Un système coopératif basé sur les transactions

Manuel Munier, Khalid Benali, Claude Godart

Catégorie : Chercheur

Un système coopératif basé sur les transactions

Manuel Munier[1,2], Khalid Benali[1] , Claude Godart[1]

[1] **LORIA** - INRIA - UMR CNRS
Campus Scientifique BP 239,
54506 Vandœuvre-lès-Nancy Cedex
France

[2] **LIUPPA**
BP 1155,
64013 Pau Cedex
France

E-mail: munier@marsan.univ-pau.fr, benali@loria.fr, godart@loria.fr,
Fax : 03 83 41 30 79
Tél : 03 83 59 30 98

RÉSUMÉ : La conception ou la réalisation de tout projet un tant soit peu conséquent sous-entend l'implication d'un certain nombre de personnes, voire d'un certain nombre d'équipes ou d'entreprises. Outre le fait qu'ils travaillent tous sur le même projet, les différents acteurs coopèrent et collaborent. En effet la réalisation de tout projet n'est pas faite en une succession d'étapes menées par un seul acteur mais par une synergie d'acteurs coopérant à la réalisation d'un but commun. C'est le concept même d'ingénierie concourante. Celle-ci nécessite une coopération entre les différents acteurs et un échange des données produite par chacun d'entre eux. De plus en plus d'entreprises utilisant internet pour l'échange de données, les différents acteurs ne sont alors plus soumis à l'obligation de travailler dans un même lieu géographique. On parle alors d'entreprise virtuelle ou d'entreprise-projet si le travail en commun dure le temps d'un projet. Cependant, il ne suffit pas simplement d'échanger des données pour travailler ensemble, il faut aussi contrôler et gérer ces échanges. La collaboration impliquant une certaine concourance dans le travail de plusieurs acteurs, la production de différentes versions des documents échangés implique un contrôle des échanges. L'objectif de cet article est la description d'un système de coopération réellement distribué. La philosophie globale de notre système est la distribution du contrôle des échanges et l'accès aux données échangées de manière standard. Adoptant une approche transactionnelle pour la réalisation de notre système de coopération, nous avons formalisés notre système transactionnel coopératif et notre critère de correction distribué (DisCOO-sérialisabilité). Dans cet article, nous nous servons d'un exemple support pour illustrer notre propos et appliquer cette formalisation à un cas concret. Nous présentons le prototype DisCOO (réalisé en Java et Prolog au-dessus d'un ORB) mettant en oeuvre notre système.

MOTS-CLÉS : Systèmes coopératifs, modèles de transactions avancés, systèmes distribués.

1 Introduction

La conception ou la réalisation de tout projet un tant soit peu conséquent sous-entend l'implication d'un certain nombre de personnes, voire d'un certain nombre d'équipes ou d'entreprises. En effet les compétences nécessaires sont diverses et variées. Si nous prenons le domaine des Bâtiments et Travaux Publics qui va nous fournir l'exemple support de présentation de notre approche, les compétences pour réaliser une simple maison vont de la conception des volumes par l'architecte à la spécification des éléments structurels par l'ingénieur-structure en passant par la climatisation faite par le thermicien. Dire que ces différents corps de métier sont amenés à travailler sur le même projet ne suffit pas, car outre le fait qu'ils travaillent tous, les différents acteurs coopèrent et collaborent. En effet la réalisation d'un bâtiment ou de tout projet n'est pas faite en une succession d'étapes menées par un seul acteur mais par une synergie d'acteurs coopérant à la réalisation d'un but commun. C'est le concept même d'ingénierie concourante qui permet un travail en synergie des différents acteurs autorisant une réduction des délais de production et une meilleure prise en compte, tout au long du projet, des spécificités et des compétences de chacun des acteurs. Cette ingénierie concourante nécessite une coopération entre les différents acteurs et un échange des données produite par chacun d'entre eux. L'échange de don-nées existe depuis longtemps sous des formes simples (courrier, fax, échange de disquettes,...) dans les entreprises. Dans le contexte actuel, avec la démocratisation d'internet, de plus en plus d'entreprises utilisent ce medium pour l'échange de données. Les différents acteurs ne sont alors plus soumis à l'obligation de travailler dans un même lieu géographique. On parle alors d'entreprise virtuelle ou d'entreprise-projet si le travail en commun dure le temps d'un projet (par exemple un chantier en BTP). Cependant, cet échange non contrôlé ne permet pas une réelle synergie entre les différents acteurs. Il ne suffit pas simplement d'échanger des données pour travailler ensemble, il faut aussi contrôler et gérer ces échanges. Pour revenir à notre exemple support, il ne suffit pas simplement d'envoyer une version du plan de l'architecte au thermicien pour résoudre les problèmes liés à leur collaboration. En effet le plan de l'architecte envoyé ne correspond qu'à une version à un moment donné. La collaboration impliquant une certaine concourance dans le travail des deux acteurs, la production de différentes versions des documents échangés implique un contrôle des échanges. Ce contrôle des échanges est actuellement possible dans les environnements d'aide à la coopération, mais avec un contrôle en général centralisé. Dans le BTP, des systèmes tels que les armoires à plans (informatisés et centralisés) répondent partiellement au contrôle des échanges, en centralisant ce contrôle et en gérant des droits spécifiques sur cette armoire à plan centrale. L'objectif de cet article est la description d'un système de coopération réellement distribué. Nous commençons par présenter, dans le paragraphe 1, les différents types d'environnement d'aide à la coopération. Nous présentons ensuite, dans le paragraphe 2, l'exemple support qui va nous permettre d'expliciter notre démarche. Les paragraphes 3 et 4 montrent la philosophie globale de notre système, à savoir, la distribution du contrôle des échanges et l'accès aux données échangées de manière standard. Le paragraphe 5 explicite notre choix d'une approche transactionnelle pour la réalisation de notre système de coopération. Le paragraphe 6 nous permet, quant à lui de présenter la mise en oeuvre de notre système grâce au prototype *DisCOO* réalisé en Java et Prolog au-dessus d'un ORB. Nous concluons enfin en présentant les résultats de notre approche et les perspectives possibles d'une telle approche.

2 Environnements d'Aide à la Coopération Existants

Dans le cas d'une application relativement complexe, il n'existe généralement pas d'acteur qui possède à lui seul la maîtrise et la connaissance intégrale de l'activité globale exécutée. Il est donc extrêmement difficile pour un acteur quelconque d'appréhender, "manuellement", toutes les conséquences d'une modification apportée à un document de l'application. Ce sont les raisons pour lesquelles il est indispensable d'utiliser des environnements mettant en oeuvre des mécanismes de coordination et de communication suffisamment sophistiqués, permettant ainsi de notifier et de propager les changements aux acteurs qui sont concernés et de s'assurer que les efforts de chacun des acteurs du projet sont coordonnés de manière à réduire l'impact d'une modification d'un document sur l'activité globale. Ces environnements peuvent être classés en quatre catégories selon leur approche de la coopération. Nous avons tout d'abord les **gestionnaires de configurations** dont l'objectif est de gérer les versions et les configurations successives des différentes don-nées partagées (cohérence des données). Viennent ensuite les **environnements centrés procédés** qui permettent de contrôler les états successifs des données partagées et/ou l'enchaînement des différentes activités. Dans le domaine des bases de données, les **systèmes transactionnels** garantissent que l'exécution en parallèle de plusieurs activités (encapsulées dans des transactions) n'introduit pas d'inconsistance au niveau des résultats produits par ces activités. La dernière catégorie, les **outils d'aide au travail coopérati**f, est plus orientée vers les aspects communication et relations humaines de la coopération.

Gestionnaires de Configurations : Afin de contrôler les mises à jour concurrentes effectuées par les différentes activités d'un système distribué, il est possible d'utiliser un outil de gestion de configurations (RCS [TIC 89], ClearCase [Atr 94], Continuus, Adèle [BEL 94]). Son rôle est d'assurer le stockage des données partagées, appelées ressources, tout en gardant une trace de leur évolution (généralement sous la forme de leurs versions successives) et en contrôlant les accès concurrents effectués par les activités. Par exemple, si deux activités modifient en parallèle une même ressource, elles vont chacune développer, à partir d'une version initiale de cette ressource, ce que l'on appelle une branche de versions. De cette façon, chacune des activités travaille sur sa propre copie de la ressource, sans être perturbée par les modifications effectuées par l'autre activité. Lorsqu'elles auront terminé leur travail, une activité (éventuellement différente) sera chargée de fusionner ces deux branches afin de ne produire qu'une seule nouvelle version, i.e. que les modifications d'une des activités n'écraseront pas les modifications de l'autre. Le rôle du gestionnaire de configurations sera alors de mémoriser le fait que cette nouvelle version est dérivée des deux précédentes. Toutefois, la plupart des gestionnaires de configurations reposent sur une architecture client/serveur (référentiel centralisé, éventuellement répliqué et/ou partitionné sur plusieurs serveurs). Ils ne conviennent donc pas à nos exigences de distribution et d'autonomie des activités. En outre, les gestionnaires de configurations sont essentiellement concernés par les problèmes de concurrence d'accès à un référentiel: gestion des versions et des configurations de ressources partagées. Ils ne définissent aucun contrôle de la coopération sur les échanges entre activités.

Environnements Centrés Procédés : A la différence des gestionnaires de configurations, les outils de gestion de pro-cédés sont principalement orientés vers la description des exécutions correctes en termes d'états successifs d'une ressource ou d'enchaînement des différentes activités: modèles à flots de tâches ou modèles de workflow [WFM 97, ALO 96] (modèle des contrats [WAC 92]), règles événement/condition/action [BAR 92a] (Adèle-Tempo [BEL 94], MARVEL [BAR 92a, BAR 92b]). Cette approche nécessite généralement de décrire l'application complète, i.e. en tenant compte de toutes les activités et de toutes les ressources. Si l'on intègre en plus les problèmes liés à la synchronisation des activités distribuées, le

modèle obtenu devient alors rapidement complexe du fait de la complexité inhérente du contexte à modéliser. Contrairement aux outils de gestion de procédés existants, notre objectif n'est pas de décrire comment les activités doivent travailler pour pouvoir coopérer, mais simplement de définir de quelle façon doivent se dérouler les échanges entre ces activités. Nous voulons imposer des contrôles sur les échanges de résultats entre activités, mais pas sur les activités elles-même ni sur la manière dont elles produisent ces résultats.

Outils d'Aide au Travail Coopératif *:* Le travail de groupe assisté par ordinateur *("Computer Supported Cooperative Work*") a pour objectif de permettre à des groupes d'usagers de collaborer à des buts communs au moyen d'un système informatique, appelé système collaboratif ou collecticiel. Toutefois, contrairement aux gestionnaires de configurations, aux outils de gestion de procédés ou aux systèmes transactionnels, un environnement CSCW n'est pas uniquement orienté vers le maintien de la cohérence des objets partagés (gestion des accès concurrents). Un tel environnement prend également en compte des aspects plus "humains" de la coopération tels que la gestion d'un groupe de personnes, les mécanismes de notification, les techniques de communication (messagerie électronique, vidéo-conférences, ...). BSCW [BEN 97a, BEN 97b] (partage d'informations au travers d'un référentiel centralisé), Wiki (rédaction collective de documents via le Web, cf wiki.lri.fr:8080/scoop/scoop.wiki) et Microsoft NetMeeting (vidéo/audio conférence, partage d'applications, tableau blanc, forums de discussion synchrone, cf www.microsoft.com/netmeeting/features) sont des exemples de tels collecticiels. En ce qui concerne le contrôle de la cohérence, les collecticiels reposent sur les mécanismes développés dans les systèmes distribués ou les gestionnaires de configurations: verrouillage des objets, passage de jeton (ou "prise de tour"), détection des dépendances (conflits résolus par les utilisateurs). A la différence des systèmes transactionnels, ces techniques ont pour objectif d'assurer la cohérence des objets partagés et non de coordonner les activités qui coopèrent.

Systèmes Transactionnels : Dans la cas d'un système transactionnel [AGR 90, BER 97], chaque activité est encapsulée dans une transaction dont l'exécution représente la séquence des opérations (lectures et écritures par exemple) invoquées par cette activité sur les objets partagés. Une transaction constitue l'unité d'exécution élémentaire: soit la transaction se termine totalement et elle a les effets désirés sur les objets auxquels elle a accédé (la transaction est dite "validée"), soit elle est interrompue et elle n'a aucun effet (la transaction est dite "annulée"). Cette règle du "tout ou rien" (encore appelée propriété d'atomicité des transactions) permet d'assurer l'atomicité aux défaillances des transactions. L'idée de base d'un système transactionnel est de garantir que si chaque transaction, prise individuellement, s'exécute correctement, alors leur exécution entremêlée (due à leurs accès concurrents aux objets partagés) devra être "correcte". Ceci est assuré par un critère de correction défini comme étant un ensemble de propriétés caractérisant l'histoire des exécutions considérées comme correctes. L'histoire de l'exécution entremêlée de plusieurs transactions est représentée par la séquence des opérations (lecture, écriture,...) invoquées, concurremment, sur les objets partagés. Le critère de correction le plus répandu dans le domaine des applications traditionnelles (administration, banque, ...) est la "sérialisabilité". Celui-ci considère que l'exécution entremêlée de plusieurs transactions est correcte si elle produit un résultat équivalent à une exécution en série de ces transactions (l'exécution est dite sérialisable). Ce critère est cependant trop strict puisqu'il garantit l'isolation des transactions (atomicité à la concurrence): les états intermédiaires d'une transaction, en termes de valeurs des objets qu'elle manipule, ne sont pas visibles par d'autres transactions. La sérialisabilité ne supporte donc pas la coopération telle que nous l'avons définie, à savoir la possibilité offerte aux transactions de s'échanger des résultats intermédiaires au cours de leur exécution. De nouveaux modèles de transactions et critères de correction ont toutefois été définis pour

relâcher l'isolation entre les transactions: transactions emboîtées [MOS 81], transactions multi-niveaux [BEE 88], sagas [GAR 87],…

3 Présentation de l'Exemple Support

L'exemple sur lequel nous allons nous baser est celui développé dans [BIG 98, BEN 98b, BEN 99] (lui-même inspiré de celui présenté dans [ROS 96]). L'application considérée a pour objectif la conception d'un appartement sur un niveau ayant une salle de séjour et dont l'un des côtés est constitué intégralement d'une baie vitrée. Plusieurs partenaires interviennent lors de cette conception, formant ainsi une entreprise-projet. Ceux-ci travaillent sur trois documents qu'ils sont amenés à s'échanger: le **plan** rédigé conjointement par l'architecte et l'ingénieur-structure, l**'avis** de l'urbaniste et, dans une moindre mesure sur cet exemple, les spécifications de la **baie** vitrée définies par le thermicien. Dans cet exemple, pour des raisons de simplification, nous représenterons chaque partenaire par une activité (figure 2.1).

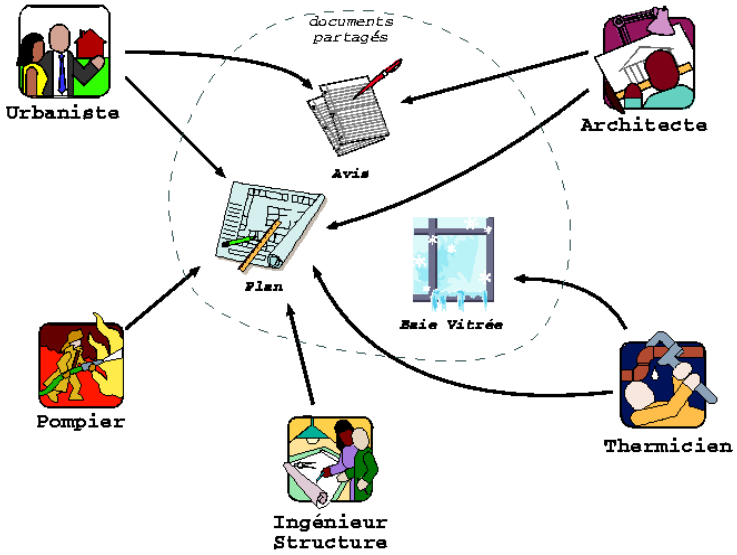

Figure 2.1 *Partenaires et documents partagés*

— l'**architecte** : Il s'occupe de la conception du bâtiment en termes de disposition des murs (disposition et taille des pièces), d'emplacements de fenêtres (luminosité), ...,c'est-à-dire de dessiner le plan du bâtiment en ne considérant que les aspects volume, espace et luminosité.

— l'**ingénieur-structure** : Son activité consiste à spécifier les éléments structurels de l'appartement et à garantir la stabilité de la construction. De tels éléments (murs porteurs, poutres ou colonnes de soutènement, ...) seront choisis de manière à respecter le plus possible l'harmonie et les choix de l'architecte.

— le **thermicien** : Il est en charge de la climatisation du bâtiment (chauffage, isolation thermique, ...). Nous limitons son intervention au choix du type de baie vitrée (matériaux, épaisseur) en fonction de l'exposition et de la surface vitrée.

— l'**urbaniste** : Il contrôle le plan produit par l'architecte et émet un avis sur son travail en fonction de critères d'intégration urbaine du bâtiment. L'architecte doit alors tenir compte de cet avis pour rédiger un nouveau plan acceptable, et accepté, par l'urbaniste.

— le **pompier** : Avant d'émettre son avis, l'urbaniste peut demander conseil à un pompier pour vérifier que l'appartement est conforme aux normes incendie en vigueur (emplacement des issues de secours, des trappes de désenfumage, ...).

Les différents échanges de données entre les partenaires ne sont toutefois pas soumis aux mêmes règles. S'il est souhaitable, dans certains cas, d'encourager une coopération maximale entre les partenaires, dans d'autres il peut être nécessaire de fixer certaines contraintes. C'est le cas par exemple de l'urbaniste et de l'architecte: ils se partagent certains documents, mais seulement en lecture, chacun ne pouvant modifier les documents de l'autre. Les règles de coopération négociées pour contrôler ces échanges de documents entre les différents partenaires sont représentées sur la figure 2.2.

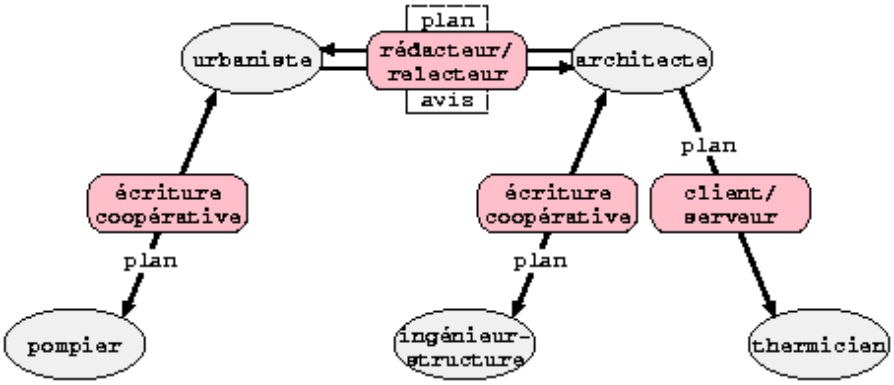

Figure 2.2 *Schémas de Coopération Négociés pour les Echanges de Documents*

— client/serveur : L'architecte (le "serveur") fournit différentes versions successives du plan au thermicien (le "client"). Les échanges se font donc uniquement de l'architecte vers le thermicien.

— rédacteur/relecteur : L'urbaniste (le "relecteur") lit, mais ne modifie pas, le plan qui lui est fourni par l'architecte (le "rédacteur"). Il rédige alors son avis qu'il transmet à l'architecte qui le lit, mais ne le modifie pas, pour mettre à jour son plan, et ainsi de suite.

— écriture coopérative : L'architecte et l'ingénieur-structure travaillent tous les deux à la rédaction du plan (même objet logique). Durant toute la durée de l'activité de conception ils le modifient, éventuellement simultanément, intègrent les modifications effectuées par l'un ou l'autre, dans le but de fournir au final une version de ce plan qui les satisfait tous les deux.

Notre objectif est donc de concevoir une infrastructure pour **coordonne**r, via l'utilisation de différents **schémas de coopératio**n, les **échanges de données** entre différentes activités. Il nous faut pour cela pouvoir stocker des données au niveau d'une activité, définir des protocoles d'échange, puis les appliquer pour contrôler les interactions entre activités.

4 Contrôle des Echanges

Lorsqu'une activité veut partager un objet donné avec une autre activité, il y a tout d'abord une phase de négociation pour définir le schéma de coopération qui contrôlera les échanges concernant cet objet entre ces deux activités. Il s'agit, au minimum, de s'assurer que le schéma que l'une des activités désire utiliser est connu de l'autre activité. Le résultat de cette négociation est un **contrat** passé entre les deux activités et fixant les règles de coopération à respecter pour le partage de l'objet concerné. Par exemple *Contract[archi,therm, {plan},client_server]* représente le contrat passé entre l'architecte et le thermicien pour partager l'objet `plan` selon le mode de coopération "client/serveur". Nous obtenons ainsi pour chaque activité du système une **table de coopération** contenant tous les contrats signés par cette activité.

Activité	partenaire	objets	schéma	Rôle

thermicien	architecte	{plan}	client/serveur	Client
Activité	**partenaire**	**objets**	**schéma**	**Rôle**
architecte	inge.-structure	{plan}	écriture coopérative	~
architecte	thermicien	{plan}	client/serveur	Serveur
architecte	urbaniste	{{plan},{avis}}	rédacteur/relecteur	~

Figure 3.1 *Tables de Coopération pour la Figure 2.2*

Lors d'un échange de données entre deux activités, ce qui constitue une opération de transfert, chacune de ces deux activités contrôlera localement (i.e. à partir des informations contenues dans sa table de coopération) que cet échange est correct par rapport au contrat qu'elles auront négocié toutes les deux. Si l'une ou l'autre détecte une violation du contrat (ou plus exactement du schéma de coopération "figurant" sur le contrat) cette opération d'échange sera refusée. Une activité comporte donc deux composants dédiés au contrôle de ses échanges avec les autres activités: un **protocole** chargé de la gestion de la table de coopération de l'activité ainsi que de l'évaluation des schémas de coopération figurant dans cette table; un **coordinateur** pour contrôler que tous les accès réalisés sur l'espace de coopération respectent le protocole.

5 Accès aux Données

Le référentiel local d'une activité est composé de deux parties: une partie publique, nommée **espace de coopération**, et une partie privée, nommée **espace de travail**. L'espace de coopération contient les versions des objets rendues publiques par l'activité, c'est-à-dire les versions pouvant être importées par d'autres activités, ainsi que les relations entre ces différentes versions le cas échéant. Les échanges de données entre activités seront réalisés entre leurs espaces de coopération respectifs. L'espace de travail permet quant à lui de présenter à l'utilisateur les objets de l'espace de coopération sous une forme utilisable par ses applications existantes (fichiers `.DXF` pour Autocad ou `.DOC` pour Word par exemple). C'est l'endroit où les applications vont effectivement manipuler les données. Comme cela est représenté figure 4.1, l'utilisateur devra tout d'abord importer depuis une autre activité le document (sous forme d'objet avec ses "informations de contrôle") qu'il désire utiliser. Ce document sera alors stocké dans son espace de coopération. Afin de pouvoir le manipuler (sous forme de fichier par exemple) avec ses applications courantes, il devra ensuite transférer (opération `Check_Out`) ce document dans son espace de travail. Les modifications qu'il effectuera alors sur ce document ne seront pas visibles aux autres activités (l'espace de travail est une zone privée). Quand il jugera son travail terminé, il publiera (opération `Check_In`) la nouvelle version de ce document, ce qui aura pour effet de mettre à jour l'objet document stocké dans son espace de coopération. A partir de cet instant, ce document (sous forme d'objet) pourra être à son tour importé par d'autres activités. Le référentiel local d'une activité est ainsi représenté par deux composants distincts: une zone d'échange, l'espace de coopération, accessible aux autres activités et dans laquelle sont stockés les objets partagés; une zone privée, l'espace de travail, sur les données de laquelle l'activité peut travailler (appels d'outils existants) pour accomplir sa tâche. Les transferts entre ces deux zones sont réalisés à l'initiative de l'activité (et donc de l'utilisateur). En d'autres termes, c'est l'utilisateur qui décide du moment où il publie ses résultats (intermédiaires ou finaux) ainsi que du moment où il intègre, au niveau de son espace de travail, les modifications effectuées sur les objets partagés par les autres activités (préalablement importées dans son espace de coopération).

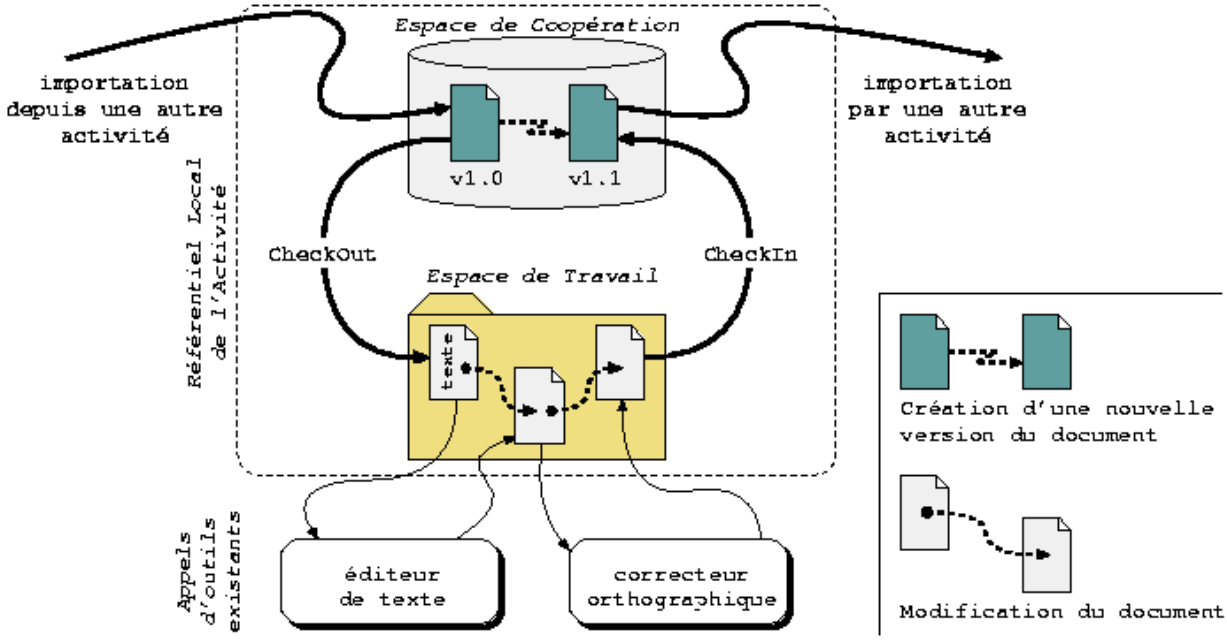

Figure 4.1 Espace de Coopération & Espace de Travail

Notre système et notre approche ayant pour objectif de permettre la coopération et le travail collaboratif sans changer les outils et les modes de production individuels, nous ne nous intéressons et ne gérons que les données de l'espace de coopération. L'espace de travail ne nous intéresse que comme destination d'un transfert depuis l'espace de coopération ou comme source d'une publication vers celui-ci. Du point de vue des acteurs du système, c'est-à-dire les utilisateurs, il est nécessaire qu'ils puissent utiliser leurs applications courantes (Autocad, Word, emacs, gcc, ...) sur les données partagées. Ces données doivent donc être accessibles dans leur format natif, i.e. des fichiers `.DXF` ou `.DOC`, voire même des tuples dans une base de données ou bien des objets dans un environnement orienté objet (ex: CORBA).

6 Contrôle des échanges : une *approche transactionnelle*

Dans le cas des applications coopératives distribuées et hétérogènes, la programmation explicite des interactions est généralement difficile à maîtriser puisqu'elle nécessite de prévoir toutes les interactions possibles (problème combinatoire). En outre, cette approche "programmation concurrente" est également source d'erreurs (interactions non prévues ou incompatibles, verrous mortels,...). A l'inverse, une approche "contrôle de la concurrence d'accès" a justement pour objectif de masquer cette complexité et de décharger au maximum les programmeurs d'applications des problèmes liés aux interactions entre activités concurrentes: un protocole de contrôle de la concurrence assure que le fonctionnement global d'un ensemble d'activités concurrentes est correct, à supposer bien entendu que chaque activité soit individuellement correcte. Nous avons donc choisi une **approche transactionnelle** plutôt qu'une approche basée sur les gestionnaires de configurations, les environnements centrés procédés ou les outils de CSCW. Le contrôle de la concurrence est ainsi réalisé par un critère de correction qui définit les exécutions (concurrentes) considérées comme étant correctes. Une application coopérative est donc vue comme un ensemble de transactions qui accèdent "en même temps" à un ensemble d'objets. Chaque transaction encapsule une activité (supposée correcte) de l'application afin de lui cacher les problèmes

liés à la concurrence d'accès. Une transaction peut ainsi être représentée par la séquence des opérations invoquées sur les objets manipulés. Mais le problème de la synchronisation de ces transactions est plus complexe que celui de la synchronisation des transactions dans des contextes applicatifs traditionnels (administration, banque, ...) [GRA 93] où les activités sont essentiellement concurrentes, i.e. pouvant s'exécuter de manière isolée en s'ignorant complètement l'une de l'autre. Dans notre cas, il serait donc plus exact de parler d'une approche "contrôle de la coopération" que d'une approche "contrôle de la concurrence". Nous aborderons donc le problème de la coopération dans les applications coopératives distribuées et hétérogènes sous l'angle des **systèmes transactionnels coopératif**s.

De nouveaux modèles de transactions dits "étendus" ont été développés afin de relâcher ou d'assouplir une ou plusieurs des propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité) des transactions classiques, d'organiser les transactions selon les spécificités de l'application (parallélisme local, décomposition d'une transaction en sous-transactions, ...) ou de permettre l'utilisation de critères de corrections moins contraignants que la sérialisabilité. Si les différents modèles de transactions et critères de correction existants permettent effectivement de prendre en compte certains aspects de la coopération, aucun ne répond simultanément à nos besoins de **coopération** (les transactions ne doivent pas être isolées les unes des autres), de **distribution** (chaque transaction a sa propre copie des objets qu'elle manipule) et d**'autonomie** (pas de site central; le contrôle des interactions est réalisé localement par chaque transaction). Ce sont les raisons pour lesquelles nous avons décidé de définir un nouveau modèle de transactions étendu qui satisfasse ces différents besoins de la coopération [MUN 98, BEN 98a, MUN 99a, MUN 99 b]. Comme nous l'avons vu dans la section 1, les environnements d'aide à la coopération actuels ne sont pas adaptés aux applications distribuées. La plupart d'entre eux sont basés sur une architecture client/serveur dans laquelle, bien que les différentes activités puissent s'exécuter sur des sites géographiquement distribués, le contrôle de la coopération (cohérence des données partagées, enchaînement des activités,...) reste centralisé sur un référentiel commun (éventuellement répliqué et/ou partitionné sur plusieurs serveurs). La centralisation facilite la synchronisation et le contrôle de l'activité globale, mais son principal inconvénient est la nécessité, pour les activités clientes, d'être connectées en permanence à ce serveur, ou du moins de devoir s'y connecter pour échanger des données avec une autre activité. Un second inconvénient de ces environnements d'aide à la coopération concerne les techniques mises en oeuvre pour coordonner les différentes activités. Les gestionnaires de versions et de configurations se limitent à assurer le stockage des données partagées tout en gardant une trace de leur évolution (sous la forme d'un graphe des versions successives) et en contrôlant les accès concurrents effectués par les activités. De leur côté, les environnements centrés procédés nécessitent généralement de décrire toute l'application, c'est-à-dire de prévoir toutes les interactions possibles entre toutes les activités, une tâche qui peut rapidement devenir très complexe. Quant aux systèmes transactionnels, ils permettent de garantir que si chaque activité (encapsulée dans une transaction), prise individuellement, s'exécute correctement, alors leur exécution entremêlée (due à leurs accès concurrents aux objets partagés) sera conforme à un ensemble de propriétés appelé critère de correction. Les critères de correction existants sont toutefois mieux adaptés aux transactions de courte durée et plutôt isolées les unes des autres qu'aux activités coopératives distribuées qui nous intéressent. Partant des travaux réalisés dans *COO* [MOL 96] en ce qui concerne la correction syntaxique des interactions coopératives, la première étape de notre travail consistait donc à passer d'un contrôle des accès concurrents (interactions implicites) à un contrôle des échanges de données entre transactions (interactions explicites). Nous avons pour cela défini les notions de **référentiel local** d'une transaction, d**'histoire locale** d'une transaction, et d**'opération de**

transfert entre transactions. L'idée est que chaque transaction possède sa propre copie des objets auxquels elle accède et coopère avec les autres transactions en échangeant des valeurs de leurs copies respectives. Ce sont ces opérations de transfert qui nous permettent de synchroniser les histoires locales des différentes transactions. La deuxième étape concernait la définition de nouveaux **critères de correction** qui soient eux-mêmes **distribués**. L'idée était en effet de permettre à chaque transaction du système de coordonner elle-même ses propres échanges de données avec les autres transactions. A l'inverse d'un critère de correction "classique" qui est défini sur l'histoire globale, un critère de correction dit "distribué" est destiné à être vérifié par chaque nœud du système (un nœud représentant l'exécution d'une transaction) en n'ayant accès qu'aux informations journalisées localement par ce nœud (l'histoire locale de la transaction). Nous avons ainsi défini un nouveau critère de correction local (la ***DisCOO*-sérialisabilité**) qui, lorsqu'il est assuré au niveau de chaque transaction, garantit les mêmes propriétés au niveau du système complet que son homologue "global" classique (la *COO*-sérialisabilité). Les transactions de notre modèle sont ainsi organisées en réseau, les nœuds réprésentant les transactions et les arcs représentant les schémas de coopération négociés entre les transactions. Il s'agit d'une architecture d'égal à égal dans laquelle chaque transaction est elle-même responsable du contrôle de ses propres interactions avec les autres transactions du système. L'objet de cet article n'étant pas la description complète de notre modèle transactionnel, nous suggérons au lecteur intéressé la lecture de [MUN 99b] synthétisant nos travaux dans ce domaine et décrivant formellement notre nouveau modèle transactionnel.

7 Mise en œuvre

Afin de simplifier le développement d'applications distribuées, un certain nombre de plates-formes logicielles ont été proposées et évitent ainsi aux programmeurs de de-voir accéder directement et manuellement aux couches basses du réseau. Ces plates-formes peuvent être classées en quatre catégories: appels de procédures à distance (RPC (*Remote Procedure Call)*, Java RMI (*Remote Method Invocation)*), mémoires virtuelles partagées (PVM *(Parallel Virtual Machine)*, PerDiS [SHA 97]), systèmes orientés vers la persistance des objets (Shore, Arjuna), systèmes répartis à objets (modèle CORBA *(Common Object Request Broker Architecture)* [OMG 95, WEI 96] de l'OMG (Object Management Group), architecture ActiveX/DCOM de Microsoft). CORBA constitue la plate-forme logicielle idéale pour nous permettre de mettre en œuvre les travaux présentés dans cet article. Les différents composants de notre architecture seront regroupés au sein d'un **service de coopération** faisant partie de la catégorie des **domaines d'application CORBA** *(CORBA domain*s). Il s'agit donc réellement de définir un service fournissant les mécanismes de base nécessaires au développement d'applications coopératives distribuées (notre domaine). Ainsi, les interactions entre les différentes activités de notre système, ou plus précisément entre leurs coordinateurs respectifs, seront réalisées au travers d'un ORB. Le fait que ces activités soient distribuées leur sera complètement transparent. Pour définir notre architecture, nous avons identifié quatre services essentiels pour la coopération: un service d**'espace de coopération** pour gérer les ressources de la base de données locale d'une activité; un service d**'espace de travail** qui permet à l'utilisateur de manipuler les ressources de son espace de coopération à l'aide de ses applications habituelles (Word, Autocad, emacs, gcc, ...); un service de **coordination** qui garantit que toutes les requêtes transmises au service d'espace de coopération sont validées par le service de protocole (cf. couche "contrôle des interactions"); un service de **protocole** chargé, pour une activité donnée, du contrôle effectif des interactions de cette activités avec les autres activités du système (contrats négociés, histoire locale de l'activité, vérification des schémas de coopération par rapport à l'histoire locale courante) (cf. figure 6.1). Nous avons

développés ces services en tant que services CORBA de manière à ce qu'ils puissent être à leur tour utilisés par d'autres services ou objets CORBA, au même titre que les services CORBA standard.

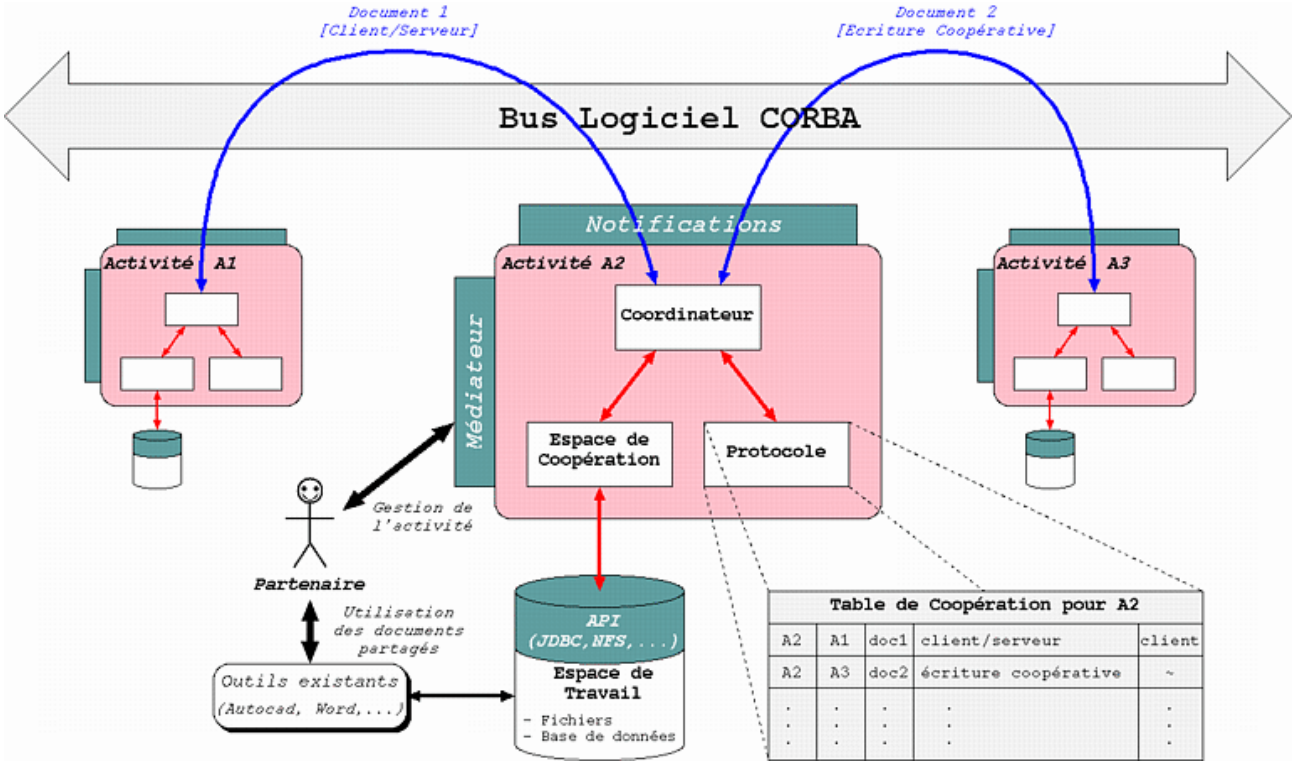

Figure 6.1 *Mise en œuvre de notre Architecture au-dessus d'un ORB*

En fait, tous les composants de notre architecture sont eux-même des objets CORBA. La mise en oeuvre du modèle de transactions que nous avons présenté dans la première partie de cet article concerne donc essentiellement l'implémentation des services de coordination et de protocole. Au niveau du prototype *DisCOO*, l'implémentation des services d'espace de coopération et d'espace de travail a été simplifiée, notre objectif étant de fournir les fonctionnalités minimales pour développer un exemple d'application coopérative distribuée démontrant le bon fonctionnement des services de coordination et de protocole. Ainsi, dans un premier temps, un espace de coopération ne gère qu'une liste des versions successives d'une ressource donnée (et non un graphe) et les seuls ressources pouvant être partagées sont des objets de type fichier externalisables vers un espace de travail matérialisé par un répertoire sur le disque dur de la machine. L'utilisateur peut ainsi manipuler ses ressources (sous forme de fichiers) à l'aide de ses applications habituelles.

Service de Coordination : Le principal composant de ce service est le coordinateur. Son rôle au sein de notre architecture est de "filtrer" les échanges d'une activités avec les autres activités du système, c'est-à-dire de refuser tout échange qui ne vérifierait pas le(s) schéma(s) de coopération négocié(s). Sinon la requête est transmise à l'espace de coopération de l'activité. En ce qui concerne l'opération de négociation, l'implémentation du coordinateur impose que tout contrat négocié est journalisé simultanément par les deux partenaires et qu'il n'est pas possible de négocier un contrat concernant un objet figurant déjà sur un autre contrat. Cela signifie en particulier que cette implémentation du coordinateur n'autorise pas les activités à re-négocier un schéma de coopération.

Service de Protocole : Le cœur du prototype *DisCOO* est le service de protocole, et plus particulièrement les composants chargés de la gestion de l'histoire locale d'une activité et de la vérification d'un schéma de coopération donné par rapport à cette histoire locale. Au niveau formel, nous avons définis un schéma de coopération comme étant un ensemble de règles de coopération. Ces règles sont elles-même des prédicats destinés à être évalués par rapport à l'histoire locale courante de l'activité. Par exemple, le schéma de coopération "écriture coopérative" est représenté par un prédicat.

Plutôt que de reprogrammer des algorithmes représentant ces différentes règles de coopération **nous avons choisi de les implémenter directement sous la forme de prédicats Prolog**. En procédant de cette manière, les schémas et règles de coopération (formalisés en ACTA) sont réellement implémentés sous la forme de propriétés (prédicats Prolog) évaluées sur les histoires locales des activités. Du point de vue de la programmation cela nous permet de créer et de modifier les schémas de coopération de façon "naturelle" par rapport à leur définition en ACTA. L'ajout de nouveaux schémas de coopération au prototype *DisCOO* en est ainsi grandement facilitée. Pour conclure la présentation du prototype *DisCOO* signalons que tous les composants ont été programmés en Java au-dessus d'un ORB lui-même développé en Java (JacORB). Quant à l'interpréteur Prolog, il est lui aussi développé en Java. Le fait d'utiliser le langage Java nous permet ainsi de déployer *DisCOO* dans tout environnement (Windows 98/NT, Unix, MacOS, ...) pour lequel une machine virtuelle Java est disponible. Le prototype *DisCOO* est donc complètement découplé de l'environnement d'exécution, tant du point de vue du système d'exploitation (bytecode Java) que de l'infrastructure de communication (CORBA). A titre d'exemple nous avons testé *DisCOO* sur un réseau local hétérogène composé à la fois de PC fonctionnant sous Windows NT, de stations de travail Solaris et de PC sous Linux.

8 Conclusion

Les travaux décrits dans cet article et détaillés dans [MUN 99a, MUN 99b] ont donné lieu à deux résultats. D'un point de vue formel tout d'abord, il s'agit de la définition d'un nouveau **modèle de transactions avanc**é, les *DisCOO*-transactions, ainsi que d'un **critère de correction distribué**, l*a DisCOO*-sérialisabilité (avec trois schémas de co-opération: "écriture coopérative", "client/serveur" et "rédacteur/relecteur"). Le second résultat est la définition d'une **architecture** nous permettant de construire de telles applications en "connectant", une fois qu'un schéma de coopération aura été négocié, les différentes activités distribuées. Cette architecture a ensuite été mise en oeuvre sous forme de **services de base pour la coopération** au sein de notre prototype *DisCOO*. Notre objectif était de ne pas développer "un système propriétaire supplémentaire". L'idée était plutôt de réaliser une infrastructure intégrant les mécanismes de base nécessaires à la coopération et à partir de laquelle il serait possible de programmer de telles applications coopératives distribuées. Lors de la présentation de l'architecture retenue pour notre système de coopération (figure 6.1), nous avons indiqué qu'une activité était constituée de quatre composants principaux, définissant ainsi les quatre **services de coopération de base**: un espace de coopération pour stocker les objets partagés, un espace de travail pour manipuler ces objets à l'aide des outils existants, un protocole pour contrôler les échanges de l'activité avec ses partenaires, et enfin un coordinateur qui joue le rôle d'interface de l'activité. Lorsque nous avons présenté le contrôle des échanges entre activités (section 5), nous ne nous sommes en fait intéressés qu'au résultat de la négociation d'un schéma de coopération entre deux transactions, c'est-à-dire à l'événement *Contract[*t_i*,* t_j*, O, S]* qui est journalisé dans l'histoire locale de chaque transaction. Une de extensions à ces travaux a consisté à considérer les opérations de négociation et de re-

négociation comme étant des opérations classiques (au même titre que les opérations read et write par exemple) pouvant être invoquées sur des "objets de négociation" qui seraient porteurs d'informations spécifiques: liste des schémas proposés par chaque partenaire, liste des schémas convenant aux deux partenaires, proposition d'un autre schéma en cas de refus de celui demandé par le partenaire,… Cette approche a été détaillée et une description de cette approche de la négociation peut être trouvée dans [MUN 00, BAI 01].

9 Bibliographie

[AGR 90] AGRAWAL D., ABBADI A., « Database transaction models for advanced applications » In Transaction Management in Database Systems, EL MAGARMID (Ed.), Morgan Kauffman, 1990.

[ALO 96] ALONSO G., AGRAWAL D., ABBADI A. E., MOHAN C., « Functionality and Limitations of Current Workflow Management Systems. » In *IEEE Expert Journa*l, 1996.

[Atr 94] ATRIA SOFTWARE INC., « ClearCase Product Summary », Rapport technique, Atria Software Inc., 24 Prime park Way, Natick, Massachusetts 01760, 1994.

[BAI 01] BAÏNA K, MUNIER M., BENALI K., « Un modèle de négociation pour les applications de travail coopératif » *soumis à BDA 2001*, Agadir, Maroc

[BAR 92a] BARGHOUTI N., « Concurrency Control in Rule-Based Software Development Environments». PhD thesis, Columbia University, 1992. Technical Report CUCS-001-92.

[BAR 92b] BARGHOUTI N., « Supporting Cooperation in the MARVEL Process-Centered SDE ». *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environment*s, vol. 17, n o 5, p. 21–31, December 1992.

[BEE 88] BEECH D., « A Foundation for Evolution from Relational to Object Databases ». In *Proceedings of the International Conference on Extending Database Technolog*y,p.251–267, venise, March 1988.

[BEL 94] BELKHATIR N., ESTUBLIER J. « ADELE–TEMPO: An Environment to Support Process Modelling and Enaction. » In *Software Process Modelling and Technology*, A. FINKELSTEIN J. K., NUSEIBEH B. (Ed), Research Study Press, 1994.

[BEN 97a] BENTLEY R., APPELT W., BUSBACH U., HINRICHS E., KERR D., SIKKEL K., TREVOR J., WOETZEL G., « Basic Support for Cooperative Work on the World Wide Web » In *International Journal of Human-Computer Studies: Special issue on Innovative Applications of the World Wide We*b, vol. 46, n o 6, p. 827–846, June 1997.

[BEN 97b] BENTLEY R., HORSTMANN T., TREVOR J., « The World Wide Web as enabling technology for CSCW: The case of BSCW ». In *Computer-Supported Cooperative Work:Special issue on CSCW and the We*b, vol. 6. Academic Press, 1997.

[BEN 98a] BENALI K., CANALS G., GODART C. et TATA S., « An Approach for Developing Cooperation in Project-Enterprises ». In *3 th International Conference on the Design of Cooperative System*s, Cannes, France, may 1998.

[BEN 98b] BENALI K., MUNIER M., GODART C., « Cooperation models in co-design » In *International Conference on Agile Manufacturing (ICAM'98*), Minneapolis, USA, June 1998.

[BEN 99] BENALI K., MUNIER M., GODART C., « Cooperation models in co-design » In *International Journal on Agile Manufacturing (IJAM*), vol. 2, n° 2, 1999.

[BER 97] BERNSTEIN P., NEWCOMER E., *Principles of Transaction Processing*. Morgan Kaufmann, 1997.

[BIG 98] BIGNON J., HALIN G., BENALI K., GODART C., « Cooperation models in co-design: application to architectural design ». In *4 th International Conference on Design and Decision Support Sys tems in Architecture and Urban plannin*g, Maastrich, July 1998.

[CAN 98] CANALS G., GODART C., MUNIER M., TATA S., « Supporting cooperation in project-enterprises ». In *Proceedings of the 1998 International Conference on Enterprise Networking and Computing (ENCOM'98*), Atlanta, Georgia, June 1998.

[CAR 89] CART M., FERRIÉ J., RICHY H., « Contrôle de l'exécution de transactions concur-rentes» In *Techniques et Sciences Informatique*s, n° 16, p. 225–240, Mars 1989.

[CAR 90] CART M., FERRIÉ J., « Integrating Concurrency Control into an Object-Oriented Database System ». In *Proceedings of Advances in Database Technology - EDBT'90, LNCS-41*6, p. 363–377, march 1990.

[CHR 94] CHRYSANTHIS P., K.RAMAMRITHAM, « Synthesis of Extended Transaction Models» In *ACM Transactions on Database System*s, vol. 19, n o 3, p. 451–491, September 1994.

[GAR 87] GARCIA-MOLINA H., SALEM K., « Sagas ». In *Proceedings of the 12th Annual ACM Conference on the Managemant of Dat*a, p. 249–259, San Francisco, California, May 1987.

[GRA 93] GRAY J., REUTER A., *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993.

[MOL 96] MOLLI P., « *Environnements de Développement Coopératifs* ». Thèse en Informatique, Université de Nancy I – Centre de Recherche en Informatique de Nancy, 1996.

[MOS 81] MOSS J. E., « Nested Transactions: An Approach to Reliable Distributed Computing». PhD thesis, MIT, 1981.

[MUN 98] MUNIER M., GODART C., « Cooperation services for widely distributed applications». In *Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE'98*), San Francisco Bay, USA, 1998.

[MUN 99a] MUNIER M., « Une architecture pour intégrer des composants de contrôle de la coopération dans un atelier distribué ». Thèse en Informatique, Université Henry Poincaré - Nancy I, Loria, janvier 1999.

[MUN 99b] MUNIER M., BENALI K., GODART C., « DisCOO, a really distributed system for cooperation». In *Networking and Information Systems Journal, Vol 2, N° 5-6*, 1999, hermes science publishing.

[MUN 00] MUNIER M., BAÏNA K., BENALI K., « *A Negotiation Model for CSCW* ». In *7th International Conference on Cooperative Information Systems -CoopIS'2000, LNCS 1901*., September 2000, Eilat, Israel.

[NOD 92] NODINE M., RAMASWAMY S., ZDONIK S., A Cooperative Transaction Model for Design Databases. In *Database transaction models for advanced application*s, EL MAGARMID (Ed), Morgan Kauffman, 1992.

[OMG 95] OMG, « The Common Object Request Broker: Architecture and Specification ». Rapport technique 2.0, Object Management Group, 1995.

[ROS 96] ROSENMAN M., GERO J., « Modelling multiple views of design objects in a collaborative CAD environment. ». In *Computer-Aided Desig*n, vol. 28, n o 3, p. 193–205, 1996.

[SHA 97] SHAPIRO, KLOOSTERMAN, RICCARDI, « PerDiS —a Persistent Distributed Store for Cooperative Application » In *Proceedings of the 3rd Cabernet Plenary Worksho*p, IRISA, Rennes, France, April 1997.

[TIC 89] TICHY, WALTER F., « RCS —A system for version control » In *Software — Practice and Experienc*e, vol. 15, n o 7, p. 637–654, July 1989.

[WAC 92] WACHTER H., REUTER A., « The ConTract Model » In *Database Transaction Models for Advanced Application*s, ELMAGARMID (Ed), p. 219–258. Morgan Kauffman, 1992.

[WEI 84] WEIHL W., « Specification and Implementation of Atomic Data Types » PhD thesis, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA, March 1984.

[WEI 89] WEIHL W., « Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types » In *ACM Transactions on Programming Languages and System*s, vol. 11, n o 2, p. 249–282, April 1989.

[WEI 96] WEISS M., JHONSON A., KINIRY J., « Distributed Computing: Java, CORBA, and DCE» Open Software Foundation Version 2.1, February 1996.

[WFM 97] WFMC, « Terminology & Glossary ». Rapport technique WFMC-TC-1011, Issue 2.0, Workflow Management Coalition, June 1997.