi que du contexte des actions supposé partagé par les utilisateurs, pour mener notre étude des outils de réalisation logicielle. L'originalité de cette démarche repose donc sur la prise en compte de propriétés ergonomiques et d'un modèle de l'activité de groupe incluant les trois espaces fonctionnels du trèfle (production, communication et coordination) pour jeter un nouveau regard sur les outils de réalisation logicielle. En effet, l'objectif affiché de nos travaux

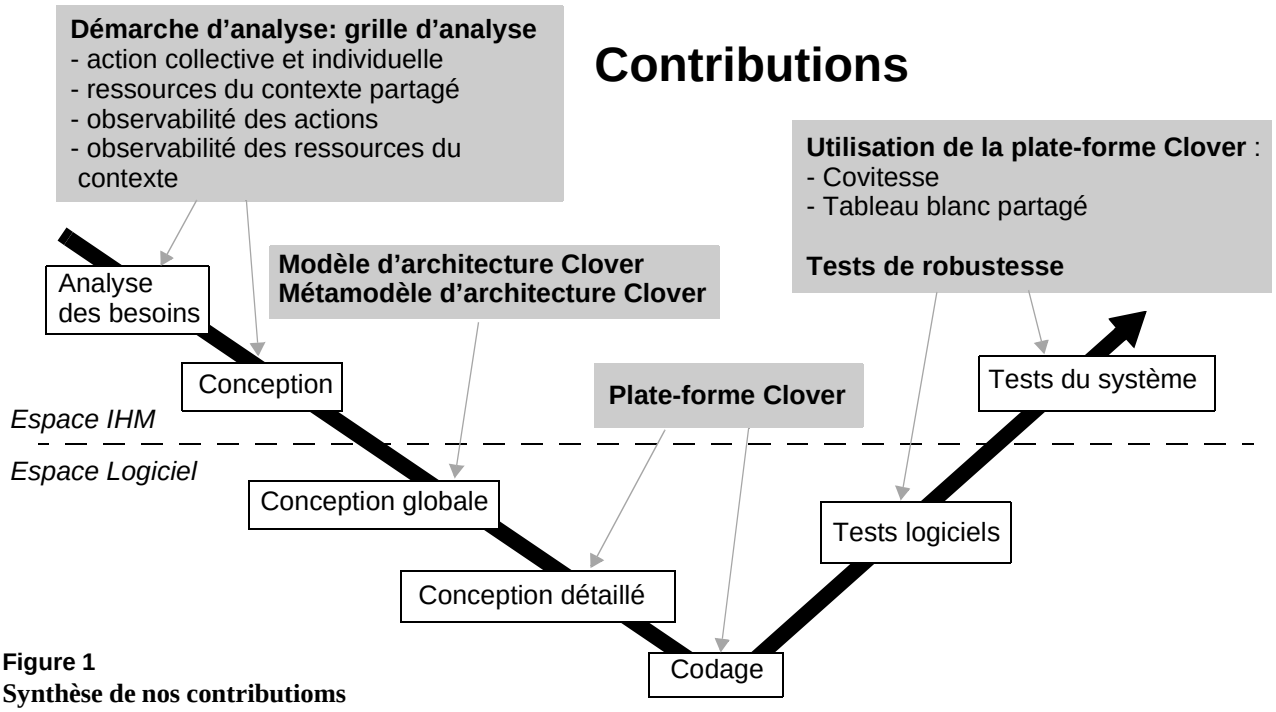


Figure 1
Synthèse de nos contributioms

(Chapitre II) est de faciliter le passage entre la conception de l'interaction guidée par des facteurs ergonomiques, sociaux et éthiques et la réalisation logicielle dirigée par des critères de qualité logicielle. Ainsi, la réalisation logicielle ne se résume pas seulement à exploiter au mieux les ressources techniques, mais à développer des techniques d'interaction qui soient conformes aux attributs de l'activité de groupe.

Concrètement, comme le montre la Figure 1, cette démarche de travail s'est traduite par l'élaboration d'une grille d'analyse (Chapitre II, Figure 13) qui organise les paramètres de l'activité de groupe en quatre dimensions : les actions, l'observabilité des actions, les ressources du contexte supposé partagé et l'observabilité de ces ressources. Cette grille est ensuite utilisée pour étudier les architectures conceptuelles et les outils de développement pour les collecticiels mais aussi pour motiver nos contributions.

Modèle d'architecture Clover et approche par composants

Fidèle à notre démarche de travail, nous avons étudié les modèles d'architecture conceptuelle pour collecticiels selon notre grille d'analyse (Chapitre III). Les lacunes des modèles existants vis-à-vis de notre grille d'analyse nous ont alors incités à proposer un nouveau modèle : le modèle Clover (Chapitre IV).

D'une part, l'action collective déclinée selon le modèle du trèfle (production, communication et coordination) est explicite dans le modèle d'architecture Clover. En effet, ce dernier préconise une décomposition du collecticiel en composants (au sens de la programmation par composants) contenant trois sous-composants dédiés à la production, à la

communication et à la coordination. Ainsi l'unité structurante du modèle Clover est un composant logiciel collaboratif, doté d'une interface et encapsulant des fonctions de production, de communication et de coordination. Le modèle Clover rend donc explicites les éléments de notre grille d'analyse qui décrivent les activités collectives : pour ces éléments de la grille, nous caractérisons alors la compatibilité entre la grille et le modèle Clover comme forte.

D'autre part, les ressources du contexte individuel et partagé ainsi que leur observabilité (deux dimensions de notre grille) sont prises en compte dans le modèle Clover par les deux notions orthogonales, de composants privés/publics et de politique de filtrage. Nous constatons là encore une compatibilité forte entre la grille et le modèle Clover.

La validation d'un modèle d'architecture logicielle conceptuelle est certes une entreprise difficile. Une architecture n'est jamais intrinsèquement bonne ou mauvaise, mais sa qualité se juge à la lumière de propriétés identifiées par avance. Nous avons avancé un ensemble de propriétés véhiculées par le modèle Clover, comme la modularité selon des éléments de conception liés à l'activité de groupe (les liens entre le modèle et la grille d'analyse). Une façon complémentaire de valider un modèle est de démontrer qu'il correspond à une pratique logicielle. En montrant que le modèle est en accord avec la structuration de trois collecticiels existants (Chapitre IV), nous visons à démontrer que le modèle est implicitement appliqué par les développeurs de collecticiels. Enfin, une dernière approche de validation est de concevoir une plate-forme logicielle de développement qui génère des collecticiels structurés selon le modèle Clover.

Plate-forme Clover et couverture fonctionnelle générique

Appliquant la même démarche de travail que pour les modèles d'architecture, nous avons étudié les outils de développement pour les collecticiels (Chapitre V) de façon systématique, à la lumière de notre grille d'analyse. Les manques identifiés de ces outils par rapport à notre grille d'analyse ont motivé l'élaboration d'une nouvelle plate-forme de développement pour les collecticiels : la plate-forme Clover (Chapitre VI).

Pour concevoir la plate-forme, nous nous sommes appuyés sur la grille d'analyse, centrale à notre démarche de recherche. Au lieu de raisonner en termes de composants logiciels comme nous l'avons fait pour notre étude des modèles d'architecture logicielle, nous avons traduit les éléments de notre grille en termes de fonctions génériques (visant une couverture fonctionnelle), de structures de données et de mécanismes logiciels. Ainsi, nous avons défini un modèle de couverture fonctionnelle de l'activité de groupe à partir duquel nous avons déduit trois couvertures fonctionnelles dédiées à la production, à la communication et à la

coordination. Ces couvertures fonctionnelles ou ensembles de fonctions génériques rendent explicites les éléments de notre grille d'analyse qui décrivent les actions collectives (production, communication et coordination) et individuelles, au sein même de la plate-forme et donc du code des collecticiels produits. De plus la description des ressources collectives et individuelles, une dimension de notre grille d'analyse, se traduit dans notre plate-forme par les structures de données suivantes : l'espace de l'activité de groupe, les participants à l'activité de groupe et le support à l'activité de groupe. Enfin l'observabilité des actions et des ressources, deux dimensions de la grille, ont donné lieu à des mécanismes logiciels comme des services de notification et de filtrage.

Ces fonctions, structures et mécanismes génériques ont été traduits sous la forme de classes et interfaces Java qui constituent notre plate-forme Clover. Pour illustrer l'utilisation de cette plate-forme et motiver l'apport qu'elle offre aux développements des collecticiels, nous avons développé deux collecticiels (Chapitre VII) : le système CoVitesse, système de navigation collaborative sur le *WWW*, et un système de tableau blanc partagé.

Utilisation du modèle d'architecture Clover et de la plate-forme Clover

Les systèmes CoVitesse et le système de tableau blanc partagé ont été construits à partir de notre modèle d'architecture Clover et développés à l'aide de la plate-forme Clover. Nous avons ainsi testé et vérifié l'utilité et l'utilisation de ces deux outils.

Nous avons notamment pu vérifier la propriété de modularité et l'approche par composant véhiculées par le modèle d'architecture Clover : la forte modularité nous a permis en particulier de programmer et tester séparément des éléments logiciels, qui ont été ensuite assemblés au sein des deux collecticiels.

De plus nous avons pu informellement mesurer les apports de la plate-forme en terme de coût de développement, pour deux collecticiels développés mais aussi pour leurs tests. En effet, le développement de robots pour tester la robustesse s'est appuyé sur la réutilisation de composants logiciels. Les tests des deux collecticiels montrent que la plate-forme Clover est robuste. Néanmoins cette dernière n'est pas complète. Les extensions de cette plate-forme font l'objet du paragraphe suivant.

2. *Limitations et extensions*

Le développement d'une plate-forme de réalisation de collecticiels est certes une tâche ambitieuse dans le temps imparti d'une thèse. Aussi la plate-forme Clover est incomplète. En effet, notre approche a consisté à développer un noyau minimal dont la robustesse a été testée afin de définir une base solide. Nos perspectives à court terme visent donc la complétude de la plate-forme Clover, qui permettra ensuite de la comparer aux outils de construction existants.

Compléter la plate-forme Clover

D'une part, la plate-forme Clover n'implémente que partiellement la couverture fonctionnelle relative à la communication. Seul le concept d'espace de communication a été développé entièrement. Les concepts d'abonné et de liste de diffusion ont été implémentés de façon minimaliste afin que le système puisse fonctionner. Par conséquent, il convient de compléter les développements afin d'intégrer entièrement les concepts d'abonné et de liste de diffusion. De plus, la version actuelle de la plate-forme met en œuvre uniquement des moyens de communication reposant sur un mode de communication textuelle. Des développements sont en cours afin d'étendre les moyens de communication en intégrant l'audio et la vidéo.

D'autre part, la plus grande attention a été portée sur la couverture fonctionnelle, pour identifier et proposer des fonctionnalités à haut niveau d'abstraction. La couverture fonctionnelle définie, il semble maintenant nécessaire d'améliorer ces fonctionnalités par l'intégration de services systèmes plus élaborés que ceux existants ; notamment, en vue de rendre possible le développement de collecticiels selon un schéma d'exécution totalement réparti, il convient d'améliorer le service de gestion de la concurrence et de la cohérence des données.

Affiner la couverture fonctionnelle

La plate-forme Clover, telle qu'elle a été développée, met en œuvre des moyens pour définir une politique de filtrage et de publication des données, afin de garantir la protection de l'espace privé tout en préservant la conscience de groupe et la rétroaction de groupe. Cependant, nous avons observé que, d'une part, la notion de filtrage était implicite dans le modèle de couverture fonctionnelle, et que, d'autre part, une trop grande part de la mise en œuvre d'une politique de filtrage est laissée à la charge du développeur. Ainsi, dans un premier temps, il convient d'affiner le modèle générique de couverture fonctionnelle pour y intégrer explicitement la notion de filtrage. Dans un second temps, la plate-forme Clover doit être étendue pour prendre en compte cette évolution de la couverture fonctionnelle et ainsi fournir au développeur des mécanismes logiciels pour définir une politique de filtrage à coût réduit.

Comparer la plate-forme avec les outils existants

Afin de mesurer les apports de notre plate-forme Clover, il convient enfin de considérer différents scénarios de mise en œuvre d'un collecticiel, pour comparer la plate-forme Clover à d'autres outils existants. L'éventail de scénarios nécessaires à la comparaison doit couvrir le développement mais aussi les évolutions du collecticiel. On relèvera ici l'analogie avec les scénarios centrés sur l'utilisateur et conçus pour tester les modèles de tâches ou les spécifications externes d'un système interactif. Des mesures de comparaison peuvent inclure : le temps de développement, le temps de modification, le nombre de lignes de code ou l'efficacité à l'exécution du collecticiel produit.

3. Prolongements

Tandis que nos travaux à court terme se concentrent sur la plate-forme Clover, nos prolongements à plus long terme s'organisent selon deux axes : le premier axe concerne l'architecture logicielle et l'approche par composants tandis que le deuxième axe vise à étendre notre démarche de recherche en considérant des scénarios d'utilisabilité.

3.1. ARCHITECTURE LOGICIELLE

Nous identifions deux prolongements à notre étude des architectures logicielles. Un premier sujet d'étude concerne la remise en cause de la métaphore de la "fermeture éclair" dans l'organisation des composants logiciels publics et privés d'un collecticiel. Le deuxième sujet d'étude vise à exploiter la structuration des unités logicielles de notre modèle Clover dans une approche de programmation par composants.

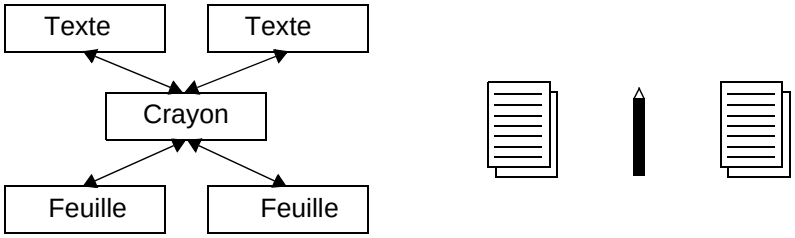


Figure 2
Architecture en X

Architecture en X

Les modèles d'architecture conceptuelle étudiés ainsi que notre modèle Clover préconisent des composants logiciels publics et privés correspondant à des niveaux d'abstraction allant du niveau le plus haut de nature sémantique (Niveau S), au niveau le plus bas dépendant du matériel. Appliquant la métaphore de la "fermeture éclair", certaines couches sont publiques et constituent la base du système (niveaux S à N+1) jusqu'à un point de branchement (niveau N+1) à partir duquel les couches sont privées pour chaque utilisateur.

Remettant en cause cette métaphore de la “fermeture éclair”, nous envisageons plusieurs points de branchement au sein de la pile de composants, afin d’en étudier l’implication sur l’activité de groupe.

Par exemple, considérons une architecture en X comme celle de la Figure 2. Cette architecture peut traduire une situation où deux individus ne disposent que d’un seul crayon pour que chacun puisse écrire son propre texte dans un temps limité. Il s’agit en l’occurrence d’une activité collaborative synchrone fondée entièrement sur le partage. Transposons cet exemple dans l’univers du logiciel : l’architecture logicielle du collecticiel correspondant ressemble à X. En effet, le Noyau Fonctionnel, c’est-à-dire le texte écrit, n’est pas public puisque chaque utilisateur écrit son propre texte. De même, la présentation reste un composant privé puisque chacun dispose de sa propre feuille. Seul le crayon est public, car partagé, et conditionne l’interaction en la couplant très fortement.

Etudier des architectures logicielles remettant en cause le point de branchement unique qui délimite les composants publics de ceux privés, constitue un moyen pour définir de nouvelles formes de collaboration.

Composants collaboratifs

Nous nous sommes reposés sur notre modèle d’architecture conceptuelle Clover pour définir la plate-forme Clover de développement de collecticiels. Cette plate-forme génère des collecticiels dont l’architecture vérifie le modèle Clover.

Une autre façon d’exploiter notre modèle Clover pour la réalisation logicielle est de définir des composants logiciels réutilisables dont la structure repose sur le modèle Clover. En effet ce dernier préconise des composants logiciels constitués d’une abstraction encapsulant trois sous-composants dédiés respectivement à la production, à la communication et à la coordination. Nous envisageons d’appliquer cette structuration générique d’un composant du collecticiel à un collecticiel complet, pour définir un composant collaboratif au sens de la programmation par composants :

> “*un composant est un fragment de logiciel assez petit pour qu’on puisse le créer et le maintenir, et assez grand pour qu’on puisse l’installer et en assurer le support. De plus, il est doté d’interfaces standard pour pouvoir interopérer.*” [Orfali 1999]

Aussi, l’objectif serait de proposer des composants collaboratifs dotés d’un modèle de collaboration, véhiculant des propriétés ergonomiques, sociales et éthiques, et interopérables avec d’autres composants collaboratifs. En particulier, il semble nécessaire de définir une spécification précise et rigoureuse de ces composants, de telle sorte que l’agencement de deux composants collaboratifs auto-produise un modèle de collaboration cohérent, tout en respectant des propriétés.

Par exemple, plusieurs outils étudiés dans ce manuscrit visent à extraire l'activité de coordination des collecticiels. Le but est d'offrir un moteur de coordination externe aux applications : ce moteur a la charge de réguler les interactions parmi des applications existantes connues de l'utilisateur, comme un tableau blanc partagé ou un système de vidéoconférence. Cette approche autorise aussi l'utilisation d'applications mono-utilisateur dans un contexte collaboratif (à mi-chemin entre la collaboration transparente et la collaboration implicite). Lors de la réutilisation de collecticiels, cette approche peut aboutir à des incompatibilités dues à un moteur de coordination différent de ceux des collecticiels existants. Le développement de composants collaboratifs, tel que nous venons de l'évoquer, offre une alternative intéressante pour éliminer cette incompatibilité potentielle : le collecticiel, assimilé à un composant collaboratif, est alors interopéré avec le moteur de coordination dont la fusion produirait automatiquement un modèle de coordination cohérent.

Cette approche nécessite bien sûr de produire des composants logiciels collaboratifs capables de décrire leur modèle de collaboration (réflexivité) et de le modifier automatiquement dès qu'ils sont combinés avec d'autres composants. Il en est de même pour les propriétés véhiculées par le collecticiel, et donc le composant.

3.2. PASSAGE DE LA CONCEPTION À LA RÉALISATION D'UN COLLECTICIEL

Lors de notre étude, nous avons jeté un regard nouveau sur les outils de réalisation logicielle, en étudiant comment de tels outils dédiés au développement du logiciel peuvent intégrer des aspects de la conception comme ceux relevant de l'ergonomie ou de l'activité de groupe.

Dans [Bass 2001] nous trouvons une approche récente pour coupler les aspects d'utilisabilité et les aspects d'architecture logicielle. Les objectifs sont donc similaires à notre démarche de recherche, mais l'approche est dédiée aux interfaces mono-utilisateur. Elle repose sur des scénarios compréhensibles par les concepteurs ergonomes et psychologues, tout comme les concepteurs de logiciels. Ces scénarios constituent le point d'ancrage entre les aspects d'utilisabilité et les mécanismes architecturaux.

Exemple de scénario pour systèmes mono-utilisateurs

Par exemple, le scénario suivant traite de l'observabilité de l'état interne du système (scénario No 17) :

> "***Observing System State**:*
> *A user may not be presented with the system shared data necessary to operate the system (e.g., uninformative error messages, no file size given for folders). Alternatively, the system state may be presented in a way that violates human tolerances (e.g., is presented too quickly for people to be read). The system state may also be presented in an unclear*

> *fashion, thereby confusing the user. System designers should account for human needs and capabilities when deciding what aspects of the system state to display and how to present them.*"

Ce scénario traduit la capacité d'un système à rendre observable son état interne (propriété ergonomique relative à l'observabilité et à l'honnêteté). Le bénéfice escompté en terme d'utilisabilité est d'améliorer le confort et la confiance de l'utilisateur vis-à-vis du système ("*Increases confidence and comfort*"). Ainsi le mécanisme architectural proposé est de stocker les données indépendamment du reste du système ("*Separation / Data from Commands*") dans un composant spécifique et accessible par l'utilisateur.

Extension de l'approche par scénarios aux collecticiels

Nous envisageons d'étendre cette approche dédiée aux interfaces mono-utilisateurs aux cas des collecticiels. Notre étude a permis d'identifier un ensemble de liens entre les aspects d'utilisabilité et les aspects d'architecture logicielle que nous pourrions traduire dans le format (scénario, mécanisme architectural).

Par exemple, nous pourrions ajouter le cas suivant au tableau déjà établi dans [Bass 2001] avec le scénario suivant :

> "*L'utilisateur souhaite pouvoir masquer un ensemble d'informations personnelles aux autres membres du groupe. Il souhaite aussi pouvoir changer le niveau d'observabilité de ces informations selon sa situation courante.*"

Du point de vue utilisabilité, ce scénario est mis en relation avec les bénéfices escomptés d'utilisabilité, que nous assimilons ici à la propriété de respect de la vie privée.

Du point de vue architectural, le scénario ci-dessus est mis en relation avec le mécanisme architectural suivant :

- un composant privé qui maintient les informations personnelles. Ce composant est la propriété exclusive de l'utilisateur. Il inclut des fonctions permettant à l'utilisateur de spécifier un niveau d'observabilité de ces informations,

- un mécanisme de filtrage des informations personnelles permettant de fixer le niveau d'observabilité des informations avec la possibilité de modifier la politique de filtrage ainsi mise en œuvre. Le mécanisme de filtrage indépendamment d'une application particulière manipule des variables comme : l'information à filtrer, le type de filtre à appliquer, les paramètres du filtre ou encore le degré d'observabilité de l'information.

Bibliographie

Références

[Abdel-Wahad 1999] Abdel-Wahad, Hussein, Kim, Okhee, Kabore, Paul et Favreau, Jean-Philippe. Java-Based Multimedia Collaboration and Application Sharing Environment. Actes du *Colloque Francophone sur L'Ingénierie des Protocoles (CFIP'99)*, 1999.

[Anderson 1994] Anderson, Robert. Representations and Requirements: The value of Ethnography in System Design. Journal, *Human-Computer Interaction (HCI)*, 1994, volume 9, numéro 2, pages 151-182, Lawrence Erlbaum Associates.

[Anderson 2000] Anderson, Gary, Graham, Nicholas et Wright, Timothy. Dragonfly: Linking Conceptual and Implementation Architectures of Multiuser Interactive Systems. Actes de la *conférence International Conference on Software Engineering (ICSE'00)*, 2000, pages 252-261, ACM Press.

[Backer 2001] Backer, Kevin, Greenberg, Saul et Gutwin, Karl. Heuristic Evaluation of Groupware Based on the Mechanics of Collaboration. Actes de la *conférence IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'01)*, 2001, pages 123-140, Springer-Verlag.

[Baecker 1992] Baecker, Ronald, Nastos, Dimitris, Posner, Ilona et Mawlby, Kelly. The user-centered iterative design of collaborative writing. Actes du groupe de travail *Workshop on Real Time Group Drawing and Writing Tools*, en marge de la *conférence ACM Computer-Supported Cooperative Work (CSCW'92)*, 1992.

[Bannon 1998] Bannon, Liam. CSCW: Towards a Social Ergonomics? Présentation, *NATO Human Factors and Medicine Panel meeting on Collaborative Crew Performance in Complex Operational Systems*, 1998, Edimburgh, Ecosse.

[Barnard 1987] Barnard, Philip. Cognitive Resources and the Learning of Computer Dialogs. Livre, *Interfacing Thought, Cognitive aspects of Human Computer Interaction (Caroll éditeur)*, 1987, chapitre 6, pages 112-158, MIT Press.

[Bass 1992] Bass, Len, Faneuf, Ross, Little, Reed, Mayer, Niels, Pellegrino, Bob, Reed, Scott, Seacord, Robert, Sheppard, Sylvia et Szczur, Martha. A Metamodel for the Runtime Architecture of an Interactive System. Journal, *ACM Special Interest Group Computer-Human Interface bulletin (SIGCHI)*, 1992, volume 24, numéro 1, pages 32-37, ACM Press.

[Bass 1998] Bass, Len, Clements, Paul et Kazman, Rick. Livre, *Software Architecture in Practice*, 1998, 452 pages, Addison-Wesley.

[Bass 2001] Bass, Len, John, Bonnie et Kates, Jesse. Achieving Usability Through Software Architecture. *Rapport Technique*, Institut d'Ingénierie Logicielle, Carnegie Mellon, Pittsburg, Etats-Unis, 2001, référence CMU/SEI-2001-TR-005, 86 pages.

[Begole 1999] Begole, James, Rosson, Mary et Shaffer, Clifford. Flexible Collaboration Transparency: Supporting Worker Independence in Replicated Application-Sharing Systems. Journal, *ACM Transactions on Computer Human Interaction (ToCHI)*, 1999, volume 6, numéro 2, pages 95-132, ACM Press.

[Boehm 1981] Boehm, Barry. Livre, *Software Engineering Economics*, 1981, 767 pages, Prentice-Hall.

[Bourguin 2000] Bourguin, Grégory. Un support informatique à l'activité coopérative fondé sur la théorie de l'activité : le projet DARE. *Thèse de doctorat Informatique*, Université des sciences et technologies de Lille, Lille, France, Juillet 2000, 215 pages.

[Bowers 1995] Bowers, John, Button, Graham. Workflow from Within and Without. Actes de la *conférence European Conference on Computer Supporter Cooperative Work (ECSCW'95)*, 1995, pages 51-66, Kluwer Academic.

[Brave 1998] Brave, Scott, Ishii, Hiroshi et Dahley, Andrew. Tangible interfaces for remote collaboration and communication. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'98)*, 1998, pages 169-178, ACM Press.

[Calvary 1997] Calvary, Gaëlle, Coutaz, Joëlle et Nigay, Laurence. From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW. Actes de la *conférence ACM Conference on Human Factors and Computing Systems (CHI'97)*, 1997, pages 242-249, ACM Press.

[Card 1983] Card, Stuart, Moran, Thomas et Newell, Allen. Livre, *The Psychology of Human-Computer Interaction*, 1983, 469 pages, Lawrence Erlbaum Associates.

[Choudary 1995] Choudhary, Rajiv et Dewan, Prasun. A General Multi-User Undo/Redo Model. Actes de la *conférence European Conference on Computer Supporter Cooperative Work (ECSCW'95)*, 1995, pages 231-246, Kluwer Academic.

[Coutaz 1984] Coutaz, Joëlle, Herrmann, Mark. Adèle et le Médiateur-Compositeur ou Comment rendre une Application Interactive indépendante de l'Interface Usager. Actes du *deuxième colloque de Génie Logiciel (AFCET'84)*, 1984, pages 195-212.

[Coutaz 1990] Coutaz, Joëlle. Livre, *Interfaces homme-ordinateur : conception et réalisation*, 1990, 455 pages, Dunod.

[Coutaz 1997] Coutaz, Joëlle, Crowley, James et Bérard, François. Eigen Space Coding as a means to Support Privacy in Computer Mediated Communication. Actes de la *conférence IFIP International Conference on Human-Computer Interaction (INTERACT'97*), 1997, pages 532-538, Chapman&Hall.

[Coutaz 1999] Coutaz, Joëlle, Bérard, François, Carraux, Eric, Astier, William. CoMedi: Using Computer Vision to Support Awareness and Privacy in Mediaspaces. Actes (extension) de la *conférence ACM Conference on Human Factors and Computing Systems (CHI'99)*, 1999, pages 13-14, ACM Press.

[Coutaz 2001] Coutaz, Joëlle. Architecture logicielle conceptuelle des systèmes interactifs. Livre, *Analyse et conception de l'IHM : interaction homme-machine pour les SI (Kolski éditeur)*, 2001, volume 1, chapitre 7, 256 pages, Hermès.

[Dahlbäck 1993] Dahlbäck, Nils, Jönsson, Arne et Ahrenberg, Lars. Wizard of Oz Studies -- Why and How. Actes de la *conférence ACM International Workshop on Intelligent User Interfaces (IWIUI'93)*, 1993, pages 193-200, ACM Press.

[Dewan 1995] Dewan, Prasun et Choudhary, Rajiv. Coupling the User Interfaces of a Multiuser Program. Journal, *ACM Transactions on Computer Human Interaction (ToCHI)*, 1995, volume 2, numéro 1, pages 1-39, ACM Press.

[Dewan 1999] Dewan, Prasun. Architectures for Collaborative Applications. Livre, *Computer-Supported Cooperative Work (Beaudouin-Lafon éditeur), Trends in Software*, pages 169-194, John Wiley & Sons.

[Dewan 2001] Dewan, Prasun. An Integrated Approach to Designing and Evaluating Collaborative Applications and Infrastructures. Journal, *Journal of Computer-Supported Cooperative Work (JCSCW)*, 2001, volume 10, numéro 1, pages 75-111, ACM Press.

[Dix 1993] Dix, Alan, Finlay, Janet, Abowd, Gregory et Beale, Russel. Livre, *Human-Computer Interaction*, 1993, 570 pages, Prentice-Hall.

[Dourish 1992] Dourish, Paul et Bly, Sara. Portholes: Supporting Awareness in a Distributed Work Group. Actes de la *conférence ACM Conference on Human Factors and Computing Systems (CHI'92)*, 1992, pages 541-547, ACM Press.

[Dourish 1996a] Dourish, Paul. Open Implémentation and Flexibility in CSCW toolkits. *Thèse de doctorat en informatique*, University College of London, Londres, Royaume-Uni, Juin 1996, 132 pages.

[Dourish 1996b] Dourish, Paul. Consistency Guarantees : Exploiting Application Semantics Consistency Management in a Collaboration Toolkit. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'96)*, 1996, pages 268-277, ACM Press.

[Edwards 1996a] Edwards, Keith. Coordination Infrastructure in Collaborative Applications. *Thèse de doctorat en informatique*, College of Computing, Georgia Institute of Technologie, Atlanta, Georgie, USA, 1996, 148 pages.

[Edwards 1996b] Edwards, Keith. Policies and Roles in Collaborative Applications. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'96)*, 1996, pages 11-20, ACM Press.

[Ellis 1991] Ellis, Clarence, Gibbs, Simon et Rein, Gail. Groupware: Some Issues and Experiences. Journal, *Communications of the ACM (CACM)*, 1991, volume 34, numéro 1, pages 38-58, ACM Press.

[Ellis 1994] Ellis, Clarence et Wainer, Jacques. A Conceptual Model of Groupware. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'94)*, 1994, pages 79-88, ACM Press.

[Ellis 1999] Ellis, Clarence. Workflow Technology. Livre, *Computer-Supported Cooperative Work (Beaudouin-Lafon éditeur), Trends in Software*, pages 29-54, John Wiley & Sons.

[Flores 1988] Flores, Fernando, Graves, Michael, Hartfield, Brad et Winograd, Terry. Computer systems and the design of organizational interaction. Journal, *Transactions on Information Systems (ToIS)*, 1988, volume 6, numéro 2, pages 153-172, ACM Press.

[Finn 1997] Finn, Kathleen, Sellen, Abigail et Wilbur, Sylvia. Livre, *Video-mediated communication*, 1997, 584 pages, Lawrence Erlbaum Associates.

[Fitts 1954] Fitts, Paul. The information capacity of the human motor system in controlling the amplitude of movement. Journal, *Journal of Experimental Psychology (JEP)*, 1954, volume 47, numéro 6, pages 381-391.

[Graham 1996] Graham, Nicholas, Morton, Catherine et Urnes, Tore. ClockWorks: Visual programming of component-based software architectures. Journal, *Journal of Visual Languages and Computers (JVLC)*, 1996, volume 7, numéro 2, pages 175-196, Academic Press.

[Graham 1997] Graham, Nicholas et Urnes, Tore. Integrating Support for Temporal Media into an Architecture for Graphical User Interfaces. Actes de la *conférence International Conference on Software Engineering (ICSE'97)*, 1997, pages 172-182, ACM Press.

[Green 1985] Green, Mark. Report on Dialogue Specification Tools. Journal, *User Interface Management Systems (UIMS), Eurographics Seminars*, 1985, pages 9-20, Springer-Verlag.

[Green 1990] Green, Thomas. The cognitive dimension of viscosity: A sticky problem for HCI. Actes de la *conférence IFIP International Conference on Human-Computer Interaction (INTERACT'90)*, 1990, pages 76-86, Elsevier.

[Greenberg 1998] Greenberg, Saul et Roseman, Mark. Using a Room Metaphor to Ease Transitions in Groupware. *Rapport de recherche*, Département d'informatique, Université de Calgary, Calgary, Canada, 1998, référence 98/611/02, 31 pages.

[Grudin 1994] Grudin, Jonathan. CSCW: History and Focus. Jounal, *IEEE Computer*, 1994, volume 27, numéro 5, pages 19-26, IEEE.

[Gutwin 2000] Gutwin, Carl et Greenberg, Saul. The Mechanics of Collaboration: Developing Low Cost Usability Evaluation Methods for Shared Workspaces. Actes de la *conférence IEEE 9th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE'00)*, 2000, pages 98-103, IEEE.

[Hall 1996] Hall, Robert, Mathur, Amit, Jahanian, Farnam, Prakash, Atul et Rassmussen, Craig. Corona: A Communication Service for Scalable, Reliable Group Collaboration Systems. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'96)*, 1996, pages 140-149, ACM Press.

[Harrison 1996] Harrison, Steve et Dourish, Paul. Re-Place-ing Space: The Roles of Place and Space in Collaborative Systems. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'96)*, 1996, pages 67-76, ACM Press.

[Hill 1994] Hill, Ralph, Brink, Tom, Rohal, Steven, Patterson, John et Wayne, Wilner. The RendezVous Architecture and Language for Constructing Multi-User Applications. Journal, *ACM Transactions on Computer Human Interaction (ToCHI)*, 1994, volume 1, numéro 2, pages 81-125, ACM Press.

[Hoogstoel 1995] Hoogstoel, Frédéric. Une approche organisationnelle du travail coopératif assisté par ordinateur : Application au projet Co-Learn. *Thèse de doctorat Informatique*, Université de Lille 1, Lille, France, Novembre 1995.

[IFIP 1996] IFIP Working group 2.7 (co-auteur). Livre, *Design Principles for Interactive Software (Gram et Cockton éditeurs)*, 1996, 248 pages, Chapman & Hall.

[Ishii 1994] Ishii, Hiroshi, Kobayashi, Minoru et Arita, Kazuho. Iterative design of Seamless Collaborative Media. Journal, *Communication of the ACM (CACM)*, 1994, volume 37, numéro 8, pages 83-97, ACM Press.

[Karsenty 1994] Karsenty, Alain. Le collecticiel : de l'interaction homme-machine à la communication homme-machine-homme. Journal, *Technique et Science Informatiques (TSI)*, 1994, volume 13, numéro 1, pages 105-127, Hermès.

[Karsenty 1997] Karsenty, Laurent et Pavard, Bernard. Différents Niveaux d'Analyse du Contexte dans l'Etude Ergonomique du Travail Collectif. Journal, *revue Réseaux*, 1997, numéro 85, pages 73-99, Hermès.

[Kiczales 1992] Kickzales, Gregor. Towards a New Model of Abstraction in Software Engineering. Actes de la *conférence International Workshop on New Models in Software Architecture (IMSA'92) on Reflection and Meta-Level Architectures*, 1992.

[Kiczales 1996] Kiczales, Gregor. Beyond the Black Box: Open Implementation. Journal, *IEEE Software*, 1996, volume 13, numéro 1, pages 8-11, IEEE.

[Kiczales 1997] Kiczales, Gregor, Lamping, John, Lopes, Cristina, Maeda, Chris, Mendhekar, Anurag et Murphy, Gail. Open implementation design guidelines. Actes de la *conférence ACM International Conference on Software Engineering (ICSE'97)*, 1997, pages 481-490, ACM Press.

[Krasner 1988] Krasner, Glenn et Pope, Stephen. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. Journal, *Journal of Object Oriented Programming (JOOP)*, 1988, volume 1, numéro 3, pages 26-49.

[Kuuti 1991] Kuuti, Kari. The Concept of Activity as a Basic Unit of Analysis for CSCW research. Actes de la *conférence European Conference on Computer Supporter Cooperative Work (ECSCW'91)*, 1991, pages 249-264, Kluwers Academics.

[Kyng 1997] Kyng, Morten et Mathiassen, Lars. Livre, *Computers and Design in Context*, 1997, 418 pages, MIT Press.

[Laurillau 1999a] Laurillau, Yann. Techniques de Navigation Collaborative. *Mémoire de Diplôme d'Etudes Approfondies (DEA) en informatique*, Université Joseph Fourier, Grenoble, France, Juin 1999, 136 pages.

[Laurillau 1999b] Laurillau, Yann. Synchronous Collaborative Navigation Techniques on the WWW. Actes de la *conférence ACM Conference on Human Factors and Computing Systems (CHI'99)*, 1999, pages 308-309, ACM Press.

[Laurillau 1999c] Laurillau Yann. Techniques de Navigation Collaborative Synchrone sur le WWW. Actes de la *conférence Francophone Interaction Homme-Machine (IHM'99)*, 1999, pages 116-117, Cépadues.

[Laurillau 2000] Laurillau, Yann et Nigay, Laurence. Modèle de Navigation Collaborative Synchrone pour l'Exploration des Grands Espaces d'Information. Actes de la *conférence Francophone Interaction Homme-Machine (IHM'00)*, 2000, pages 121-128, CRT ILS&ESTIA.

[Laurillau 2002a] Laurillau, Yann et Nigay, Laurence. Clover Architecture for Groupware. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'02)*, 2002, À paraître.

[Laurillau 2002b] Laurillau, Yann et Nigay, Laurence. Architecture Clover pour les Collecticiels. Actes de la *conférence francophone ACM Interaction Homme-Machine (IHM'02)*, 2002, À paraître.

[Li 1998] Li, Du et Muntz, Richard. COCA: Collaborative Objects Coordination Architecture. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'98)*, 1998, pages 179-188, ACM Press.

[Li 1999] Li, Du, Wang, Zhenghao et Muntz, Richard. "Got COCA?" A New Perspective in Building Electronic Meeting Systems. Actes de la *conférence ACM International Conference on Work activities Coordination and Collaboration (WACC'99)*, 1999, pages 89-98, ACM Press.

[Malone 1990] Malone, Thomas et Crowston, Kevin. What is Coordination Theory and How can it help Design cooperative Work Systems ? Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'90)*, 1990, pages 289-297, ACM Press.

[McCall 1977] MacCall, James. Quality Factors. Livre, *Encyclopedia of Software Engineering (Marciniak éditeur)*, 1994, pages 958-969, John Wiley & Sons.

[McDermid 1984] McDermid, John et Ripkin, Knut. Livre, *Life Cycle Support in the ADA environment*, 1984, 259 pages, Cambridge University Press.

[Mackay 1999] Mackay, Wendy. Media Spaces: Environments for Informal Multimedia Interaction. Livre, *Computer-Supported Cooperative Work (Beaudouin-Lafon éditeur), Trends in Software*, pages 55-82, John Wiley & Sons.

[Nielsen 1990] Nielsen, Jakob et Molich, Rolf. Heuristic Evaluation of User Interfaces. Actes de la *conférence ACM Conference on Human Factors and Computing Systems (CHI'90)*, 1990, pages 249-256, ACM Press.

[Nigay 1994] Nigay, Laurence. Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales. *Thèse de doctorat Informatique*, Université Joseph Fourier, Grenoble, France, Janvier 1994, 363 pages.

[Nigay 1997] Nigay, Laurence et Coutaz, Joëlle. Software architecture modelling: Bridging Two Worlds using Ergonomics and Software Properties. Livre, *Formal Methods in Human-Computer Interaction (Palanque et Paterno éditeurs)*, 1997, pages 49-73, Springer Verlag.

[Nigay 1998] Nigay, Laurence et Vernier, Frédéric. Design Method of Interaction Techniques for Large Information Spaces. Actes de la *conférence ACM Conference in Advanced Visualization for Interfaces (AVI'98)*, Mai 1998, pages 37-46, ACM Press.

[Nigay 2001] Nigay, Laurence. *Thèse d'habilitation à diriger des recherches en informatique*, Université Joseph Fourier, Grenoble, France, Décembre 2001.

[Orfali 1999] Orfali, Robert, Harkey, Dan et Edwards, Jeri. Livre, *Client/Serveur : Guide de survie*, troisième édition, traduction de François Leroy et Jean-Pierre Gout, 1999, 782 pages, Vuibert.

[Ouadou 1994] Ouadou, Kamel. AMF : Un modèle d'architecture multi-agents multi-facettes pour Interfaces Homme-Machine et les outils associés. *Thèse de doctorat Informatique*, Ecole Centrale de Lyon, Lyon, France, 1994, 210 pages.

[Ousterhout 1994] Ousterhout, Jhon. Livre, *Tcl and the Tk Toolkit*, 1994, 458 pages, Addison-Wesley.

[Oviatt 1999] Oviatt, Sharon. Ten myths of multimodal interaction. Journal, *Communications of the ACM (CACM)*, 1999, volume 42, numéro 11, pages 74-81, ACM Press.

[Palen 2002] Palen, Leysia et Grudin, Jonathan. Discretionary Adoption of Group Support Software: Lessons from Calendar Applications. Livre, *Organizational implementation of collaboration technology (Munkvold éditeur)*, 2002, à paraitre.

[Patterson 1994] Patterson, John. A Taxonomy of Architectures for Synchronous Groupware Applications. Actes du groupe de travail *Workshop on Software Architectures for Cooperative Systems* en marge de la conférence *ACM Conference on Computer-Supported Cooperative Work (CSCW'94)*, 1994, pages 317-328, ACM Press.

[Patterson 1996] Patterson, John, Day, Mark et Kucan, Jakov. Notification Servers for Synchronous Groupware. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'96)*, 1996, pages 122-129, ACM Press.

[Phillips 1999] Phillips, Greg. Architectures for Synchronous Groupware. *Rapport Technique*, Département d'informatique, Université du Queens, Kingston, Canada, 1999, référence 1999-425, 53 pages.

[Prakash 1999] Prakash, Atul. Group Editors. Livre, *Computer-Supported Cooperative Work (Beaudouin-Lafon éditeur), Trends in Software*, pages 103-133, John Wiley & Sons.

[Ramduny 1998] Ramduny, Devina, Dix, Alan et Rodden, Tom. Exploring the design space for notification servers. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'98)*, 1998, pages 227-235, ACM Press.

[Reynard 1998] Reynard, Gail, Benford, Steve, Greenhalgh, Chris et Heath, Christian. Awareness Driven Video Quality of Service in Collaborative Virtual Environnments. Actes de la *conférence ACM Conference on Human Factors and Computing Systems (CHI'98)*, 1998, pages 464-471, ACM Press.

[Robert 1992] Robert, Paul. Le petit Robert. Livre, *Dictionnaire de la langue Française*, 1992, 2171 pages, Le Robert.

[Roseman 1992] Roseman, Mark et Greenberg, Saul. GroupKit: A groupware Toolkit for Building Real-Time Conferencing Applications. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'92)*, 1992, pages 43-50, ACM Press.

[Roseman 1996a] Roseman, Mark et Greenberg, Saul. Building Real-Time Groupware with GroupKit, A Groupware Toolkit. Journal, *ACM Transactions on Computer Human Interaction (ToCHI)*, 1996, volume 3, numéro 1, pages 66-106, ACM Press.

[Roseman 1996b] Roseman, Mark et Greenberg, Saul. TeamRooms: Network Places for Collaboration. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'96)*, 1996, pages 325-333, ACM Press.

[Royce 1970] Royce, Walker. Managing the development of large software systems. Actes de la *conférence IEEE Western Electronic Show Convention (WESTCON*), 1970, pages 1-9, IEEE. Version réimprimée dans les actes de la *conférence ACM International Conference on Software Engineering (ICSE'87)*, 1987, pages 328-338, ACM Press.

[Salber 1995] Salber, Daniel. De l'interaction individuelle aux systèmes multi-utilisateurs. L'exemple de la Communication Homme-Homme-Médiatisée. *Thèse de doctorat Informatique*, Université Joseph Fourier, Grenoble, France, Septembre 1995, 303 pages.

[Salvador 1996] Salvador, Tony, Scholtz, Jean et Larson, James. The Denver Model for Groupware Design. Journal, *ACM Special Interest Group Computer-Human Interface bulletin (SIGCHI)*, 1996, volume 28, numéro 1, édition en ligne, ACM Press.

[Scapin 1989] Scapin, Dominique et Pierret-Golbreich, Christine. Towards a method for task description. Actes de la conférence *Working With Display Units (WWDU'89)*, 1989, pages 371-380.

[Shepherd 1989] Shepherd, Andrew. Analysis and Training in Information Technology Tasks. Livre, *Task Analysis for Human-Computer Interaction (Diaper éditeur)*, 1989, chapitre 1, pages 15-55, Ellis Horwood.

[Steele 1990] Steele, Guy. Livre, *Common Lisp: The Language*, 1990, 1029 pages, Digital Press.

[Stefik 1987] Stefik, Mark, Bobrow, Daniel, Foster, Gregg, Lanning, Stan et Tatar, Deborah. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. Journal, *ACM Transactions on Computer Human Interaction (ToCHI)*, 1987, volume 5, numéro 2, pages 147-167, ACM Press.

[Suchman 1987] Suchman, Lucy. Livre, *Plans and Situated Actions: The problem of human machine communication*, 1987, 203 pages, Cambridge University Press.

[Tarpin 1997] Tarpin-Bernard, Franck. Travail coopératif synchrone assisté par ordinateur : Approche AMF-C. *Thèse de doctorat Informatique*, Ecole Centrale de Lyon, Lyon, France, Juillet 1997, 147 pages.

[Van Der Veer 1996] Van Der Veer, Gerrit, Lenting, Bert et Bergevoet, Bas. GTA: Groupware Task Analysis - Modeling Complexity. Journal, *Acta Psychologica*, 1996, volume 91, pages 297-32.

[Whittaker 2000] Whittaker, Steve, Nardi, Bonnie et Bradner, Erin. Interaction and Outeraction: Instant Messaging in Action. Actes de la *conférence ACM Computer Supported Cooperative Work (CSCW'00)*, 2000, pages 79-88, ACM Press.

[Zafer 2001] Zafer, Ali. NetEdit: a Collaborative Editor. *Mémoire de master scientifique en informatique (Master of Science in Computer Science)*, Université de Viginie, Blacksburg, Etats-Unis, 23 Avril 2001, 82 pages.

Sites WWW

[Apple 2002] Apple corporation
http://www.apple.com/

[ARToolkit 2002] ARToolkit,
http://www.cs.nps.navy.mil/people/faculty/capps/4473/projects/01Summer/AR/AR_Simply/ARToolKit.htm

[Assali 1998] Alissali, Mamoun. Introduction au Génie Logiciel. Support de cours en ligne, 1998, Université du Maine, Le Mans
http://www-ic2.univ-lemans.fr/~alissali/Enseignement/Polys/GL/gl.html

[Blizzard 2002] Blizzard, StarCraft,
http://www.blizzard.com/worlds-starcraft.shtml

[CoVitesse 2002] CoVitesse,
http://iihm.imag.fr/demos/CoVitesse/

[Eastgate 2002] Eastgate,
http://www.eastgate.com/

[IDC 2002] IDC Consulting,
http://www.businessweek.com/technology/content/dec2001/tc20011213_7143.htm

[Id Software 2002] Id Software, Quake,
http://www.idsoftware.com/quake/

[IRC 2002] IRChelp.org, IRC (Internet Relay Chat)
http://www.irchelp.org/

[Java 2002] Sun microsystems, Java, web site,
http://www.javasoft.com/

[Lotus 2002] Lotus,
http://www.lotus.com/

[MIT Medialab 2002] MIT, Media Laboratory, Facilitator Room, http://www-white.media.mit.edu/facilitator/introduction.html

[Netopia 2002] Netopia, Timbuktu, http://www.netopia.com/

[PictureTel 2002] PictureTel corporation, http://www.picturetel.com/

[Qualcomm 2002] Qualcomm, Eudora, http://www.qualcomm.com/

[Sun 2002] Sun Microsystems http://www.sun.com/

[UsabilityFirst 2002] Usability First; groupware principles, http://www.usabilityfirst.com/glossary/cat_40.txl

[Vitesse 2002] Système Vitesse, logiciel de visualisation d'une grande quantité d'information, http://iihm.imag.fr/demos/VitesseDemos.html

[VNC 2002] AT&T labs, Virtual Network Computing, http://www.uk.research.att.com/vnc/

[X-Windows 2002] X.Org consortium, X-Windows system, http://www.x.org/

Annexe

Types de navigation collaborative

Cette annexe détaille une partie des travaux menés sur la navigation collaborative [Laurillau 1999a] [Laurillau 2000] dans de grands espaces d'information. La première partie présente une définition des quatre types de navigation collaborative et la seconde présente la caractérisation de ces types de navigation à l'aide du modèle Denver.

1. Quatre Types de navigation collaborative

Les quatre types de navigation sont définis comme suit :

- **Visite guidée** :
  Inspirée de la visite guidée d'un musée ou d'un site, les utilisateurs membres du groupe cèdent le contrôle de l'interaction à un guide qui les emmène découvrir l'espace d'information. A tout moment un utilisateur peut joindre ou quitter la visite guidée.

- **Navigation relâchée** :
  La rencontre est informelle, et chacun navigue indépendamment les uns des autres, mais dispose des résultats collectés par les membres du groupe. N'importe qui peut devenir un membre et, à n'importe quel moment, il est possible de donner le contrôle de la navigation à un autre.

- **Navigation coordonnée** :
  Les membres du groupe travaillent séparément, dans un espace

restreint ou non, sur un sujet commun, mais sans forcément avoir exactement les mêmes buts. Tous ont les mêmes droits, et n'importe qui peut décider si un nouveau venu peut devenir ou non un membre.

- **Navigation coopérative** :
  Un responsable de groupe oriente la recherche. Il lui incombe la responsabilité de définir les objectifs et les stratégies à suivre, de manière autoritaire ou concertée, et de répartir, si cela est nécessaire, les sous-espaces à explorer. Il est le seul à décider si un nouveau venu peut devenir membre. Il a de plus la possibilité de déplacer ("téléporter") l'ensemble du groupe dans un endroit qu'il peut juger intéressant.

Afin de détailler les interactions pour chaque type de navigation, nous appliquons le modèle *Denver*, que nous décrivons dans la partie suivante.

2. *Modélisation des interactions avec le modèle Denver*

Le sous-modèle le plus détaillé est le modèle de conception (*Design Model*). Ce sous-modèle définit un cadre de conception d'un collecticiel, par cinq catégories de caractéristiques :

- **Les Personnes** (*People*) : cette catégorie inclut les caractéristiques des utilisateurs, des groupes et des rôles qu'une personne peut avoir au sein d'un groupe.

- **Les Artéfacts** (*Artifacts*) : cette catégorie se réfère à tous les objets manipulés, produits ou consommés pendant l'interaction. Par exemple un ensemble de pointeurs vers des pages Web constitue le résultat produit par un groupe, en phase de recherche d'information.

- **Les Tâches et Activités** (*Tasks and Activities*) : cette catégorie regroupe les caractéristiques des tâches et activités réalisables par le biais du collecticiel. Nos quatre types de navigation sont des éléments de cette catégorie.

- **Les Situations d'Interaction** (*Interactive Situations*) : Dans notre version adaptée du modèle, les situations d'interaction sont définies par cinq axes, comme le montre la Figure 1 :

  * **La taille d'un groupe** (*Size*) : celle-ci peut être grande ou petite.

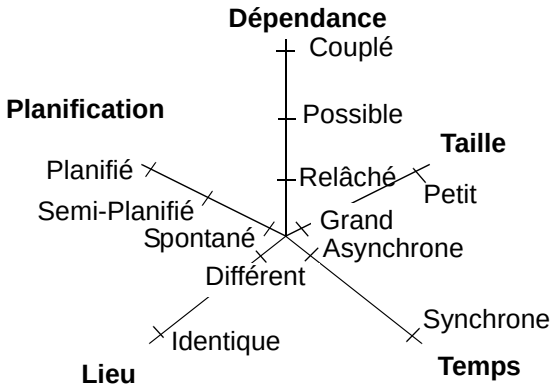


Figure 1
Les cinq axes caractérisant la situation d'interaction

* **Le lieu** (*Location*) : l'action peut se dérouler soit dans un même lieu, soit dans des lieux différents.

* **La dépendance** (*Dependency*) : trois types de dépendance sont identifiés au cours de l'interaction : soit les utilisateurs agissent ensemble de manière fortement couplée, soit la dépendance est possible (c'est-à-dire que l'on peut être dépendant de quelqu'un de manière ponctuelle, que ce soit volontaire ou non), soit elle est totalement relâchée (ou encore découplée).

* **Le niveau de planification de l'interaction** (*Timing*) : l'interaction peut être soit planifiée, soit semi-planifiée, soit spontanée.

* **Le temps** (*Time*) : l'interaction peut être réalisée de façon synchrone ou asynchrone.

- **Les Protocole Social de l'Interaction** (*Interactive Social Protocol*) : nous proposons aussi une version étendue de cette catégorie en décrivant le protocole social de l'interaction par six axes, comme le montre la Figure 2 :

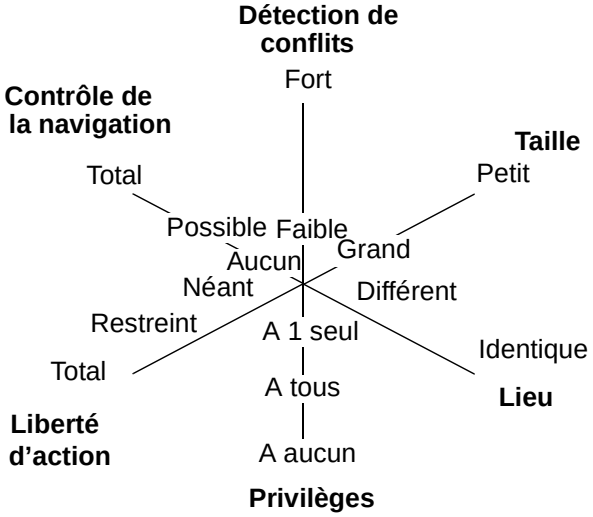


Figure 2
Les six axes caractérisant le protocole d'interaction

* **La taille d'un groupe** (*Size*) : celle-ci peut être grande ou petite.
* **Le lieu** (*Location*) : l'action peut se dérouler soit dans le même lieu, soit dans des lieux différents.
* **La détection de conflits** (*Contention Detection and Resolution*) : cet axe caractérise la liberté accordée aux utilisateurs pour résoudre des conflits potentiels.
* **Le contrôle de la navigation par un autre** : cet axe se substitue à l'axe de contrôle de l'interaction (*Floor Control*). Sur cet axe, la prise de contrôle peut être totale, possible ou impossible (aucune prise de contrôle).
* **La liberté d'action** : cet axe se substitue lui aussi à l'axe de style de la rencontre (*Meeting Style*). Il définit les libertés accordées à un utilisateur (par le système ou par un utilisateur privilégié) pendant la navigation. Cette caractérisque recouvre deux concepts : l'initiative et le contrôle des outils de navigation. Cependant cet axe n'est pas totalement orthogonal avec l'axe contrôle : dans le cas où le contrôle serait total, la liberté d'action est réduite à néant ; de même, dans le cas ou il n'y a aucune prise de contrôle possible, la liberté d'action est totale; enfin, lorsque la prise de contrôle est possible, la liberté d'action est alors quelconque (dans ce cas, les deux axes sont bien indépendants).
* **Les privilèges** : cet axe définit les rôles de certains utilisateurs au sein d'un groupe et des pouvoirs qui leur ont été accordés. Ces privilèges peuvent être accordés à aucun utilisateur, à tous les utilisateurs ou bien à un seul.

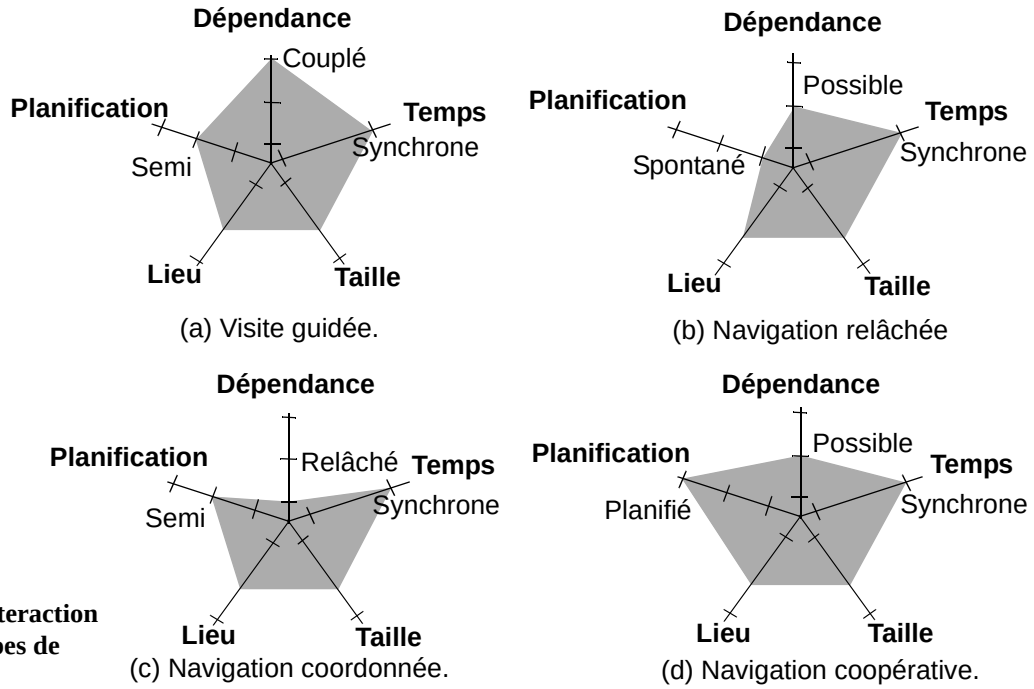
En appliquant notre version étendue du modèle Denver, nous détaillons nos quatre types de navigation, en décrivant leur situation et leur protocole de l'interaction.

2.1. SITUATIONS D'INTERACTION

Chaque polygone de la Figure 3 représente un type de navigation : visite guidée, navigation relâchée, coordonnée et coopérative. Les situations d'interaction explicitent ici les différences entre les types de navigation.

Toutes les situations ont en commun la taille du groupe (qui est quelconque), la localisation des personnes (du réseau local au réseau mondial) et la synchronisation. La différenciation se fait au niveau des deux axes "Planification" et "Dépendance" :

- **Planification** : une navigation coopérative (Figure 3 (d)) est une quête d'information qui doit être réfléchie, décidée avant d'agir, c'est-à-dire planifiée. A l'opposé, une navigation relâchée (Figure 3 (b)) est spontanée, c'est une réponse à une demande d'aide, c'est une offre



(a) Visite guidée. (b) Navigation relâchée

(c) Navigation coordonnée. (d) Navigation coopérative.

Figure 3
Les situations d'interaction pour les quatre types de navigation

spontanée à une personne que l'on rencontre. Entre les deux, la navigation coordonnée (Figure 3 (c)) et la visite guidée (Figure 3 (a)) sont semi-planifiées : en effet les groupes se constituent soit parce que les gens se connaissent en navigation coordonnée, soit parce que quelqu'un décide de devenir guide, et cela suppose qu'il sait où il va se déplacer pour la visite.

- **Dépendance du travail de chacun** : la dépendance la plus forte est celle qui existe entre les utilisateurs guidés et le guide. C'est le guide (Figure 3 (a)) qui décide où va le groupe. En fait, la production d'information est ici réalisée par la démarche du guide : il emmène le groupe vers des sources d'information. Au contraire, la dépendance est relâché dans une navigation coordonnée (Figure 3 (c)); tout le monde travaille de façon équivalente et indépendante : les collaborateurs d'une navigation coordonnée se découpent un espace d'information, puis chacun de leur côté, ils explorent individuellement le sous-espace qui leur a été attribué pour pouvoir ensuite communiquer les résultats de leur recherche. Entre les deux, il y a la navigation coopérative (Figure 3 (d)), où il y a un chef de session qui n'a pas autant d'emprise sur le groupe qu'un guide, mais qui peut tout de même regrouper tous les utilisateurs sur une source d'information intéressante. Parallèlement, dans une navigation relâchée (Figure 3 (b)), l'utilisateur dépend de la personne qu'il suit, mais il peut tout de même faire autre chose, comme "papillonner".

2.2. PROTOCOLE SOCIAL DE L'INTERACTION

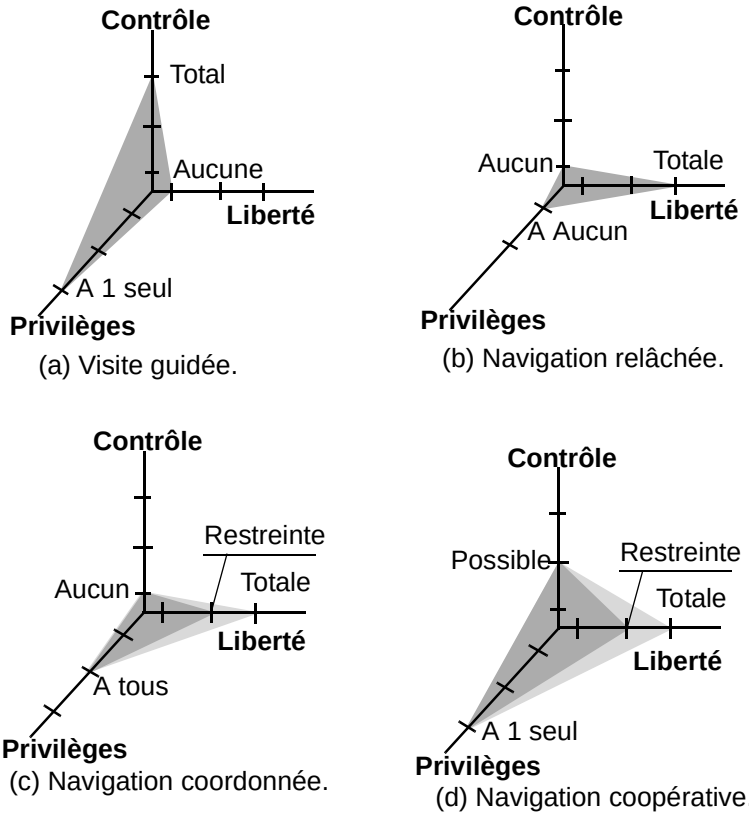


Figure 4
Les protocoles d'interaction des quatre types de navigation

En ce qui concerne les protocoles d'interaction, nous avons adapté les axes définis par le modèle Denver, tout en y ajoutant un nouveau : privilèges. En effet, nous avons défini l'axe "Contrôle de l'interaction" (*Floor Control*) par l'axe "Contrôle de la navigation par un autre", et l'axe "Style de la rencontre" (*Meeting Style*) par l'axe "Liberté d'action". L'axe "Privilèges" définit à quels utilisateurs sont attribués certains droits. Par exemple, cela peut être le droit de contrôler la navigation d'un autre.

Chaque polygone de la Figure 4 représente un type de navigation : visite guidée, navigation relâchée, coordonnée et coopérative. Tous ont en commun les caractéristiques suivantes (représentées sur les diagrammes de la Figure 4) : la taille du groupe (qui est quelconque), les formalismes de communication et la résolution de conflits. Les utilisateurs sont libres pour résoudre d'éventuels conflits, ceci justifie alors la valeur élevée attribuée à cette caractéristique (en fait valeur infinie). Nous avons en effet décidé de ne pas injecter de règles sociales dans notre système. Les différences entre types de navigation se font au niveau du contrôle des outils de navigation par le groupe, de la liberté d'action et des privilèges accordés à certains utilisateurs :

- Dans le cas de la visite guidée (Figure 4 (a)), le guide mène la navigation pour l'ensemble des membres. Il est le seul à posséder tous les droits (Figure 5). Ainsi son contrôle sur l'ensemble des membres est total et par conséquent les utilisateurs n'ont aucune liberté d'action sur la navigation.

- Dans le cas de la navigation relâchée (Figure 4 (b)), la liberté d'action pour tout utilisateur est totale, sachant que personne ne peut prendre le contrôle sur un autre (mais comme cela a été expliqué dans la description des situations d'interaction, l'utilisateur peut donner temporairement le contrôle à un autre). Il n'y a aucun privilège attribué à qui que ce soit.

- Dans le cas de la navigation coordonnée (Figure 4 (c)), le créateur du groupe peut décider si le système doit ou non restreindre automatiquement la zone à explorer pour chaque utilisateur. Ce choix se traduit à la Figure 4 par un enchevêtrement de zone grisée. Pour l'utilisateur, cette restriction se traduit par un espace d'information à visiter plus réduit. Dans ce type de navigation, tous les utilisateurs ont les mêmes privilèges, à savoir que n'importe quel membre peut décider, si quelqu'un peut devenir ou non un nouvel élément du groupe. D'autre part, personne ne peut prendre le contrôle de la navigation d'un autre.

- Dans le cas de la navigation coopérative (Figure 4 (d)), et contrairement aux types de navigation précédents, ce groupe dispose d'un unique responsable, disposant seul de tous les privilèges. Le chef de session (la Figure 5 indique les protocoles d'interaction qui caractérise son type particulier de navigation), créateur du groupe, peut à tout moment prendre le contrôle du groupe pour le téléporter là où il se trouve (d'où la valeur "possible" attribuée à cette navigation sur l'axe "contrôle de la navigation"). Le responsable peut décider s'il laisse une liberté d'action totale à tous les utilisateurs ou si, au contraire, il décide de partager l'espace. Dans ce dernier cas, c'est lui-même qui attribue les portions de l'espace à explorer (de même, cette possibilité se traduit dans la Figure 4 par différentes zones grisées).

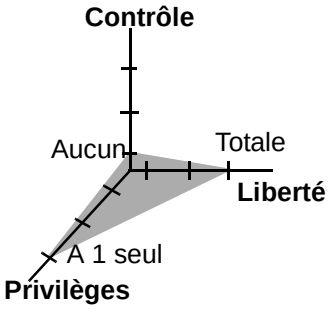


Figure 5
Situations d'interaction pour le guide et le chef de groupe
dans le cas de la visite guidée et dans le cas de la navigation coopérative

En appliquant le modèle Denver, nous avons caractérisé les types de navigation collaborative, du point de vue des interactions, situations et protocoles.

Résumé Mes travaux appartiennent au domaine de l'Interaction Homme-Machine et contribuent à l'ingénierie des systèmes interactifs multi-utilisateurs ou collecticiels. La conception et la réalisation logicielles d'un collecticiel soulèvent des problèmes propres à ce type de systèmes interactifs, plus complexes que les systèmes mono-utilisateur. Par exemple, il convient de gérer des sources d'événements différentes impliquant un phénomène de concurrence, de prendre en compte des nouvelles contraintes technologiques induites par les réseaux informatiques et de vérifier des nouvelles propriétés ergonomiques telles que le protection de la vie privée. Face à la complexité de réalisation logicielle, il est donc crucial de disposer d'outils telles que des modèles d'architecture logicielle et des plates-formes de développement.

Dans ce contexte, les contributions de ma thèse sont un modèle d'architecture pour les collecticiels et une infrastructure générique, la plate-forme Clover, pour le développement des collecticiels centré sur l'humain. Le modèle du trèfle, décrivant la couverture fonctionnelle d'un collecticiel (production, communication et coordination), a été retenu comme guide conceptuel pour l'élaboration de ce modèle d'architecture et de cette infrastructure. Une large majorité des travaux se sont concentrés sur la résolution de problèmes techniques tels que la fusion de données. L'approche retenue est complémentaire puisque centrée sur l'activité de groupe. Les requis retenus sont : offrir un niveau d'abstraction centré sur l'activité de groupe, opposé à centré sur la technique, et offrir une couverture fonctionnelle générique, couvrant tous les espaces du modèle du trèfle. L'étude d'un ensemble d'outils de développement a permis de montrer que les plates-formes de haut niveau d'abstraction n'offraient que des services de coordination, dont j'ai montré les limitations. Le système CoVitesse, logiciel de navigation collaborative sur le WWW, et un système de tableau blanc partagé ont été développés pour illustrer cette infrastructure.

Mots-clés Interaction Homme-Machine, collecticiels, TCAO, modèle d’architecture logicielle, plate-forme de développement, modèle du trèfle.

Abstract My work belongs to the Human-Computer Interaction research field and contributes to the field of multi-user interactive applications or groupware applications engineering. The software design and development of a groupware application raise specific problems more complex than those raised by single-user interactive applications. For example, in a groupware application, it is necessary (i) to manage multiple sources of events which generate concurrency situations, (ii) to face new technological constraints raised by networks and (iii) to take into account new kinds of human factors such as privacy. This complexity of development shows the importance of tools such as architectural models or software toolkits and infrastructures for groupware.

In this context, the main contributions of this thesis are an architectural model for groupware applications, the Clover architecture model, and a generic platform, the Clover platform, for the development of groupware applications centered on group activity. The Clover model, which defines three classes of services that a groupware application may support (production, communication and coordination), serves as a conceptual guide to design our architectural model and our platform. Most of the work done focuses on technical solutions such as merging and versioning. Our approach is complementary since we focus on group activity. Our requirements are to offer functionnalities of a high level of abstraction centered on group activity (vs technical solutions) and covering the three functional spaces defined by the Clover model. The study of a set of existing development tools shows that most of them focus only on coordination.We show the limit of the latter approach based on coordination. The CoVitesse system, a groupware application that enables collaborative navigation on the WWW, and a shared whiteboard have been developed with this platform as illustrative examples.

Keywords Human-Computer Interaction, groupware, CSCW, software architectural model, platform of development.