# CONCEPTS ET MÉCANISMES POUR LA MISE EN OEUVRE D'UN ENVIRONNEMENT D'ÉDITION COOPÉRATIVE SUR UN RÉSEAU À GRANDE ÉCHELLE

THÈSE N$^{O}$ 1335 (1995)

PRÉSENTÉE AU DÉPARTEMENT D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

FRANÇOIS PACULL

D.E.A. en informatique de l'Institut National Polytechnique de Grenoble, France

de nationalité française

acceptée sur proposition du jury:

Prof. A Schiper, rapporteur
Prof. G. Coray, corapporteur
Dr V. Quint, corapporteur
Prof. M. Raynal, corapporteur

Lausanne, EPFL
1995

## Résumé

Dans ce travail de thèse nous abordons un aspect de l'édition coopérative peu exploré jusqu'à présent : la considération des limitations imposées par un réseau à grande échelle (typiquement Internet). Ce modèle de système implique l'hétérogénéité des machines, des délais de communication non bornés et des pannes. Dans ce contexte, les techniques traditionnelles utilisées pour l'édition coopérative dans un réseau local sont inadaptées.

Le modèle proposé dans cette thèse est basé sur une décomposition dynamique du document en parties indépendantes, maintenues individuellement par un noyau dupliqué. La décomposition est contrôlée par les utilisateurs eux-mêmes, qui peuvent ainsi agir sur la concurrence autorisée au niveau du document. Plus un document est fragmenté, plus la concurrence potentielle est grande. La duplication des parties indépendantes au sein d'un noyau partagé permet d'assurer la disponibilité du document aux utilisateurs, et ce indépendamment des performances des sites utilisateurs. Les communications sont réduites au minimum, autorisant une utilisation à large échelle. Les utilisateurs interagissent avec le noyau soit en récupérant une copie locale d'un objet du noyau qu'ils peuvent modifier à loisir dans leur environnement local, soit au contraire en transmettant au niveau du noyau une copie locale. La gestion de la cohérence se faisant à ce moment là suivant plusieurs politiques que l'utilisateur peut choisir en fonction de la maturité du document ou encore des autres utilisateurs.

Concernant la gestion de la cohérence, un critère à été défini : la NTR-linéarisabilité. C'est un critère mieux adapté que la linéarisabilité, qui permet en particulier une mise en œuvre non bloquante en cas de panne (panne d'un site utilisateur ou panne d'un site du noyau), ce qui est important dans le cadre des réseaux à large échelle. Ce critère conserve la propriété de localité de la linéarisabilité. Contrairement à l'approche classique, ce critère de cohérence n'est pas appliqué aux opérations lecture-modification-écriture garantissant l'atomicité des actions suivantes : chargement dans un fichier d'une partie de document, modification grâce à l'éditeur, puis finalement sauvegarde de la partie ainsi modifiée. La mise en œuvre assure uniquement l'atomicité des opérations d'écriture et de lecture. L'atomicité des opérations lecture-modification-écriture est considérée uniquement au niveau applicatif, afin de prendre en considération les connaissances des utilisateurs. Ainsi, la manière dont elle va être réalisée est laissée au choix des utilisateurs, grâce à plusieurs politiques de contrôle de concurrence qui dépendront de règles établies entre les utilisateurs, soit optimistes basées sur des détections de conflit a posteriori, soit pessimistes basées sur des capacités, soit hybrides.

## Abstract

The following dissertation addresses one aspect of cooperative editing which has not been well examined until now : the consideration of hardware limitations imposed by a large scale distributed system (namely the Internet). This system model implies heterogeneous hardware, unbounded communication delays, and process failures. In this context, mechanisms usually used for cooperative editing in a local area network are not suitable.

The model proposed here is based on a dynamic decomposition of the document into independant parts, maintained by a replicated kernel. The decomposition is controlled by the users themselves, who can then act on the concurrency allowed at the document level. The more a document is fragmented, the more concurrency is allowed. The replication of the independant document parts within the kernel, ensures the document availability and this regardless of the performances of the users' sites. Communication is reduced to minimum, thus allowing use in large scale distributed system. Users interact with the kernel either by copying a kernel object to their own local environment where it can be modified, or by handling over the modified object to the kernel. Consistency is managed according to several policies that can be chosen by the user depending on the document maturity or the required interaction with the other users.

A new criterion has been defined to ensure consistency : the NRT-linearizability. This criterion is more adapted to our problem than linearizability, and allows in particular a non-blocking implementation even in the case of failures (user site as well as kernel site failure), which is important when considering large scale distributed system. This criterion maintains the locality property of the linearizability. Contrary to the classical approach, this criterion is not applied to the read-modify-write operations ensuring the atomicity of the following actions : loading a document part into an editor, modifying it with the editor, and storing the modified part. The atomicity of the read-modify-write operations is only considered ot the application level and takes into account the knowledge of the users. Thus, the user is free to choose the way it will be performed His/her choice will be made between several concurrency control policies based on rules established between the users, either optimistic based on conflict detection, pessimistic based on capabilities, or hybrid.

## Remerciements

Je tiens à remercier :

le professeur A. Schiper d'une part pour avoir accepté de diriger ce travail de recherche et d'autre part pour ses précieux conseils et son soutien. Disponibilité, confiance et clairvoyance. Ceux qui ont eu la chance de travailler avec lui comprendront.

les membres du jury d'avoir accepté de consacrer à l'examen de cette thèse leur temps et leurs compétences.

notre sécrétaire, K. Verhame ainsi que les membres du Laboratoire de Systèmes d'Exploitation qui m'ont à un moment ou à un autre apporté leur soutien moral ou matériel.
Tout particulièrement :

R. Guerraoui pour son intérêt pour le sujet qui a permis de nombreuses discussions enrichissantes. Son expérience et son esprit critique m'ont été une aide appréciable.

B. Garbinato, K. Mazouni pour les remarques pertinentes et les précieux commentaires dont ils m'ont fait part lors de la première lecture de cette thèse. Je leur suis reconnaissant de la disponibilité dont ils ont fait preuve.

A. Sandoz avec qui j'ai eu de fructueuses discussions à l'origine du projet.

L. Clavien et J.-C. Lugeon pour leur contribution à l'environnement DUPLEX.

J. Vachon et U. Wilhelm pour leur aide dans le maniement des expressions anglo-saxonnes. Je remercie également U. Wilhelm de m'avoir libéré de ma charge d'enseignement dans la phase finale de rédaction

Que cette thèse soit un témoignage de ma gratitude envers mes parents et ma sœur qui m'ont apporté leur soutien moral tout au long de cette thèse.

Je ne saurais terminer ces remerciements sans un énorme merci à Laurence qui, partageant ma vie, a supporté avec patience et abnégation mon stress, mon caractère et mes nombreuses absences notamment durant la phase finale de rédaction. De plus, je la remercie pour sa contribution à rendre l'orthographe de cette thèse conforme à l'académie française.

Un "pardon" à Emelyne pour avoir failli à mon rôle de père pendant ces derniers mois.

C'est par une porte bleu que l'on entre dans le monde,

où les chatons font de la peinture

# Table des matières

# Liste des Figures

# Liste des Algorithmes

# Introduction

Les dix dernières années ont vu le nombre des machines augmenter et la taille des réseaux s'accroître dans une proportion considérable. Parallèlement à cela, une pléthore d'applications réparties mettant en jeu ces nouvelles ressources ont vu le jour. Parmi celles-ci se trouve toute une catégorie d'applications regroupées sous le vocable "Computer Supported Cooperative Work" (CSCW) et consacrées au travail coopératif assisté par ordinateur. Le but de ces applications est de fournir la logistique nécessaire à un groupe de personnes qui travaillent à un but commun, et qui se trouvent géographiquement disséminées. Si ce mode de travail coopératif existe depuis longtemps, il prend un tout autre intérêt lorsque les systèmes informatiques peuvent apporter un gain de temps précieux, en libérant les personnes des déplacements physiques (réunions de travail) et des tâches fastidieuses liées à la diffusion de l'information (téléphone, fax, courrier), en assurant la cohérence des information manipulées ou encore en coordonnant les actions de chacune des parties en présence.

Parmi ces applications, le travail d'édition coopérative constitue une classe qui mobilise une bonne partie de la communauté scientifique travaillant sur le CSCW. En effet, ce type d'outils représente un réel intérêt puisque selon les études, 67 à 85% des articles, papiers, rapports et documentations techniques sont écrits conjointement par plusieurs auteurs [Ede 90, Bair 85].

De nombreuses réalisations ont vu le jour ces dernières années tant au niveau de projets universitaires que de grands groupes industriels. Cependant, un aspect qui a souvent été négligé est celui qui concerne la prise en compte des limitations du matériel. Par exemple, s'il est possible d'offrir pour un réseau local de machines, des outils où chaque utilisateur voit en temps-réel les actions des autres utilisateurs, cela devient impossible sur un réseau à large échelle en raison des mauvaises performances des communications et des risques de pannes.

Ce travail a pour but de proposer un environnement qui tient compte de ces contraintes, afin d'offrir un environnement de travail coopératif à la fois convivial et efficace.

La thèse est décomposée en trois parties.

La première partie est consacrée à une présentation de l'édition coopérative dans un cadre général, et présente une synthèse des travaux effectués dans ce domaine. Le chapitre 1 présente les caractères généraux de l'édition coopérative tant sur le plan humain (comment les personnes interagissent dans une collaboration), que sur le plan matériel (implication de la collaboration sur les contraintes et les performances). Le chapitre 2 présente les différents composants que doit comporter un environnement d'édition coopérative. Dans cette partie, nous faisons ressortir un certain nombre de remarques qui servent de "cahier des charges" pour la mise en œuvre de l'environnement d'édition coopérative DUPLEX.

La seconde partie de la thèse est consacrée au problème de la cohérence des données partagées. Le chapitre 1 formalise les critères de cohérence dit "forts" proposés dans la littérature. Ces critères répondent mal aux nécessités d'un système distribué sur un réseau à large échelle. Notre contribution en ce domaine a été de définir un nouveau critère de cohérence (la *NTR-linéarisabilité*) permettant de répondre aux problèmes spécifiques du système considéré. Le chapitre 2 présente les modèles considérés dans la littérature en ce qui concerne la cohérence et le contrôle de concurrence. Ces modèles sont mal adaptés à un réseau à large échelle, car les opérations considérées, le critère de cohérence choisi, et la mise en œuvre pessimiste font que ce type d'approche n'est pas réalisable hors d'un réseau local. Outre le choix d'un critère de cohérence mieux adapté à la large échelle et une mise en œuvre optimiste, DUPLEX propose une décomposition dynamique du document en parties indépendantes, et une meilleure prise en considération des connaissances des utilisateurs afin d'augmenter la concurrence autorisée sur le document. Par exemple, pour une partie de document donnée, le système n'impose le critère de cohérence que sur les opérations de lecture et d'écriture. Le problème d'assurer que la partie du document ne sera pas modifiée par un autre utilisateur entre la lecture et l'écriture est de la responsabilité de l'utilisateur par le biais du choix entre plusieurs politiques de contrôle de concurrence. Ainsi, il est possible de choisir celle qui est la mieux adaptée à la partie du document, à l'état de maturité du document, et aux autres utilisateurs travaillant sur la même partie du document. Le troisième chapitre présente la mise en œuvre du protocole assurant la NTR-linéarisabilité dans un contexte dupliqué. La duplication est utilisée pour assurer une tolérance aux pannes et une meilleure disponibilité des données.

La troisième et dernière partie de cette thèse est consacrée à la description des concepts et mécanismes mis en œuvre dans DUPLEX. Le premier chapitre présente l'environnement DUPLEX dans son ensemble. C'est un environnement basé sur un noyau de stockage qui gère le document. Ce noyau est partagé par les utilisateurs, qui gère le document. Un utilisateur travaille dans un environnement local, et interagit avec le noyau, pour copier une partie du document localement dans son environnement, ou au contraire pour rendre public son travail en transférant vers le noyau une copie locale à son environnement. La

cohérence des données est traitée par le noyau au moment du transfert. Le second chapitre décrit les principales caractéristiques de la décomposition dynamique du document. Le document est décomposé en petites parties au début de la collaboration, permettant aux utilisateurs un maximum de concurrence, puis les parties sont regroupées en parties plus importantes, jusqu'au document complet, quand il convient de faire aboutir le document à une version définitive. Le troisième chapitre présente en détail le noyau qui est responsable de la disponibilité et de la cohérence du document partagé. Le quatrième chapitre présente l'environnement local d'un utilisateur. Ces deux derniers chapitres mettent en évidence les propriétés des approches qui ont été introduites, et insiste sur les aspects qui font que DUPLEX répond aux problèmes posés par une collaboration entre des utilisateurs dispersés sur un réseau à large échelle.

# Partie I

# État de l'art

# Chapitre 1

# De l'édition coopérative

## 1 Introduction

Le domaine de l'édition coopérative fait appel à un vaste spectre de compétences. En effet, doivent être considérés non seulement les aspects techniques liés au matériel, mais également les aspects humains : interface homme-machine et intégration d'un individu avec ses spécificités dans un groupe d'individus.

La *collaboration* peut être définie comme le processus par lequel un groupe de participants essaie de réaliser une certaine tâche en commun. Pour ce faire, les participants échangent données et points de vue par l'intermédiaire d'outils de communication. Cette définition de la collaboration est suffisamment générale pour n'être liée ni au but (l'édition coopérative) ni aux méthodes utilisées.

Dans le cadre de l'édition coopérative, cette tâche commune est un document au sens large du terme. Ce peut être un article conjointement rédigé par une équipe de chercheurs, une documentation (par exemple celle d'un système d'exploitation) faisant intervenir des programmeurs, des techniciens et des commerciaux, un contrat international réunissant plusieurs cabinet d'avocats et juristes. Le point commun entre tous ces documents est qu'ils impliquent plusieurs personnes géographiquement disséminées.

Ce chapitre a pour but de présenter l'édition coopérative dans sa généralité et de dégager un certain nombre de points qui ont influencé notre travail. Pour ce faire, la section 2 précise les méthodes associées au travail d'édition afin d'en dégager les outils nécessaires. La section 3 complète la présentation générale de l'édition coopérative en mettant l'accent sur un certain nombre de besoins que nous avons identifié. Finalement, la section 4 présente les contraintes matérielles et leurs adéquations compte tenu des besoins identifiés. Nous précisons également l'influence de ces contraintes sur les performances.

Dans ce chapitre et dans le suivant, un certain nombre de remarques sont formulées. Elles serviront de base pour l'élaboration du cahier des charges de notre plate-forme d'édition collaborative : DUPLEX.

## 2 Processus d'édition coopérative

L'étude de Posner [Posner 92] montre que le processus d'édition coopérative fait apparaître d'une part plusieurs types d'activités, et d'autre part une ou plusieurs stratégies de coordination.

L'édition, qu'elle soit coopérative ou non, fait appel à plusieurs types d'activités puisqu'elle est constituée de phases distinctes : la définition d'un plan plus ou moins précis, la rédaction proprement dite, la modification éventuelle du plan, les révisions internes et/ou externes, les corrections et raffinements, puis finalement une phase de polissage afin de dégager la version finale; les phases intermédiaires constituent un processus itératif.

L'édition coopérative nécessite également une ou plusieurs stratégies de coordination puisqu'il faut prendre en compte les actions des différentes personnes impliquées dans la coopération. Cette coordination est réalisée par un dialogue implicite ou explicite entre les différents protagonistes et peut, suivant la phase considérée, revêtir des formes diverses mettant en oeuvre des communications en temps-réel[1] où émetteur et récepteur sont actifs au même instant (e.g., talk sous Unix) ou asynchrones où la réception peut se faire en différé (e.g., E-Mail).

Les types d'activités et leurs stratégies de coordination associées sont :

- **Définition du plan du document** Associée à la définition du plan, se trouve une phase de *brainstorming* pendant laquelle les idées des différents participants sont évaluées, modifiées et raffinées, jusqu'à l'émergence d'un point de vue commun (ou à défaut, celui d'une majorité). Différents types d'actions ont été identifiés [Ellis 91] au sujet de ce travail de groupe :
  - **Apport indépendant** *(independent entry)* où un participant accorde peu d'attention à ceux que font les autres membres du groupe.
  - **Apport incrémental** *(reflective entry)* où un participant enrichit une proposition déjà connue du groupe. Celle-ci peut être issue de n'importe quel participant.
  - **Apport consensuel** *(consensus entry)* où c'est le groupe qui décide de l'apport à effectuer.

[1] La littérature utilise le terme de *communication synchrone*, pour notre part nous pensons que *communication en temps-réel* est plus approprié.

  - **Apport partitionné** *(partitioned entry)* où certains membres du groupe sont désignés comme étant responsables d'un point particulier du plan.
  - **Apport procès-verbal** *(recorded entry)* où un participant paraphrase ce qui est discuté par le groupe.

- **Rédaction** La phase de rédaction du premier jet peut faire appel à plusieurs stratégies [Posner 92]. Mis à part la première (rédacteur séparé), elles font référence à un groupe de participants travaillant de concert à la phase de rédaction.

  - **Rédacteur séparé** *(separate writer)* où chaque participant se voit attribuer une tâche distincte. Cette approche fait appel à un travail plus personnel puisque les interactions avec les autres participants n'interviennent que dans un second temps, lorsque le travail de chacun est finalement évalué, commenté, corrigé puis intégré dans le document final.
  - **Scribe** où l'un des participants est responsable de la transcription en direct des idées émanant du groupe (ce mode n'est pas sans rappeler l'apport procès-verbal dans la définition du plan (Cf. ci-dessus).
  - **Rédacteur unique** *(single writer)* où l'idée est quelque peu différente puisque la transcription n'est pas effectuée en temps-réel : un participant est responsable après la discussion de synthétiser les idées échangées par le groupe.
  - **Rédaction conjointe** *(joint writing)* où le groupe peut conjointement rédiger la partie concernée. La communication entre auteurs est alors très présente et nécessite un éditeur acceptant en temps-réel des modifications provenant de plusieurs origines ou un moyen de communication en temps-réel annexe (vidéo-conférence ou téléphone).

- **Révision / modification** Les phases de révision et modification font appel à une succession d'interactions entre les participants responsables de chacune des deux phases. La communication est alors soit directe (un réviseur peut par exemple demander à l'auteur des précisions sur un point resté obscur à ses yeux), soit indirecte (des annotations sont ajoutées au texte lui-même; dans ce cas la connaissance des auteurs n'est pas nécessaire). Ceci entraîne des communications en temps-réel ou asynchrones, suivant la disponibilité de chacun.

- **Polissage** La phase de polissage, suivant le nombre de personnes qu'elle implique et leurs façons de travailler, peut revêtir deux formes. Dans la première, chacun procède à une passe et modifie le texte en conséquence. Dans la seconde, les différents protagonistes mettent en œuvre une phase de *brainstorming* dans laquelle ils décident d'un commun accord des modifications à apporter au texte. Ces deux formes peuvent être bien entendu combinées à loisir.

En conclusion, un environnement d'édition coopérative doit offrir des média de communication entre les participants au même titre que des outils d'édition plus traditionnels. De plus, l'édition coopérative se compose de phases d'activités fortement synchrones où les participants travaillent en même temps et de phases asynchrones où les participants travaillent à des instants différents [Miles 93]. La nature même du travail est variée puisque l'on peut identifier des activités diverses: planification, rédaction, révision, communication, etc. Finalement, la manière dont les participants partagent l'information et coordonnent leurs activités dépend de beaucoup de facteurs. En particulier des relations entre les participants et de leurs rôles sociaux aussi bien que de l'état d'avancement du projet commun [Lewis 88].

# 3 Besoins

Certains besoins d'un environnement d'édition coopérative [Greif 92] peuvent être mis en avant. Ce sont le partage des données, l'autonomie de chacun des participants, la fiabilité du système, le caractère dynamique de la collaboration, les aspects interface-utilisateur et finalement les performances.

### Partage des données

La collaboration impose par essence un partage des données. Cela soulève les problèmes de concurrence d'accès à ces données (e.g., écritures concurrentes), de protection d'informations à caractère privé (e.g., brouillon, premier jet) et finalement de protection de la communauté contre les erreurs d'un utilisateur (e.g., fausses manipulations).

### Fiabilité

Le système doit bien sûr pouvoir résister aux pannes possibles des sites *(node failure)* et des canaux de communication entre les sites *(link failure)*. Cela signifie d'une part qu'une panne ne doit pas corrompre les données, mais également que celles-ci doivent rester accessibles même si plusieurs sites sont en panne. Cette tolérance aux pannes doit être transparente aux utilisateurs.

### Autonomie

Si un individu est une pièce du puzzle que constitue la collaboration, il doit toutefois être indépendant des autres participants et des sites impliqués dans le projet. Par exemple, il est inconcevable qu'un auteur impliqué uniquement dans la rédaction d'un chapitre $X$ ne puisse plus travailler sous prétexte qu'un des participants n'est plus accessible par le réseau

ou bien que le site (ou les sites) où est stocké le chapitre $Y$ est en panne. De plus, un mode dégradé doit être offert afin de permettre aux utilisateurs déconnectés du réseau de pouvoir exécuter certaines actions; cette déconnexion étant volontaire (e.g., ordinateur portable) ou involontaire (e.g., panne locale du réseau).

#### Configuration dynamique

Les participants doivent pouvoir évoluer librement, être actifs ou oisifs, raccordés au réseau ou pas. Un participant n'a pas à être associé au site où il travaille habituellement. De plus, il doit pouvoir rejoindre ou quitter la communauté à tout moment sans qu'il soit pour autant nécessaire de procéder à une reconfiguration du système. En un mot, un participant en déplacement doit pouvoir travailler depuis l'endroit où il se trouve sans impliquer de manipulations importantes.

#### Environnement d'édition puissant

L'environnement d'édition doit offrir au moins les mêmes fonctionnalités qu'un environnement mono-utilisateur. De plus, il doit être configurable afin de permettre à chacun des participants de garder ses habitudes de travail. Il est effectivement peu fréquent qu'il y ait, au sein d'un groupe, un consensus a priori sur le type d'outils utilisés. Finalement, l'aspect collaboratif doit être aussi convivial que possible et s'adapter à l'utilisateur et non l'inverse.

#### Efficacité

L'environnement d'édition coopérative doit être efficace en terme de temps de réponse et d'opérations réalisables grâce aux outils fournis ce qui, compte tenu des besoins précédemment cités, constitue le défi.

## 4 Contraintes matérielles et qualité de service

Un compromis doit être réalisé entre l'efficacité d'une part et les besoins et les contraintes matérielles d'autre part. Pendergast [Pendergast 90] a défini quatre points permettant d'analyser les choix possibles : concurrence, granularité des données, localisation de celles-ci et communication. Nous étendons ces points en considérant également le système sous-jacent et le niveau de cohérence souhaité.

## 4.1 Concurrence des actions

La concurrence des actions peut être définie comme la non séquentialité des actions initiées par les utilisateurs. Cette concurrence est généralement bénéfique.

> (1) *Augmenter la concurrence permet de diminuer le temps de réponse pour chaque utilisateur, donc d'augmenter les performances du système*[2].

## 4.2 Granularité des données partagées

Un document peut être considéré comme un bloc monolithique ou comme un ensemble de parties plus petites. Dans ce dernier cas, deux unités sont à considérer. D'une part, l'*unité élémentaire* ou atomique, qui correspond à la plus petite partie accessible par l'utilisateur (e.g., si la ligne de texte est définie comme unité élémentaire alors il n'est pas possible d'accéder séparément aux caractères qui la composent[3]). D'autre part, l'*unité utilisable*, composée d'une ou plusieurs unités élémentaires. Une opération invoquée sur une unité utilisable sera exécutée sur toutes les unités élémentaires qui la compose. Informellement, l'unité élémentaire définit le *grain minimal* pour une opération, et l'unité utilisable définit le *grain* effectif que peut choisir l'utilisateur.

La littérature fait référence à deux approches en ce qui concerne la modélisation d'un document. La première considère qu'un document n'est qu'une suite de lignes, voire de caractères, la seconde se base sur la structure récursive d'un document décomposé en *parties logiques*, c'est-à-dire en sections, contenant des sous-sections et ainsi de suite jusqu'aux paragraphes. Informellement, à une partie logique correspond une entrée dans la table des matières du document. Le modèle de décomposition du document influence la définition de la granularité.

Par exemple dans la première catégorie, Mace [Newman-Wolfe 91a] et Mule [Pendergast 90] considèrent la ligne comme l'unité élémentaire et un bloc contigu de lignes comme l'unité utilisable. MMM [Bier 92, Bier 91] pour sa part définit la ligne comme étant à la fois l'unité élémentaire et l'unité utilisable. ShrEdit [McGuffin 92] et DistEdit [Knister 93] considèrent le caractère comme unité élémentaire et comme unité utilisable. GROVE [Ellis 91] considère que l'unité utilisable est soit un caractère (mode par défaut), soit un bloc de texte.

Dans la deuxième catégorie, Quilt [Fish 88, Leland 88] définit la partie logique la plus grande (section) comme étant à la fois l'unité élémentaire et l'unité utilisable. Ainsi, deux utilisateurs peuvent accéder à deux sections concurremment, mais en revanche ils ne peuvent pas accéder de manière concurrente à deux sous-sections de la même section. D'une manière

[2] Le but des remarques (1) à (33) est de fournir des critères qui justifieront les choix de conception de notre plate-forme d'édition coopérative.

[3] La ligne entière est considérée comme modifiée dès que l'utilisateur modifie un caractère.

similaire, SharedBook [Lewis 88] considère des parties de document (appelées entrées) qui définissent à la fois l'unité élémentaire et l'unité utilisable. Ces entrées sont construites sur la base de parties logiques d'un document. Dans ces deux cas, l'unité utilisable est fixée par le système et imposée à l'utilisateur. En revanche, Iris [Borghoff 93], CES [Greif 92] ou MultimETH [Lubich 90] définissent le paragraphe comme l'unité élémentaire et n'importe quelle partie logique du document comme unité utilisable. Ainsi, il devient possible d'accéder concurremment à deux sous-sections de la même section; il n'est par contre pas possible d'accéder concurremment à une section et à une de ses sous-sections.

La première méthode permet une granularité beaucoup plus fine mais occulte en revanche la structure du document. Par exemple, il est possible de faire un bloc avec les trois dernières lignes du chapitre 1 et les cinq premières du chapitre 2. Quelle est la sémantique qui peut être associée à un tel bloc ? De plus, pouvoir sélectionner un tel bloc signifie que l'on doit pouvoir effectivement y apporter des modifications. Effacer ce bloc entraîne toutefois de profonds bouleversements à la structure du document.

(2) *Plus la granularité est faible plus la concurrence peut être grande.*

Une unité utilisable fixée par le système constitue une limitation quand le grain est important. En effet, l'utilisateur ne peut pas utiliser uniquement la partie dont il a réellement besoin mais est contraint à utiliser un sur-ensemble.

(3) *Si le grain est fixé par le système, sa taille doit être faible.*

Diminuer le grain permet de favoriser la concurrence, cependant pour obtenir la même quantité d'information, il faudra réaliser un plus grand nombre de requêtes, donc d'accès au réseau.

(4) *Plus la granularité est faible plus le réseau sera sollicité.*

## 4.3 Gestion des données

La gestion des données peut être caractérisée par deux aspects: (1) données dupliquées ou non dupliquées, (2) centralisées ou réparties. En première approche, si l'on considère la manière dont les données sont utilisées nous pouvons formuler les deux règles suivantes.

(5) *Si les opérations de consultation sont les plus fréquentes alors il est préférable que les données soient dupliquées et présentes sur chacun des sites des utilisateurs.*

(6) *Si les opérations de modification sont les plus fréquentes alors il est préférable d'adopter une solution centralisée avec une seule copie.*

Malheureusement, il n'est pas simple de définir dans le cadre de l'édition coopérative laquelle de ces deux approches opposées est la bonne. De plus, les précédentes règles ne tiennent pas compte des événements extérieurs tels que les pannes. Nous proposons donc une description plus pragmatique.

### Stockage centralisé vs. stockage réparti

La centralisation implique que le stockage se fait sur un ou plusieurs sites dédiés alors que la répartition suppose que chacun des sites utilisateurs est responsable du stockage d'une partie des données (un site stocke par exemple les données les plus fréquemment accédées par l'utilisateur de ce site).

L'approche centralisée va à l'encontre de l'autonomie puisque tous les sites utilisateurs dépendent du serveur central. Il constitue un goulot d'étranglement et un site critique en cas de pannes.

(7) *La solution centralisée entraîne plus d'activité réseau.*

L'approche répartie nécessite que les sites utilisateurs offrent la fonctionnalité de serveur en plus de celle de client. De plus, certains sites peu performants peuvent dégrader les performances globales de tout le système en rendant leurs données difficilement accessibles.

(8) *La solution répartie favorise le travail des utilisateurs possédant leurs données locales au détriment des autres.*

Parmi les systèmes présentés dans la littérature, Mule [Pendergast 90] , DistEdit [Knister 93], Iris [Borghoff 93], GROVE [Ellis 91], et CES [Greif 92] proposent une distribution des données alors que MultimETH [Lubich 90], Mace [Newman-Wolfe 91a], MMM [Bier 92, Bier 91] et Quilt [Fish 88, Leland 88], Prep [Neuwirth 90], SharedBook [Lewis 88] ont mis en œuvre une solution centralisée.

### Copie unique vs. duplication

Dupliquer les données permet de pallier aux pannes des sites de stockage. En effet, si plusieurs copies sont disponibles, la probabilité qu'une copie soit accessible croît proportionnellement. Toutefois, la duplication nécessite des mécanismes de gestion pour assurer la cohérence mutuelle des copies.

(9) *Maintenir cohérent un ensemble de copies en lieu et place d'un exemplaire unique nécessite plus d'accès au réseau.*

Sans duplication, dans le meilleur des cas, une donnée peut rester inaccessible pendant toute la durée de la panne du site où elle est stockée (cas favorable où une copie en mémoire permanente existe)[4]. En l'absence de pannes, les performances sont toutefois optimales.

> (10) *Ne pas dupliquer les données laisse les utilisateurs à la merci d'une panne de site ou de lien de communication.*

Parmi les systèmes présentés dans la littérature, Mule [Pendergast 90], Iris [Borghoff 93], GROVE [Ellis 91], et DistEdit [Knister 93] proposent une duplication totale des données. CES [Greif 92] ne duplique que les données concernant la structure du document; le contenu effectif n'est pas dupliqué. Enfin, MMM [Bier 92, Bier 91], MultimETH [Lubich 90], Quilt [Fish 88, Leland 88], Mace [Newman-Wolfe 91a], Prep [Neuwirth 90] et SharedBook [Lewis 88] ne dupliquent pas les données.

## 4.4 Contraintes liées au système

### Hétérogénéité

L'hétérogénéité est définie comme la diversité des machines, des systèmes d'exploitation et des espaces d'adressage. Par exemple, si l'on veut pouvoir garantir qu'un site quelconque puisse être utilisé comme site utilisateur, il faut que l'interface utilisateur soit indépendante d'un environnement spécifique.

A titre d'exemple, Mule [Pendergast 90] ne fonctionne que sur un réseau local de PC, ShrEdit [McGuffin 92] et Prep [Neuwirth 90] sont liés au monde Macintosh. Dans le même ordre d'idée, Quilt [Fish 88, Leland 88] est basé sur une plate-forme logicielle particulière (Orion [Banerjee 87]).

> (11) *Baser un environnement d'édition coopérative sur un type de matériel ou une plate-forme logicielle particulière nuit à sa portabilité.*

L'utilisation de standards logiciels permet d'offrir le même environnement sur des types de matériels différents. Par exemple, l'interface utilisateur de Iris [Borghoff 93] est basée sur Interview et celle de DistEdit [Knister 93] sur X11.

> (12) *L'utilisation de standards logiciels permet de masquer les spécificités matérielles d'un système.*

Finalement, l'environnement de travail ne doit pas être figé, mais au contraire pouvoir être personnalisé pour chacun des utilisateurs. Dans ce sens, DistEdit [Knister 93], Mace

[4] Le pire des cas étant une perte de l'information non stockée en mémoire permanente.

[Newman-Wolfe 91a], Quilt [Fish 88, Leland 88] permettent de paramétrer les outils utilisés (par exemple l'éditeur de texte). Par contre, les systèmes basés sur des éditeurs ad-hoc, GROVE [Ellis 91], ShrEdit [McGuffin 92] imposent aux utilisateurs une phase d'apprentissage d'un $n^{ème}$ éditeur.

> (13) *Un environnement d'édition coopérative doit s'adapter aux utilisateurs et non l'inverse.*

#### Taille du système

La taille du système est déterminée par la distance entre les sites. Par distance entre sites, nous entendons délais de communication plutôt que distance réelle, puisqu'une liaison par satellite peut être plus rapide qu'une liaison entre deux bâtiments au travers d'un modem.

L'aspect communication est très important puisque la dissémination géographique peut impliquer des communications de mauvaise qualité, voire défaillantes. La notion de distance au sens métrique du terme peut également avoir une influence curieuse: il peut être impossible de mettre en œuvre des communications en temps-réel uniquement parce que les participants se trouvant dans des fuseaux horaires différents, il est difficile de trouver une plage horaire permettant un rendez-vous.

> (14) *Les performances du réseau seront prépondérantes pour assurer des communications en temps-réel.*

En revanche, le nombre de sites d'un système a peu d'influence, car les sites impliqués par une collaboration sont généralement peu nombreux même si le système distribué global auquel les sites appartiennent est très important. Le système distribué qui nous intéresse est en fait la restriction du système global aux sites concernés par la collaboration : sites utilisateur ou de stockage des données.

> (15) *Le système réellement impliqué dans une collaboration est petit même si le système sous-jacent est très important (e.g., Internet).*

Bien que parmi les environnements d'édition coopérative proposés dans la littérature, certains envisagent une utilisation dans des systèmes à grande échelle, aucune étude de performances n'a été publiée à ce jour.

### 4.5 Communication

La communication concerne l'échange d'informations entre les participants géographiquement disséminés. Deux types de fonctionnements sont à considérer: le *mode synchrone* où

chaque utilisateur voit les interactions des autres participants en temps-réel, et le *mode asynchrone* où les modifications ne sont pas immédiatement répercutées globalement.

(16) *Plus les mises à jour sont fréquentes plus le réseau sera sollicité.*

Le premier mode est désigné par l'acronyme WYSIWIS [Stefik 87b] pour *What You See Is What I See* qui exprime bien la philosophie sous-jacente. Plusieurs éditeurs GROVE [Ellis 91], ShrEdit [McGuffin 92], Mule [Pendergast 90] et MMM [Bier 92, Bier 91] fonctionnent selon ce mode. Ils sont caractérisés par une granularité fine (caractère ou ligne) qui permet des mises à jour fréquentes (la fréquence des mises à jour étant liée à la taille des données à modifier.).

(17) *Plus la granularité est fine plus les mises à jour peuvent être fréquentes.*

Poussé à l'extrême, ce mode implique que non seulement *l'espace de travail* (les fichiers manipulés) est le même pour tous les utilisateurs mais encore que *l'espace d'affichage* (généralement la fenêtre de visualisation) est identique. Par exemple, le défilement occasionné par un des utilisateurs se répercute sur tous les écrans et la position des curseurs de chacun des participants est visible par tous.

Le mode synchrone ou temps-réel peut être relaxé [Stefik 87a] suivant quatre directions :

- **l'espace d'affichage :** tous les participants travaillent sur les mêmes données mais chacun des participants peut contrôler sa fenêtre d'affichage et visualiser une partie particulière de ces données indépendamment des autres participants.
- **le temps :** tous les participants travaillent sur les mêmes données, mais les mises à jour ne sont pas répercutées immédiatement ce qui peut impliquer des divergences momentanées entre les différents espaces d'affichage, voire entre les espaces de travail.
- **les données :** tous les participants ne travaillent pas sur les mêmes données, leurs espace de travail sont différents. L'affichage est alors lié au travail de chacun.
- **la vue :** les données (espace de travail) sont les mêmes mais la manière dont elles sont représentées diffère d'un espace d'affichage à l'autre. Par exemple, une feuille de calcul peut être représentée sous forme textuelle par un participant et sous forme graphique par un autre.

Cette relaxation peut être plus ou moins importante jusqu'à envisager des espaces de travail et d'affichage complètements indépendants.

A titre d'exemple, SharedBook [Lewis 88] propose le mécanisme suivant : une mise à jour n'est pas immédiatement visualisée sur les sites des autres participants, mais un mécanisme

garantit que ceux-ci ne pourront pas effectuer d'opération avant d'avoir reçu cette mise à jour. Cette technique est également proposée dans [Garreth 86] sous le nom de *notification scheme*.

Un autre mode est défini par l'acronyme WYSIWIMS [Newman-Wolfe 91b] pour *What You See Is What I May See* : les mises à jour ne sont pas systématiquement diffusées, mais il est garanti que les utilisateurs peuvent y accéder s'ils le souhaitent.

Iris [Borghoff 93] définit un mode appelé semi-synchrone, dans lequel les participants travaillent sur des versions divergentes d'une même copie mais peuvent aisément visualiser et fusionner les différences entre leur propre version et celles des autres.

Finalement, un mode purement asynchrone considère que les utilisateurs travaillent dans un espace de travail local et qu'ils diffusent leurs modifications quand bon leur semble (e.g., Prep [Neuwirth 90], Quilt [Fish 88, Leland 88]). Il permet en particulier de travailler avec des versions brouillons qui sont améliorées avant d'être livrées aux autres participants.

(18) *Le mode asynchrone favorise le caractère privé du travail de chacun.*

Certains systèmes considèrent plusieurs modes comme par exemple CES [Greif 92] qui propose un mode purement asynchrone, ainsi qu'un mode qui autorise un seul participant à modifier le document tout en permettant aux autres participants de voir les modifications en temps-réel.

## 4.6 Cohérence des données

Que les données soient dupliquées ou non, garantir la cohérence des données partagées est important. En effet, des accès concurrents aux données partagées sont possibles et par ce biais peuvent entraîner des incohérences. Par exemple, deux opérations modifiant une même portion de texte peuvent être entrelacées, produisant un résultat non souhaité. Lorsque la duplication est mise en œuvre, à ce problème s'ajoute celui de la cohérence mutuelle des copies. Plusieurs copies peuvent effectuer des opérations dans des ordres différents ce qui entraîne une incohérence de l'ensemble des copies.

Différents critères de cohérence sont définis dans la littérature : le chapitre 3 est consacré à ce sujet. Pour l'instant retenons que ces critères offrent des cohérences plus ou moins fortes au sens des contraintes de sérialisation des opérations.

Garantir la cohérence est important. Toutefois, maintenir une cohérence forte sur les données, obtenir la dernière mise à jour, ou suivre les modifications en temps-réel n'est pas toujours indispensable. Par exemple, une version obsolète ou incohérente peut s'avérer suffisante lorsqu'il s'agit uniquement d'avoir un aperçu de son contenu, voire seulement son

plan. Il est donc utile d'offrir aux utilisateurs plusieurs qualités de service afin de leur laisser le choix entre une réponse rapide avec un résultat éventuellement obsolète, ou une réponse plus lente avec une absolue certitude d'obtenir la dernière version. L'utilisation de caches locaux permet par exemple de diminuer les temps de latence pour certaines opérations, au détriment de la cohérence (e.g., SharedBook [Lewis 88], Mace [Newman-Wolfe 91a]).

(19) *Plus la cohérence souhaitée est forte, plus le système sera mis à contribution.*

# 5 Conclusion

Ce chapitre avait pour but d'introduire de façon informelle le concept d'édition coopérative. Pour ce faire, nous avons présenté différentes activités regroupées sous ce vocable. Nous avons également identifié des comportements et des méthodes de travail. Cette étude nous a permis de définir un certain nombre de besoins.

Prenant en considération les contraintes matérielles et la qualité de service que doit fournir un environnement d'édition coopérative, nous avons formulé un certain nombre de remarques mettant en jeu les besoins d'un côté et les contraintes matérielles et logicielles de l'autre. Ces remarques seront utilisées pour justifier les choix de conception de notre plate-forme d'édition coopérative.

# Chapitre 2

# Environnement d'édition coopérative

## 1 Introduction

La section précédente a présenté les caractères généraux de l'édition coopérative. Nous complétons cette présentation par la description des composants d'un environnement d'édition coopérative. Il se compose non seulement d'un éditeur, mais encore d'outils de communication qui assurent la coordination des personnes participant à la collaboration. La section 2 décrit les différentes approches proposées dans la littérature en ce qui concerne l'éditeur. Finalement, les aspects liés à la communication sont abordés dans la section 3.

## 2 L'éditeur

L'éditeur est le composant autour duquel s'articule la collaboration. Il doit inclure en plus des fonctionnalités classiques d'un éditeur mono-utilisateur, les mécanismes permettant le partage du document. Ce partage implique en particulier le contrôle des accès aux données partagées; deux points sont à considérer. Le premier concerne la structuration du travail collaboratif, c'est-à-dire le moyen de coordonner les actions individuelles des participants afin de faciliter la réalisation de la tâche commune. Le deuxième point concerne les mécanismes permettant aux utilisateurs de contrôler explicitement l'entrelacement des actions effectuées sur le document. Finalement, nous terminons par une étude des facilités de révision offertes par les éditeurs coopératifs. Cette phase de révision est une phase importante dans le cadre de l'édition coopérative, puisqu'elle est réalisée généralement par des personnes extérieures au cercle des auteurs.

## 2.1 Structuration du travail collaboratif

Trois approches ont été considérées dans la littérature [Miles 93]. Les deux premières contrôlent la coopération en imposant des règles aux utilisateurs; elles se basent respectivement sur la structure du document et sur la notion de rôles (elles peuvent être conjointement utilisées). La troisième au contraire n'impose rien, les utilisateurs sont ainsi libres de définir leurs propres règles.

### Utilisation de la structure du document

CES [Greif 92], MultimETH [Lubich 90], Iris [Borghoff 93] et Griffon [Decouchant 93] considèrent une décomposition hiérarchique du document (Cf. Chapitre 7). La décomposition hiérarchique est généralement automatiquement réalisée par le système en fonction de certaines règles préétablies, et peut suivre par exemple un standard de modélisation de documents tel que ODA [ODA 88] ou SGML [SGML 86] (e.g., MultimETH et Iris sont basés sur ODA et Griffon sur SGML).

Cette hiérarchie est utilisée pour définir les parties utilisées par les participants et le cas échéant empêcher des accès simultanés aux parties d'un document par plusieurs utilisateurs. Par exemple dans MultimETH [Lubich 90], pendant qu'un utilisateur $U_i$ modifie la sous-section 2.1, il n'est pas possible qu'un autre utilisateur $U_j$ ne modifie ni la section 2, ni la section 2.1, ni la sous-sous-section 2.1.1. Cette approche autorise un contrôle de l'accès aux parties d'un document qui respecte la sémantique propre du document, c'est-à-dire la logique associée à son plan.

> (20) *Utiliser la structure naturelle du document offre un mode de fonctionnement naturel facile à appréhender par l'utilisateur.*

### Utilisation de rôles

L'utilisation de rôles est également un moyen d'organiser la collaboration. Plusieurs environnements l'utilisent, mais de façons sensiblement différentes.

Quilt [Fish 88, Leland 88] considère que chacun des participants a un rôle auteur principal, coauteur, annotateur, éditeur ou lecteur (i.e., *author*, *co-author*, *commentor*, *editor*, *reader*). À chaque partie du document est associé un mode d'accès parmi exclusif, partagé et édition (i.e., *exclusive*, *shared*, *editor*). La combinaison rôle/mode d'accès définit pour chaque participant les opérations qu'il peut effectuer sur le document. Par exemple, une partie en mode exclusif ne peut être modifiée que par l'auteur principal alors qu'une partie en mode partagé autorise en plus les coauteurs à y apporter des modifications. Le mode édition oblige

les participants à soumettre préalablement leurs propositions à un éditeur qui est le seul autorisé à apporter des modifications au texte. Dans le cas de Quilt, ces rôles et modes d'accès doivent être explicitement définis au début de la collaboration ce qui est une sérieuse limitation. En effet, il est difficile de prévoir le rôle de chacun et d'autre part ce rôle peut évoluer en fonction de divers paramètres : maturité du document (une même personne peut avoir des rôles différents à plusieurs époques de la collaboration [Greif 86]), disponibilité des participants (changements occasionnés par des activités en dehors de la collaboration).

(21) *Il est souhaitable que la définition de rôles et des droits d'accès soit un processus dynamique.*

SharedBook [Lewis 88] propose une liste d'accès pour chacune des parties du document accessibles séparément (ces parties sont appelées *entrées* par les auteurs). Pour chaque entrée, sont précisés l'accès autorisé par défaut ainsi que l'accès maximum que l'on peut obtenir. Les rôles suivants sont définis : éditeur (*production editor*) qui est autorisé à modifier la liste d'accès, auteur qui est autorisé à éditer la portion de texte correspondante, et finalement lecteur qui n'est autorisé qu'à lire. Bien que la définition des entrées soit statique, la modification de la liste d'accès est dynamique.

Un mode particulier de GROVE [Ellis 91] permet de délimiter des portions de texte : il devient ainsi possible de leur attribuer individuellement des modes d'accès public, partagé, privé. Des rôles ne sont pas explicitement définis : le mode public laisse le libre accès à tous les participants à la collaboration, le mode partagé n'accorde le droit d'écriture qu'à un nombre restreint de participants (explicitement nommés dans une liste), et le mode privé ne permet l'accès qu'au participant qui a demandé ce mode. La définition de la portion de texte ainsi que les modes accès est totalement dynamique.

MultimETH [Lubich 90] permet de définir sous forme d'attributs les accès autorisés pour chacune des parties du document. Les droits d'accès sont lire, écrire, annoter, détruire, et sont définis pour le propriétaire, la conférence ou tous. Seul le propriétaire et le superviseur (*chairman*) peuvent modifier les droits. On peut faire une analogie avec le système d'exploitation Unix, ainsi conférence désigne un groupe d'utilisateurs et superviseur la personne responsable du document.

Il est possible par l'attribution de rôles de répartir les tâches et les responsabilités de chacun des participants; le travail n'est ainsi ni négligé ni dupliqué. Il est souhaitable qu'un participant puisse avoir plusieurs rôles sur différentes portions du document.

(22) *L'attribution des rôles doit pouvoir se baser sur la granularité du document.*

### Aucune règle préétablie

Cette dernière méthode n'impose aucune règle, elle autorise tous les participants à accéder librement à tout le document. Elle se base sur le fait qu'un certain nombre de règles sociales émergent automatiquement puisque l'on se trouve dans un processus de collaboration.

ShrEdit [McGuffin 92], GROVE [Ellis 91] (dans le mode par défaut), sont deux éditeurs qui permettent en temps-réel de modifier un document et de voir les modifications des autres participants. Aucun mécanisme de contrôle d'accès n'est prévu, toutefois la mise à jour continue de la part des participants constitue une forme de dialogue implicite. La simple observation naturelle de leur écran permet aux participants d'éviter les conflits d'accès à la même zone de texte. Les personnes impliquées font d'autant plus attention à leur comportement qu'elles sont sensibles au fait qu'une attitude irresponsable de n'importe quel participant peut occasionner un comportement chaotique du groupe et une perte de temps importante pour la collaboration.

> (23) *L'absence de contrainte au niveau du système apporte un renforcement des règles sociales naturelles dans un groupe.*

Les participants organisent leur travail en fonction de conventions implicitement ou explicitement établies au début ou au cours de la collaboration. La manière dont les participants travaillent varie en fonction de la nature du travail, mais également en fonction de la personnalité des participants. Le lecteur intéressé par cet aspect pourra se référer à [Beck 93, Posner 92] qui proposent des études sociologiques qui sortent du cadre de notre travail.

> (24) *En cas d'absence de règles, les participants peuvent suivre leur propre manière de travailler et utiliser divers protocoles sociaux.*

## 2.2 Contrôle de concurrence

Les trois approches visant à structurer le travail coopératif présentées dans la section précédente ne règlent pas à elles seules le problème des accès concurrents aux données partagées. De plus, même si le système garantit la cohérence des données, rien ne garantit que le résultat sera celui escompté par les utilisateurs. Par exemple, si deux écritures sont effectuées de façon concurrente sur la même partie d'un document, le contrôle de la cohérence garantit uniquement que ces deux opérations sont effectuées en accord avec le critère de cohérence considéré. Cependant, il est possible qu'une écriture ait écrasé l'autre et donc qu'une partie du travail soit perdue, ce qui n'est pas satisfaisant du point de vue des utilisateurs.

Des mécanismes de contrôle de concurrence permettant de se prémunir contre cela existent, ils sont classés en deux catégories : mécanismes *optimistes* et mécanismes *pessimistes*. Les

mécanismes optimistes sont les plus permissifs, ils sont basés sur une détection a posteriori des éventuels conflits. Les opérations sont appliquées immédiatement, mais lors d'une détection de conflits, un *retour en arrière* (*roll-back*) peut être nécessaire pour revenir à un état cohérent. Les mécanismes pessimistes en revanche empêchent les conflits a priori, et assurent donc qu'il ne sera jamais nécessaire de défaire une opération. Toutefois, ils réduisent la concurrence et peuvent séquentialiser des opérations qui auraient pu être exécutées de façon concurrente.

Nous présentons ici les différents mécanismes proposés dans la littérature dans le cadre de l'édition coopérative. Pour chacun d'eux, nous citons les environnements qui les utilisent.

### Verrouillage explicite

C'est une méthode pessimiste classique qui se retrouve dans un grand nombre de mises en œuvre; des variantes sont toutefois à noter. Par exemple dans MultimETH [Lubich 90], Iris [Borghoff 93], SharedBook [Lewis 88] et DistEdit [Knister 93] l'utilisateur acquiert ou relâche explicitement un verrou pour une partie de document. L'obtention d'un verrou garantit la possibilité d'écrire de façon exclusive sur la partie de document concernée. Le relâchement du verrou est à l'initiative du possesseur ce qui peut être pénalisant pour les autres utilisateurs en cas de monopolisation du verrou.

> (25) *Un objet peut être verrouillé bien qu'il ne soit pas utilisé, ce qui nuit à la concurrence.*

Afin de remédier à ce problème, CES [Greif 92] propose un mécanisme qui permet à tout utilisateur d'obtenir le verrou après une période d'oisiveté de la part du possesseur courant (*tickle lock*).

Une variante du verrou est le jeton (*floor passing*) où seul le possesseur du jeton est autorisé à effectuer des opérations susceptibles de modifier la donnée partagée. À la différence du verrou, le jeton est transmis d'un utilisateur à l'autre. Le premier prototype de DistEdit [Knister 90] était basé sur ce principe. Un seul utilisateur était autorisé à écrire pendant que les autres observaient les modifications de celui-ci en temps-réel.

### Notification

La notification est une méthode optimiste permettant d'avertir chacun des participants des actions des autres de manière implicite. Il est ainsi possible de détecter et/ou d'éviter les conflits par observation de l'espace de travail.

Ce type de mécanisme est particulièrement efficace pour les activités en temps-réel où la granularité est fine et les mises à jour fréquentes. Dans ce cas, il est facile de détecter

rapidement les conflits, le travail déjà effectué est minime et le remettre en cause ne constitue pas une perte importante. Cette méthode est utilisée dans GROVE [Ellis 91] et ShrEdit [McGuffin 92] qui sont deux éditeurs, dont le grain est le caractère.

(26) *Un mécanisme de contrôle de concurrence fondé sur la notification est efficace si le travail à remettre en cause est minime.*

Lorsque le grain est plus gros, par exemple la ligne ou le bloc de texte, il n'est plus suffisant de signaler le travail effectué, mais il faut également signifier les intentions des utilisateurs. Ainsi, il est possible de détecter plus tôt les conflits et le cas échéant diminuer le travail pouvant être remis en cause.

(27) *Annoncer les intentions de chacun permet d'anticiper sur la détection des conflits.*

Par exemple Mule [Pendergast 90] affiche en différentes couleurs les paragraphes libres, ceux utilisés par l'utilisateur et ceux utilisés par les autres utilisateurs. Il est également suggéré dans une extension de signaler les paragraphes fraîchement ajoutés par une couleur différente afin d'attirer l'attention. Le problème soulevé alors est qu'une abondance d'informations peut conduire à une perturbation plutôt qu'à une aide. Par exemple la visualisation en direct des modifications des autres participants peut entraîner des distractions [Ellis 91]. Pour y remédier, et puisque l'information importante est la localisation et non le contenu, ils proposent de signaler l'emplacement des modifications des autres utilisateurs par un nuage qui grandit proportionnellement à la taille des modifications.

Pour les éditeurs asynchrones utilisant un grain encore plus gros, un tel degré de notification n'est pas possible. Cependant, les parties verrouillées ou utilisées peuvent être signalées (e.g., MultimETH [Lubich 90], Iris [Borghoff 93], SharedBook [Lewis 88]), ainsi que les actions majeures effectuées par les autres participants (e.g., Quilt [Fish 88, Leland 88]).

### Détection de dépendances

Cette méthode optimiste permet de détecter des opérations qui entrent en conflit (e.g. en utilisant un estampillage des opérations). Les conflits sont généralement résolus à la main par les utilisateurs. En cas d'absence de conflits, les opérations sont effectuées immédiatement. Cette méthode optimiste est particulièrement bien adaptée à un travail coopératif où les tâches sont bien définies et où les conflits sont rares. Cette détection de dépendances devient alors un garde-fou contre les erreurs de manipulations.

(28) *La détection de dépendances est efficace si les conflits sont peu fréquents.*

Le principal désavantage de cette méthode est que la correction des erreurs est laissée à l'utilisateur ce qui peut être source d'erreur [1]. Des outils de comparaison de versions peuvent être utilisés afin de mettre en évidence les différences entre les versions. Prep [Neuwirth 90], Quilt [Fish 88, Leland 88], Iris [Borghoff 93] offrent de tels outils.

### Divergence et fusion

Avec cette méthode optimiste [Minor 93], chaque modification génère une nouvelle version et n'importe quelle version peut être utilisée comme base pour de nouvelles modifications. Ces versions sont fusionnées dans un second temps par les utilisateurs. Cette tâche est facilitée par un outil qui permet de faire apparaître sur le même texte les parties communes et les parties propres à chacune des versions. L'utilisateur peut à volonté faire apparaître ou disparaître ces dernières, et constituer ainsi une version finale tenant compte des différentes versions. Celle-ci devient alors la version à partir de laquelle les utilisateurs pourront continuer à travailler. Cette méthode est certes dépendante de l'utilisateur, mais offre suffisamment de souplesse pour être utilisable efficacement.

(29) *La fusion de versions garantit que le travail de chacun est au moins évalué à défaut d'être finalement retenu.*

## 2.3 Outils de révision

La phase de révision est une phase importante dans le processus d'édition. Les annotations permettent d'amender le document initial afin d'en améliorer la qualité. La manière dont ces annotations sont intégrées dans le document sont multiples.

Collaborative Annotator [Koszarek 90] est un outil qui est plus particulièrement prévu pour le travail de révision. Au document initial, il est possible d'ajouter des commentaires sonores, des pense-bêtes (*yellow sticker*), des surcharges; il est également possible de surligner des passages de la même manière qu'avec un marqueur et d'utiliser plusieurs polices de caractères pour les remarques.

Un autre outil, Prep [Neuwirth 90], permet à plusieurs réviseurs d'apporter séparément leurs commentaires. Les réviseurs annotent le document dans la marge prévue à cet effet. Le rédacteur a alors accès pour chaque portion de texte aux différentes annotations visualisées dans des colonnes différentes. Ceci favorise la synthèse et l'intégration des amendements des différents réviseurs.

[1] Errare humanum est

D'autres [Minor 93] proposent pour leur part de créer une version alternative contenant annotations et modifications. Ceci couplé au mécanisme de fusion de versions permet au rédacteur de prendre en compte ou non les remarques.

D'autres systèmes permettent d'attacher des informations ou attributs aux parties du documents, ces informations contiennent notes et remarques de la part des réviseurs (e.g., SharedBook [Lewis 88], Quilt [Fish 88, Leland 88], MultimETH [Lubich 90]).

Il est à remarquer que les éditeurs en temps-réel ne permettent pas ou difficilement d'intégrer des annotations au document. En effet, les changements sont effectués sur le document lui-même et il n'est pas possible de distinguer les modifications du texte original.

# 3 Dialogue entre utilisateurs

Le dialogue entre utilisateurs concerne toutes les interactions entre les participants qui ne sont pas faites au travers de l'éditeur proprement dit, mais par l'intermédiaire d'outils annexes. Deux moyens de communication sont à distinguer par les fonctionnalités offertes :

La *communication de groupe*, qui est un outil de communication permettant aux utilisateurs impliqués dans une même tâche (e.g., la rédaction d'une partie du document) de diffuser une information. Les membres de ce groupe ne sont pas connus individuellement et leur nombre peut varier dynamiquement; ils ne sont donc accessibles que par l'intermédiaire du groupe associé à la tâche commune qui les lie. Bien qu'indispensable, ce type de communication n'est toutefois pas suffisant, car il est impossible de connaître ou de définir le ou les destinataires d'un message.

La *communication directe* dans laquelle le ou les destinataires peuvent être définis explicitement.

Dans un premier temps, nous présentons dans cette section les aspects généraux liés à la communication. Ensuite nous décrivons les méthodes proposées dans la littérature pour la structuration de cette communication.

## 3.1 Généralités sur la communication

Trois points importants sont à considérer pour la communication : le type de liaison (point-à-point, diffusion), le type de donnée (texte, graphique, audio, vidéo, etc.), le type de communication (synchrone[2], asynchrone).

[2] ou temps-réel.

### Type de communication

Plusieurs schémas de communication sont à prendre en compte : d'une part, la communication *point-à-point* faisant intervenir deux utilisateurs et d'autre part la *diffusion* pour des activités de groupe. Les couches définies dans TCP/IP [Comer 88], que ce soit UDP (mode datagramme) ou TCP (mode connecté), ne sont pas bien adaptées aux diffusions. Et bien que des efforts soient faits pour définir une norme en ce domaine [Armstrong 92, Deering 89], il ne peut être question de considérer sa disponibilité à grande échelle dans un avenir proche. Dans la majorité des cas, une couche logicielle spécifique est donc nécessaire pour leur gestion.

(30) *Au niveau protocole de transfert, seules les communications point-à-point sont disponibles en standard.*

### Types d'informations manipulées

Plusieurs types de données sont à considérer : d'une part des données classiques de type texte ou graphique, mais encore des informations de type audio et/ou vidéo. Si les premières ne posent pas de problèmes particuliers, en revanche le transfert de ces dernières nécessitent des protocoles spéciaux. En effet, ils autorisent une perte occasionnelle d'une partie des données transmises, mais requièrent en revanche un haut débit de transmission. Le protocole assurant ceci n'est pas universellement disponible et leur intégration dans des environnements d'édition coopérative ne reposant pas sur des plate-formes matérielles ou logicielles spécifiques est une tâche difficile.

(31) *La diffusion restreinte des protocoles nécessaires à l'acheminement d'information de type audio et/ou vidéo est un frein à leur portabilité.*

### Communication asynchrone et communication temps-réel

L'aspect *asynchrone* pour la communication est très important, car il n'est pas toujours possible pour un utilisateur d'être présent à une discussion. Dans ce cas, il doit être possible à ce dernier d'accéder en différé aux informations échangées au cours de la discussion. Si l'on considère les applications où chacun peut modifier un espace de travail unique et partagé (e.g., Boardnoter [Stefik 87b], Cognoter [Stefik 87b]), il est difficile après coup de déterminer qui est responsable de quoi et quand cela est arrivé. La seule information accessible est l'état courant ou final de l'espace de travail, ce qui dans certains cas peut être insuffisant. En revanche, si l'on considère le concept de *forum* (*bulletin board*), où chacun peut enrichir le débat en postant des messages, l'origine et l'ordre des interventions sont sauvegardés et permettent une reconstitution différée de la discussion.

(32) *Le modèle asynchrone permet par essence de retrouver l'historique d'une discussion.*

La communication en temps-réel est toutefois intéressante, car elle permet à une discussion de se dérouler sans perte de temps et avec une synergie que l'on ne retrouve pas si les utilisateurs travaillent leur intervention avant de la diffuser. De plus, dans une discussion en temps-réel, une seule ligne conductrice émerge alors que dans une communication asynchrone plusieurs apartés différents peuvent être générés à cause du délai induit par des réponses non instantanées. Ce phénomène peut conduire à une perte d'efficacité.

(33) *La communication en temps-réel permet de recréer la spontanéité des réunions de travail.*

## 3.2 Structuration de la communication

Concernant la structuration de la communication, deux alternatives sont à considérer : soit les participants sont encouragés à diriger eux-même la communication (*social protocol*), soit le système se charge du contrôle de celle-ci (*software protocol*) et ceci d'une manière plus ou moins rigide. La rigidité varie selon que la structure est globale (la même pour toute interaction) ou locale (dépendante de l'interaction considérée) [Miles 91]. Les trois cas de figures proposés par la littérature sont les suivants :

### Théorie de la communication (*speech act*)

Certains systèmes (e.g., Coordinator [Winograd 86]) se basent sur la théorie de la communication et offrent des schémas de communication que doivent suivre les participants. Ainsi, chaque interaction est définie par un graphe d'état pour lequel l'issue, à défaut d'être connue à l'avance, est du moins prévisible. Le principal défaut de cette méthode est qu'elle est trop contraignante [Dykstra 91] : ainsi il n'est pas possible d'intégrer un schéma pour lequel il n'y a pas de réponse (simple notification) ou encore pour lequel certaines réponses sont implicites (abstention lors d'un vote). De plus, il n'est pas possible d'omettre certains états intermédiaires lorsque des événements extérieurs sont réalisés. L'emploi de cette méthode est difficile dans le cadre d'une communication de groupe vu la complexité des graphes d'états mis en jeu.

### Messages semi-structurés

L'utilisation des messages semi-structurés [Malone 87] permet de structurer la communication sans être toutefois aussi rigide que la méthode précédente. Un message semi-structuré

est un message pour lequel certains champs sont remplis automatiquement par le système alors que d'autres doivent être remplis par l'utilisateur. Il est à noter que l'utilisateur peut éventuellement modifier les premiers ou laisser vides les seconds. L'intérêt est double : (1) l'émetteur peut se concentrer sur les aspects importants de son message puisque la partie fastidieuse est automatiquement remplie par le système et (2) le destinataire bénéficie d'une plus grande souplesse pour le tri ou le filtrage puisque les champs prédéfinis sont plus facilement interprétables (normalisés et exempts d'erreurs). Les News et le E-Mail sur Internet, LENS [Malone 86], KSM [Yoder 89] et gIBIS [Conklin 88] utilisent des messages semi-structurés.

Cette méthode est cependant mal adaptée aux communications en temps-réel, car le grain élevé inhérent aux messages semi-structurés (le message doit être complètement rempli avant d'être envoyé) ne permet pas de répondre aux besoins de ces communications. En revanche, les messages semi-structurés sont particulièrement bien adaptés à la communication asynchrone puisqu'ils permettent d'intégrer automatiquement des informations aux messages. C'est à dire pour la communication de groupe : le sujet (partie du document concernée), le groupe de destinataire et pour la communication directe : l'heure, le lieu, la référence à d'autres messages, des mots-clés.

#### Absence de structure

L'approche retenue dans ACE [Dykstra 91], Boardnoter [Stefik 87b] et Cognoter [Stefik 87b] diffère grandement des approches précédentes, puisqu'elle part du principe que les utilisateurs sont plus aptes que le système à définir la structure de l'interaction. Les modifications réalisées à l'aide de divers outils apparaissent en temps-réel dans une zone de travail partagée par les utilisateurs.

Toutefois, cette méthode souffre du manque de structure associé à la communication qui, en particulier, ne permet pas à un utilisateur de retrouver de manière asynchrone (en différé) le cheminement de la pensée des autres utilisateurs au cours de la discussion. Elle est donc mieux adaptée à une communication directe en temps-réel qu'à une communication de groupe asynchrone.

## 4 Conclusion

Ce chapitre avait pour but de présenter les composants principaux d'un environnement d'édition coopérative : l'éditeur et le module de communication (communication de groupe et communication directe). L'étude de ces composants nous a permis de décrire les différentes alternatives, plus ou moins rigides, permettant de structurer la collaboration.

Dans la section consacrée à l'éditeur, ont été présentés d'une part les mécanismes pessimistes et optimistes de contrôle de concurrence et d'autre part les différentes réponses au problème de la révision des documents que l'on peut trouver dans la littérature.

Concernant la partie communication, l'étude des aspects temps-réel et asynchrone laisse apparaître que chacun des deux modes répond à des besoins différents et complémentaires.

# Partie II

# Cohérence

# Chapitre 3

# Critères de cohérence : définitions

## 1 Introduction

La cohérence des données partagées peut être violée lorsque plusieurs utilisateurs les modifient concurremment. Ce problème n'est pas uniquement lié à l'édition coopérative, mais plus généralement aux applications distribuées. Lorsque les données sont dupliquées, à des fins de tolérance aux fautes par exemple, le problème se complique. Si une donnée $D$ existe en plusieurs exemplaires dans le but de résister aux défaillances, assurer la cohérence de chacun des duplicas $d_i$ ne suffit pas à garantir que l'ensemble des duplicas sera lui-même cohérent. En effet, les opérations peuvent avoir été appliquées dans des ordres différents sur les duplicas $d_i$ avec pour résultat que la cohérence mutuelle de l'ensemble n'est plus respectée. Ce problème suscite un grand intérêt et nombre de contributions ont été faites dans ce domaine [Sheth 90, Bernstein 81, ElAbaddi 89].

Informellement parlant, un critère de cohérence impose un ordre partiel[1] sur les événements du système. Cet ordre représente les contraintes d'ordonnancement que doit respecter le système afin de garantir le critère considéré, en d'autres termes une violation de cet ordre entraîne une violation du critère de cohérence.

Ce chapitre a pour but de présenter les contributions étant ou pouvant être utilisées dans le cadre de l'édition coopérative. La section 2 décrit le modèle que nous utilisons pour définir les différents critères de cohérence que nous avons retenus. Ces critères sont la cohérence séquentielle et la linéarisabilité; ils sont définis formellement dans la section 3. La section 4 présente les différentes mises en oeuvre proposées dans la littérature pour garantir les critères précédemment cités dans le cadre de l'édition coopérative.

[1] L'ordre total étant un cas particulier d'ordre partiel.

# 2 Modèle

Le système consiste en un ensemble de $n$ processus $\{P_1, ..., P_n\}$ accédant de manière concurrente à des objets représentant les données partagées. Les objets ne sont accessibles qu'au travers d'opérations prédéfinies qui leur sont propres. Une opération $O_i$ réalisée par le processus $P_i$ sur l'objet $X$ sera noté $O_i(X, A)R$ où $A$ est l'argument de l'opération et $R$ le résultat de l'opération retourné par l'objet[2]. L'opération $O_i(X, A)R$ se décompose en deux événements : une *invocation*, notée $INV_i(O_i(X, A))$, émise par le processus $P_i$ et destinée à l'objet $X$ et la *réponse* correspondante (même opération, même objet), notée $RES_i(O_i(X)R)$, émise par l'objet à destination du processus requérant. La figure 3.1 présente les deux manières de modéliser les opérations effectuées sur l'objet $X$ par un processus $P_i$. L'objet $X$ est de type registre, et $Ecr()$ et $Lec()$ sont les opérations d'écriture et de lecture définies sur cet objet. Le temps s'écoule de la gauche vers la droite. Dans la figure 3.1-a, les opérations mettent en œuvre des communications, symbolisées par des flèches, entre le processus $P_i$ et l'objet $X$. Dans la figure 3.1-b, la communication est considérée comme instantanée et une opération apparaît comme un intervalle compris entre l'invocation et la réponse correspondante. Nous utiliserons cette deuxième représentation dans la suite de ce chapitre.

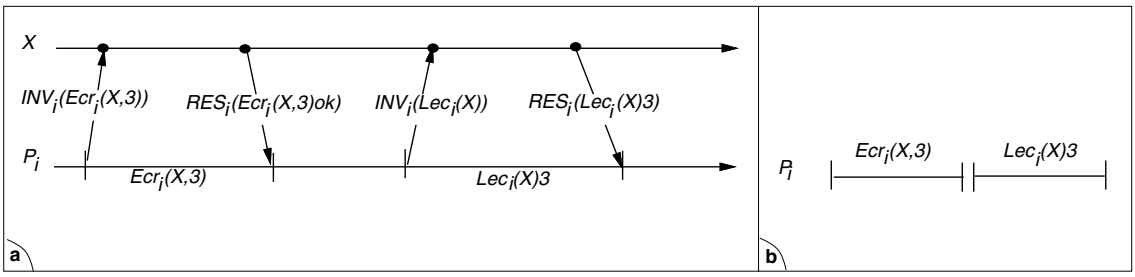


Figure 3.1 : Schéma d'une opération

Un objet dupliqué $X$ est représenté dans le système par un ensemble de *duplicas* $\{x^1, ..., x^m\}$. Nous distinguerons les deux niveaux d'abstraction en appelant *objet logique* l'objet dupliqué $X$, et *objet physique* un duplica $x^i$. Dans un contexte non dupliqué, l'objet logique et l'objet physique sont confondus.

Les opérations réalisables sur les duplicas $x^i$ sont les mêmes que celles définies sur $X$. En d'autres termes, une opération logique $O_i(X, A)R$ peut être décomposée en plusieurs opérations physiques appliquées sur les objets physiques $\{o_i^1(x^1, a^1)r^1, ..., o_i^n(x^n, a^n)r^n\}$. Les arguments $a^j$ seront identiques puisque c'est la même opération que l'on veut appliquer à chacun des duplicas; en revanche les résultats $r^j$ peuvent être différents puisqu'ils peuvent être dépendants de l'état de l'objet physique au moment de l'opération. Au niveau

[2]Dans la suite de ce chapitre, lorsque certaines de ces informations ne sont pas indispensables, elles seront omises afin de simplifier les notations.

des réponses, on considère que la réponse d'une opération logique est construite à partir des réponses des opérations physiques. Dans la suite de ce chapitre, les définitions feront référence à des objets logiques.

Une invocation est dite *en attente* tant que la réponse n'a pas été reçue par le processus invoquant, et *achevée* dans le cas contraire. Pour un objet, une opération est considérée comme prenant effet instantanément à un instant compris entre la réception de l'invocation et l'envoi de la réponse correspondante. Dans la suite de ce chapitre nous ne considérons que des opérations achevées.

L'ensemble partiellement ordonné défini par un ensemble d'opérations $\mathcal{O}$ muni d'une relation d'ordre partiel[3] $\stackrel{R}{\rightarrow}$ est appelé une *histoire* $H$, notée $(\mathcal{O}, \stackrel{R}{\rightarrow})$. Lorsque la relation d'ordre $\stackrel{R}{\rightarrow}$ définit un ordre total sur l'ensemble $\mathcal{O}$, l'histoire $(\mathcal{O}, \stackrel{R}{\rightarrow})$ est dite *séquentielle*. Considérant une histoire $H$, on peut restreindre l'ensemble des opérations considérées à un processus $P_i$, ou à un objet $X$ particulier. Soit $H|_{P_i}$ la *sous-histoire* restreinte aux opérations du processus $P_i$ et $H|_X$ la sous-histoire restreinte aux opérations réalisées sur l'objet $X$. $H|_{P_i}$ est également noté $(\mathcal{O}_{P_i}, \stackrel{R}{\rightarrow}_{H|_{P_i}})$ où $\mathcal{O}_{P_i}$ est la restriction au processus $P_i$ des opérations considérées dans $H$, et $\stackrel{R}{\rightarrow}_{H|_{P_i}}$ la restriction de la relation d'ordre $\stackrel{R}{\rightarrow}$ au sous-ensemble $\mathcal{O}_{P_i}$ de $\mathcal{O}$. De la même manière, $H|_X$ est également noté $(\mathcal{O}_X, \stackrel{R}{\rightarrow}_{H|_X})$, où $\mathcal{O}_X$ est la restriction à l'objet $X$ des opérations considérées dans $H$, et $\stackrel{R}{\rightarrow}_{H|_X}$ la restriction de la relation d'ordre $\stackrel{R}{\rightarrow}$ au sous-ensemble $\mathcal{O}_X$ de $\mathcal{O}$.

Une exécution d'un ensemble de processus $\{P_1, ..., P_n\}$ est une collection d'histoires $H|_{P_i}$ définissant les actions de chacun des processus $P_i$. Cette exécution définit un ensemble totalement ordonné d'invocations et de réponses de ces processus. Cet ordre total correspond à l'observation en temps-réel du système par un observateur externe idéal.

Il est possible à partir de cet ordre total de définir la relation d'ordre partiel $\stackrel{TR}{\rightarrow}$ sur l'ensemble des opérations $\mathcal{O}$ exécutées, avec la définition suivante[4] :

> Définition 3.1 : *ordre* $\stackrel{TR}{\rightarrow}$
> *Soient $O_i$ et $O_j$ deux opérations exécutées respectivement par les processus $P_i$ et $P_j$. Si, considérant l'ordre temps-réel dans lequel les événements du système se déroulent, la réponse d'une opération $RES_i(O_i)$ précède l'invocation d'une opération $INV_j(O_j)$, alors $O_i \stackrel{TR}{\rightarrow} O_j$.*

Dans un système fortement couplé, comme par exemple une machine parallèle, il est possible de mettre en œuvre une observation en temps-réel. En revanche pour un système faiblement couplé (e.g. un système distribué), a fortiori à grande échelle, la notion de temps-réel n'a

[3] Nous définirons par la suite ce que représente cette relation d'ordre.
[4] TR pour "Temps-Réel".

pas de sens, compte tenu de l'absence d'une horloge globale commune à tout le système distribué. De plus, sur le plan humain, cette notion d'ordre total respectant le temps-réel n'est pas appréhendable. En effet, deux personnes distantes ne sont pas capables de déterminer un ordre total de leurs actions, à moins qu'elles ne soient en communication. La notion de communication entre ces deux protagonistes implique une dépendance causale si la communication a une répercussion sur les actions futures de ces derniers. Ceci nous amène à considérer un ordre partiel qui tient compte de ces dépendances causales entre les événements du système. Ces dépendances causales peuvent avoir deux origines.

La première origine est due aux interactions directes entre processus à l'aide de messages, c'est la relation *happened before* (notée $<_{hb}$) définie par Lamport [Lamport 78] de la manière suivante :
(1) si deux événements $E1_i$ et $E2_i$ ont lieu sur un même processus $P_i$ dans l'ordre $E1_i$ puis $E2_i$ alors $E1_i <_{hb} E2_i$;
(2) si deux événements $E_i$ et $E_j$ ont lieu sur deux processus $P_i$ et $P_j$ et que $E_i$ est l'émission par $P_i$ d'un message $M$ et que $E_j$ est la réception de $M$ par $P_j$ alors $E_i <_{hb} E_j$;
(3) par transitivité, si pour trois événements $E1$, $E2$ et $E3$ nous avons $E1 <_{hb} E2$ et $E2 <_{hb} E3$ alors $E1 <_{hb} E3$.

A partir de la relation d'ordre partiel $<_{hb}$ sur les événements du système nous construisons la relation d'ordre partiel entre les opérations de la manière suivante[5] :

> Définition 3.2 : *ordre* $\overset{DCs}{\rightarrow}$
> *Si, considérant la relation d'ordre "happened before" sur les événements du système, la réponse d'une opération $RES_i(O_i)$ précède l'invocation d'une opération $INV_j(O_j)$, alors $O_i \overset{DCs}{\rightarrow} O_j$.*

La seconde origine des dépendances causales est due aux opérations invoquées par les processus sur les objets données. Elle découle de l'observation par un processus $P_i$ des actions réalisées par un processus $P_j$ sur les objets. Elle permet de définir une relation d'ordre entre les opérations d'une exécution. L'ordre $\overset{DCd}{\rightarrow}$ est défini comme suit[6] :

> Définition 3.3 : *ordre* $\overset{DCd}{\rightarrow}$
> *(1) si un objet $X$ exécute séquentiellement une opération $O_i$ invoquée par un processus $P_i$ puis une opération $O_j$ invoquée par un processus $P_j$, alors $O_i \overset{DCd}{\rightarrow} O_j$. (2) si un même processus $P_i$ exécute séquentiellement deux opérations $O1_i$ puis $O2_i$ alors $O1_i \overset{DCd}{\rightarrow} O2_i$. (3) soient trois opérations $O1$, $O2$ et $O3$ telles que $O1 \overset{DCd}{\rightarrow} O2$ et $O2 \overset{DCd}{\rightarrow} O3$, alors $O1 \overset{DCd}{\rightarrow} O3$.*

[5] DCs pour "Dépendance Causale système" : i.e., entre processus.
[6] DCd pour "Dépendance Causale sur les données".

La figure 3.2 illustre les trois relations d'ordre considérées ($\overset{TR}{\rightarrow}, \overset{DCs}{\rightarrow}, \overset{DCd}{\rightarrow}$). Soit $X, Y, Z$ des objets de type *registre* et $Ecr()$ et $Lec()$ les opérations d'écriture et de lecture sur ces objets. Nous matérialisons par une flèche oblique une communication entre les processus $P_j$ et $P_i$.

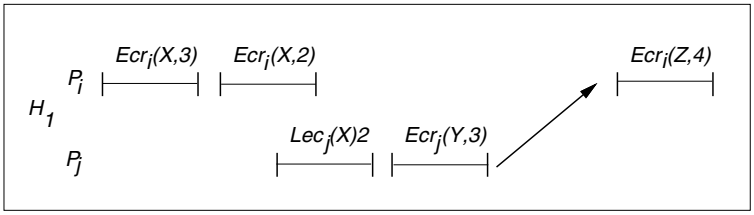


Figure 3.2 : Ordres sur les opérations

Nous avons $Ecr_i(X,2) \overset{TR}{\rightarrow} Ecr_j(Y,3)$ car la réponse correspondant à l'opération $Ecr_i(X,2)$ précède l'invocation de l'opération $Ecr_j(Y,3)$ lorsque le temps-réel est considéré. A cause du message entre $P_j$ et $P_i$ nous avons $Ecr_j(Y,3) \overset{DCs}{\rightarrow} Ecr_j(Z,4)$. Finalement l'objet $X$ réalise d'abord l'écriture de 2 par $P_i$ et ensuite la lecture de cette valeur par $P_j$. Les deux opérations sont donc ordonnées par $\overset{DCd}{\rightarrow}$ de la manière suivante : $Ecr_i(X,2) \overset{DCd}{\rightarrow} Lec_j(X)2$. Nous définissons $\overset{DC}{\rightarrow}$ comme étant la réunion de $\overset{DCs}{\rightarrow}$ et $\overset{DCd}{\rightarrow}$.

Définition 3.4 : *ordre* $\overset{DC}{\rightarrow}$
*(1) si deux opérations* $O1$ *et* $O2$ *sont telles que* $O1 \overset{DCs}{\rightarrow} O2$ *alors nous avons également* $O1 \overset{DC}{\rightarrow} O2$. *(2) si deux opérations* $O1$ *et* $O2$ *sont telles que* $O1 \overset{DCd}{\rightarrow} O2$ *alors nous avons également* $O1 \overset{DC}{\rightarrow} O2$. *(3) soit trois opérations* $O1$, $O2$ *et* $O3$ *tel que* $O1 \overset{DC}{\rightarrow} O2$ *et* $O2 \overset{DC}{\rightarrow} O3$ *alors par transitivité* $O1 \overset{DC}{\rightarrow} O3$

Nous allons illustrer l'intérêt de considérer l'ordre $\overset{DC}{\rightarrow}$ imposé par les dépendances causales plutôt que $\overset{TR}{\rightarrow}$ imposé par le temps-réel sur un exemple concret dans le contexte de l'édition coopérative. Soient deux utilisateurs $U_i$ et $U_j$ travaillant sur le même document ($P_i$ et $P_j$ sont les processus modélisant leurs actions respectives sur le document). L'interaction normale entre $P_i$ et $P_j$ est réalisée par les accès successifs au document partagé. Supposons que $U_i$ écrive une nouvelle version du document pendant que $U_j$ désire lire le document. Deux scénarii sont possibles : (1) $U_i$ écrit et $U_j$ lit ce que $U_i$ a écrit, ou bien (2) $U_j$ lit l'ancienne version et $U_i$ en écrit une nouvelle. Si l'on considère qu'après avoir écrit la nouvelle version, $U_i$ envoie un message électronique (*e-mail*) à $U_j$, et que $U_j$ prenne connaissance de ce message avant de lire le document, alors seul le premier scénario est possible. Bien que la communication par *e-mail* soit extérieure aux processus $P_i$ et $P_j$, son existence invalide un des scénarii qui pourrait conduire à une incohérence, si par exemple le message en question fait référence aux modifications faites par $U_i$.

Plus formellement, l'important dans une exécution est d'exhiber l'ordre concernant les opéra-

tions qui sont causalement dépendantes, et non pas un ordre total sur toutes les opérations. En effet, peu importe l'ordre d'exécution de deux opérations indépendantes, puisque les effets sont identiques. C'est en ce sens que l'ordre impliqué par le temps-réel est trop contraignant. En revanche, si une dépendance *DCs* impose un ordre entre deux opérations, cet ordre peut influencer le résultat de l'exécution. C'est pour cela que ne considérer que les dépendances *DCd* est insuffisant. L'ordre construit à partir des dépendances *DCs* et *DCd* est un compromis acceptable dans le cadre des systèmes distribués (cette affirmation sera étayée dans la suite de ce chapitre).

La suite de cette section a pour but de rappeler un certain nombre de définitions visant à simplifier la formulation des critères de cohérence. Ces définitions sont pour la plupart inspirées de celles de [Herlihy 90, Attiya 91].

Nous considérons dans la suite de ce chapitre que les histoires de chacun des processus $P_i$ (i.e., les $H|_{P_i}$) sont séquentielles. Il est à remarquer que cela n'implique en rien que les histoires restreintes aux objets (i.e., les $H|_X$) soient séquentielles puisque les objets peuvent traiter de manière concurrente des requêtes en provenance de plusieurs processus.

Considérons un *environnement séquentiel* où toutes les opérations sont effectuées séquentiellement sur une machine unique. Pour chaque objet, il est possible de définir, dans cet environnement, une *spécification séquentielle*

> Définition 3.5 : *spécification séquentielle*
> *La spécification séquentielle d'un objet modélise les comportements acceptables d'un objet dans un environnement séquentiel.*

Par exemple, considérons un objet $X$ de type *registre* soumis à des requêtes séquentielles (une seule requête est traitée à la fois). Un comportement non acceptable serait qu'une écriture de $a$ dans $X$ précède immédiatement une lecture $X$ produisant la valeur $b \neq a$. Ce cas de figure ne se conforme pas à l'axiomatisation définie pour les objets de types *registres* [Misra 89].

La spécification séquentielle est définie en fonction du type de l'objet, ce qui explique qu'il ne peut être génériquement défini, et qu'une définition spécifique à chaque type d'objet doit être considérée. La spécification séquentielle peut être décrite, soit sous la forme d'axiomes [Misra 89, Herlihy 90] soit sous la forme d'une machine à états [Weihl 89].

Nous présentons ci-dessous un exemple de spécification séquentielle appliquée à un objet de type *pile* en utilisant l'axiomatisation ($q$ et $q'$ représentent ci-dessous respectivement l'état de la pile $Q$ avant et après la réalisation de l'opération.) Pour qu'une opération puisse être réalisée, il faut que la précondition soit valide et qu'une fois l'opération achevée, la postcondition soit réalisée.

Axiome 1 : *(Empiler)*
précondition : *VRAIE*
*Empiler(Q,a)Ok*
postcondition : *q' = insérer(q,a)*

Axiome 2 : *(Dépiler)*
précondition : *q n'est pas vide*
*Dépiler(Q)b*
postcondition : *q' = reste(q) et b = sommet(q)*

Si une histoire $H|_X$ de l'objet $X$ est séquentielle et conforme à la spécification séquentielle de l'objet $X$, alors elle est dite histoire séquentielle *légale*. Par extension, il est possible de définir la légalité pour une histoire séquentielle $H$ quelconque.

Définition 3.6 : *Histoire légale [Weihl 89, Herlihy 90]*
*Une histoire $H$ est légale si pour tout objet $X$, l'histoire $H|_X$ relative à l'objet $X$ est légale.*

La formalisation des critères de cohérence fait généralement appel à une exécution de référence possédant les propriétés requises pour le critère de cohérence. L'exécution que l'on veut étudier est alors comparée à cette exécution de référence. Pour ce faire, la notion d'*égalité* et d'*équivalence* d'histoires est définie ci-dessous :

Définition 3.7 : *Égalité de deux histoires*
*Deux histoires $H$ et $H'$ sont dites* égales, *(notée $H = H'$) si et seulement si les ensembles d'événements sont les mêmes, et si les relations d'ordre sur les opérations des deux histoires sont identiques.*

Définition 3.8 : *Équivalence de deux histoires [Herlihy 90]*
*Deux histoires $H$ et $H'$ sont dites* équivalentes, *(notée $H \sim H'$) si et seulement si pour tout processus $P$, $H|_P = H'|_P$.*

Les trois histoires de la figure 3.3 sont équivalentes puisque l'ordre des opérations de chacun des processus est préservé. Notons au passage l'équivalence entre l'histoire concurrente $H_2$ et l'histoire séquentielle $H_3$. Cela signifie que dans l'histoire $H_2$ tout s'est passé comme si les opérations étaient appliquées séquentiellement bien que ce ne soit pas le cas puisque en particulier l'écriture de 3 dans $X$ par $P_i$ et la lecture de cette même valeur par $P_j$ se chevauchent. Ce point sera développé dans la section qui suit, où nous nous attacherons à définir un sens à cette équivalence entre histoires.

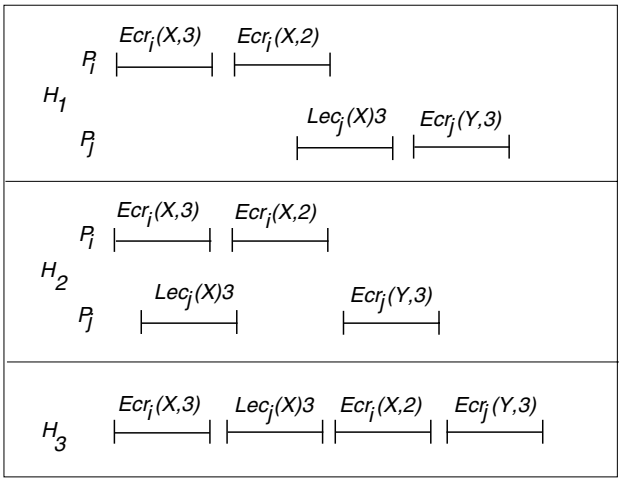


Figure 3.3 : Histoires équivalentes

# 3 Critères de cohérences

Un critère de cohérence pour une histoire $H$ peut être vu comme un ensemble de propriétés que doit respecter l'histoire $H$. Les critères de cohérence sont classés en deux catégories, les critères de cohérence *forts* et les *faibles*. Cette dichotomie se base sur une équivalence ou non de l'histoire $H$ que l'on considère, à une histoire séquentielle $S$. La non équivalence à une histoire séquentielle entraîne la "faiblesse" du critère.

Une propriété intéressante pour un critère de cohérence est la propriété de localité définie ci-dessous :

> Définition 3.9 : *Localité d'un critère de cohérence [Herlihy 90]*
> *Un critère de cohérence $\mathcal{C}$ satisfait à la propriété de localité si pour chaque histoire $H$, et chaque objet $X$, $H|_X$ satisfait $\mathcal{C}$ alors $H$ satisfait $\mathcal{C}$.*

La propriété de localité ne signifie pas que les informations qui permettent d'assurer le critère de cohérence doivent être locales[7], mais elle signifie qu'en assurant le critère de cohérence sur chacun des objets alors le même critère de cohérence est garanti sur l'ensemble du système. Nous allons présenter dans cette section trois critères de cohérence forts : la cohérence séquentielle, la linéarisabilité et finalement la *NTR-linéarisabilité*. Les deux premiers critères sont issus de la littérature, le dernier défini dans le cadre de cette thèse, est une restriction de la linéarisabilité visant à affaiblir les contraintes d'ordonnancement de la linéarisabilité.

[7] Par exemple, comme nous le verrons par la suite, la linéarisabilité se base sur le temps-réel qui est une information globale.

## 3.1 Cohérence séquentielle

La cohérence séquentielle a été introduite par Lamport, initialement dans le cadre des machines parallèles à mémoire distribuée. Cependant, ce critère est également employé dans les systèmes répartis, souvent sous la dénomination de *cohérence forte* ce qui prête souvent à confusion.

La définition informelle donnée par Lamport est la suivante :

> Définition 3.10 : *Cohérence séquentielle [Lamport 79]*
> *Une exécution est séquentiellement cohérente si son résultat est le même que si les opérations de tous les processus avaient été exécutés dans un ordre séquentiel quelconque, et que les opérations de chacun des processus apparaissent dans cette séquence dans l'ordre spécifié par son programme.*

Cette définition informelle reste incomplète par certains points. Par exemple le terme "résultat" n'est pas défini et peut signifier le résultat final, tous les résultats intermédiaires ou encore les résultats intermédiaires qui sont observés. De même, il n'est pas clair que l'ordre des opérations spécifié par le programme doit être respecté, ou si des permutations sont autorisées pour peu qu'elles ne changent pas ce résultat.

Suite à l'article de Lamport, un grand nombre de définitions ont été proposées, qui peuvent être classées en deux catégories.

La première ne considère qu'un seul type d'objet, les registres, mais prend en considération les propriétés du modèle sous-jacent. Ces définitions [Dubois 86, Scheuring 87, Afek 89, Adve 90a, Adve 90b, Gharachorloo 90, Raynal 93] sont plus particulièrement destinées au contexte de la cohérence de cache pour les machines parallèles à mémoire partagée.

La deuxième catégorie [Weihl 89, Herlihy 90, Mavronicolas 91, Attiya 91] propose une approche différente : elle considère un modèle plus général adapté au contexte des systèmes répartis. En particulier, le type de l'objet et les spécificités qui en découlent sont pris en considération. Cette approche permet de formaliser la définition de Lamport.

> Définition 3.11 : *Cohérence séquentielle [Attiya 91]*
> *L'histoire $H$ est séquentiellement cohérente s'il existe une histoire séquentielle $S$ équivalente à $H$, telle que $S$ soit légale.*

Dans cette définition, c'est la propriété de légalité qui assure la cohérence séquentielle. Les exemples des figures 3.4 et 3.5 montrent quatre exécutions mettant en jeu un objet $X$ de type registre et deux opérations : écriture $Ecr()$ et lecture $Lec()$.

Les deux histoires $H_1$ et $H_2$ de la figure 3.4 ne diffèrent que par les valeurs retournées par les

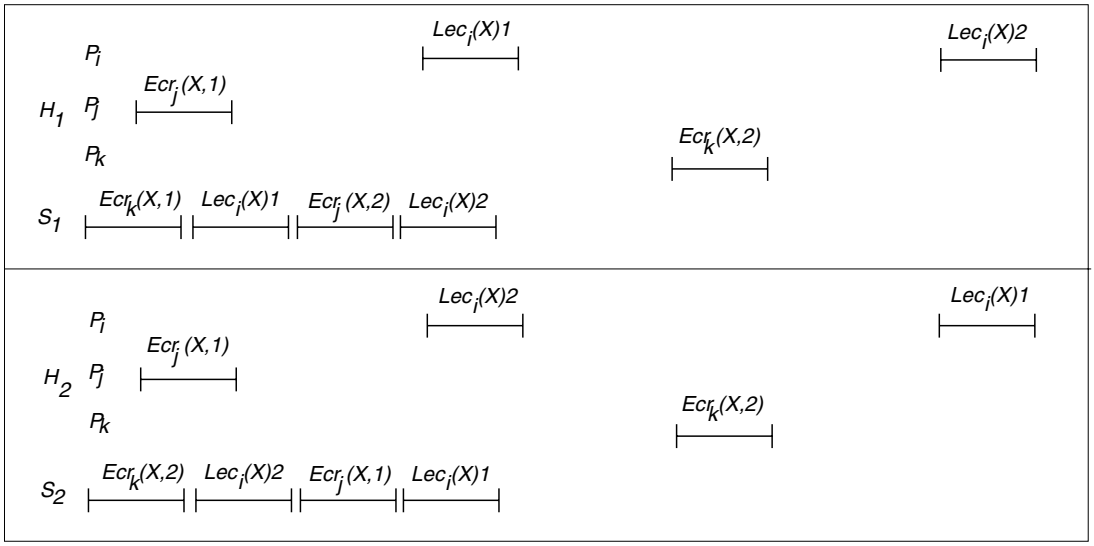


Figure 3.4 : Histoires séquentiellement cohérentes

opérations de lecture effectuées par le processus $P_i$. Dans $H_1$ la première valeur est 1 et la seconde est 2, alors que l'ordre est inversé pour $H_2$. L'équivalence à une histoire séquentielle légale ne fait intervenir que les dépendances sur les objets (ordre partiel $DCd$ entre les opérations) ce qui implique en particulier que les opérations $Ecr_j(X,1)$ et $Ecr_k(X,2)$ des histoires $H_1$ et $H_2$ sont concurrentes, et donc peuvent être réalisées dans n'importe quel ordre. En particulier, $Ecr_j(X,1)$ peut effectivement prendre effet après $Ecr_k(X,2)$ bien que l'opération ait démarré avant (si l'on considère le temps-réel). Les histoires $H_1$ et $H_2$ de la figure 3.4 respectent donc la cohérence séquentielle puisqu'elles sont équivalentes aux histoires séquentielles $S_1$, resp. $S_2$, qui sont séquentielles et légales.

L'histoire $H_3$ de la figure 3.5 correspond à l'histoire $H_1$ de la figure 3.4 dans laquelle nous avons ajouté trois couples d'opérations sur les objets de type registre $Y$, $Z$ et $T$ : $\{Ecr_j(Y,1), Lec_i(Y)1\}$, $\{Ecr_i(Z,1), Lec_k(Z)1\}$ et $\{Ecr_k(T,1), Lec_i(T)1\}$.
Les opérations sur le registre $Y$ définissent une dépendance de type $DCd$ qui ordonne $Ecr_j(X,1) \overset{DCd}{\rightarrow} Lec_i(X)1$. Les opérations sur le registre $Z$ définissent une autre dépendance $DCd$ qui ordonne $Lec_i(X)1 \overset{DCd}{\rightarrow} Ecr_k(X,2)$. Finalement, les opérations sur le registre $T$ définissent une troisième dépendance $DCd$ qui ordonne $Ecr_k(X,2) \overset{DCd}{\rightarrow} Lec_i(X)2$. L'histoire $H_4$ de la figure 3.5 correspond à l'histoire $H_2$ de la figure 3.4 dans laquelle nous avons également défini des dépendances de type $DCd$ similaires.

L'histoire $H_3$ est séquentiellement cohérente puisqu'elle est équivalente à $S_3$. En revanche pour l'histoire $H_4$ il est impossible de trouver une histoire séquentielle légale équivalente à $H_4$ à cause de la dépendance $Ecr_k(X,2) \overset{DCd}{\rightarrow} Lec_i(X)1$.

La même histoire $H_4$ va servir dans la figure 3.6 à montrer que la cohérence séquentielle ne possède pas la propriété de localité. En effet, $H_4$ n'est pas séquentiellement cohérente

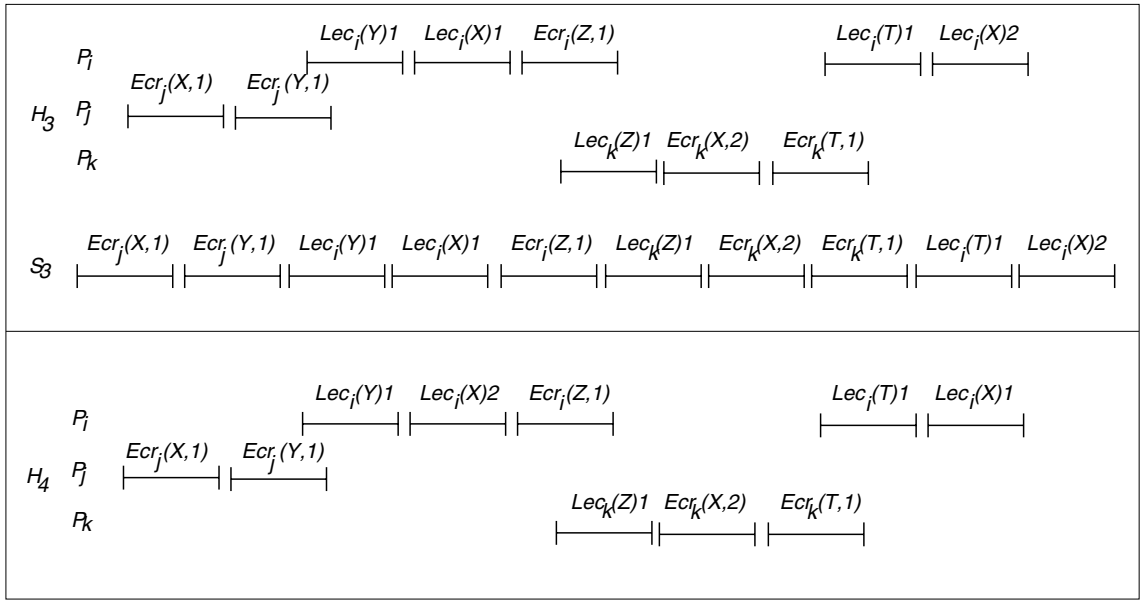


Figure 3.5 : $H_3$ est séquentiellement cohérente, $H_4$ ne l'est pas

et pourtant, les histoires $H|_X$, $H|_Y$, $H|_Z$ et $H|_T$ sont séquentiellement cohérentes puisque équivalentes à des histoires séquentielles légales.

## 3.2 Linéarisabilité

La *linéarisabilité* [Herlihy 90] est définie comme suit :

> Définition 3.12 : *Linéarisabilité [Herlihy 90]*
> *L'histoire $H$ est linéarisable s'il existe une histoire séquentielle $S$ équivalente à $H$ telle que (1) $S$ soit légale et (2) $\overset{TR}{\rightarrow}_H \subseteq \overset{TR}{\rightarrow}_S$.*

Informellement, la linéarisabilité impose en plus que l'histoire séquentielle légale $S$ respecte l'ordre temps-réel des opérations de $H$ que pourrait observer un observateur externe. En d'autres termes, il n'est pas possible de permuter dans $S$ deux opérations ordonnées par $\overset{TR}{\rightarrow}$ dans $H$. Ceci est illustré dans la figure 3.7 qui reprend les histoires $H_1$ et $H_2$ de la figure 3.4.

L'histoire $H_1$ est équivalente à l'histoire séquentielle légale $S_1$ qui respecte l'ordre $\overset{TR}{\rightarrow}$ de $H_1$. En revanche l'histoire $H_2$ n'est pas linéarisable car l'ordre $\overset{TR}{\rightarrow}$ impose pour le registre $X$ que $Lec_i(X)2 \overset{TR}{\rightarrow} Ecr_k(X,2)$ dans l'histoire séquentielle $S_2$. Dans ces conditions, $S_2$ ne peut être légale.

La linéarisabilité est un critère de cohérence qui respecte la propriété de localité. Ce qui donne l'équivalence suivante :

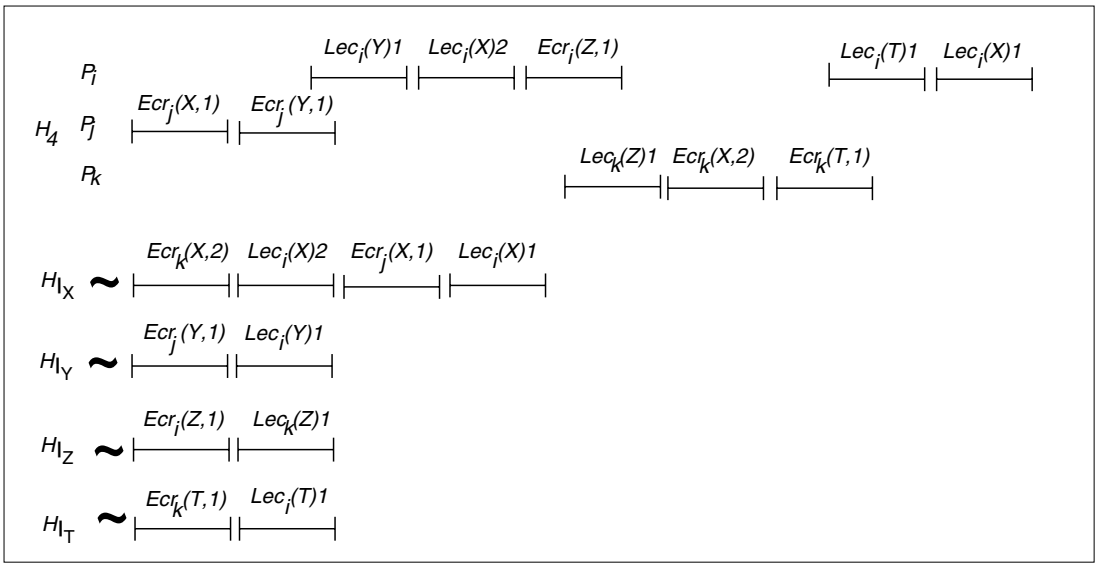


Figure 3.6 : La cohérence séquentielle ne possède pas la propriété de localité

une histoire $H$ est linéarisable si et seulement si pour tout objet $X$ son histoire $H|_X$ est linéarisable (la preuve est donnée dans [Herlihy 90]).

## 3.3 NTR-linéarisabilité

La motivation de la définition de la $NTR$-linéarisabilité est l'adaptation de la linéarisabilité au contexte réparti. En effet, le problème essentiel de la linéarisabilité est de prendre en compte l'ordre global $\stackrel{TR}{\rightarrow}$ sur les événements ce qui, nous l'avons précisé précédemment, est difficilement envisageable pour un système réparti.

Nous proposons un critère de cohérence moins contraignant que la linéarisabilité, mais qui garde la propriété de localité. L'idée de base est de remplacer l'ordre $\stackrel{TR}{\rightarrow}$ par l'ordre $\stackrel{DC}{\rightarrow}$ que nous avons défini dans la section 2.

Nous allons montrer que la relation d'ordre partiel $\stackrel{DC}{\rightarrow}$ permet de définir un critère de cohérence que nous appelons *linéarisabilité non temps-réel* (NTR-linéarisabilité), moins contraignant que la linéarisabilité, mais qui conserve néanmoins la propriété de localité. La NTR-linéarisabilité est définie comme suit :

Définition 3.13 : *NTR-linéarisabilité*
*L'histoire $H$ est NTR-linéarisable s'il existe une histoire séquentielle $S$ équivalente à $H$ telle que (1) $S$ soit légale et (2) $\stackrel{DC}{\rightarrow}_H \subseteq \stackrel{DC}{\rightarrow}_S$.*

La NTR-linéarisabilité est une propriété locale comme le stipule la proposition suivante.

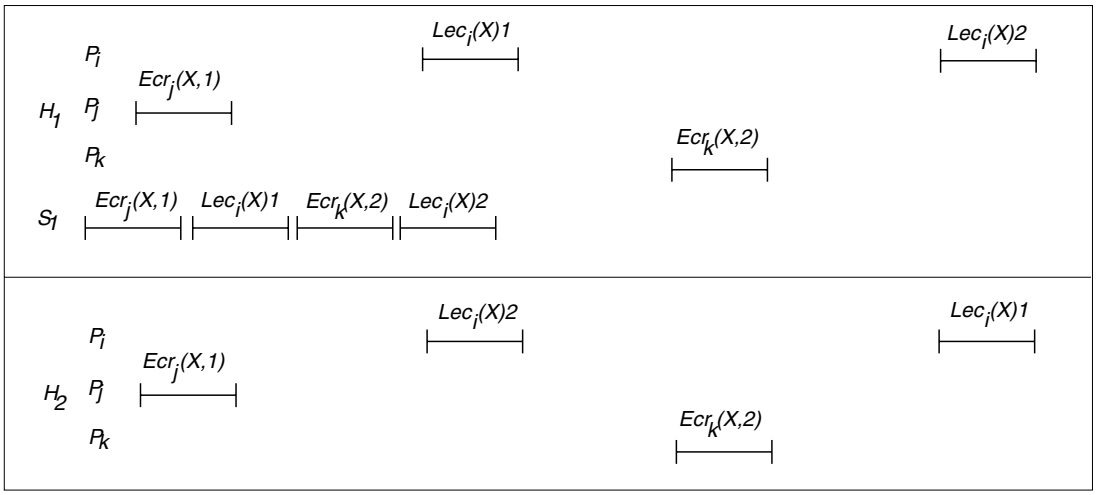


Figure 3.7 : $H_1$ est linéarisable, $H_2$ ne l'est pas

Proposition 1 : *$H$ est NTR-linéarisable $\Leftrightarrow \forall$ objet $X$, $H|_X$ est NTR-linéarisable.*

Pour démontrer cette proposition nous avons besoin du lemme 1 ci-dessous, et de quelques notations supplémentaires. La preuve est inspirée de celle utilisée pour démontrer que la linéarisabilité est une propriété locale [Herlihy 90].

Soit une histoire $H$ noté $(\mathcal{O}, \overset{DC}{\rightarrow}_H)$ et un objet $X$. Conformément aux notations introduites, la projection $H|_X$ de $H$ sur l'objet $X$ peut être notée $(\mathcal{O}|_X, \overset{DC}{\rightarrow}_{H|_X})$. L'histoire $H|_X$ est NTR-linéarisable ssi il existe une histoire séquentielle $S_X$ équivalente à $H|_X$ telle que $S_X$ soit légale et $\overset{DC}{\rightarrow}_{H|_X} \subseteq \overset{DC}{\rightarrow}_{S_X}$. Cette histoire $S_X$ est également notée $(\mathcal{O}|_X, \overset{DC}{\rightarrow}_{S_X})$

Lemme 1 : *Soit une histoire $H$ munie de la relation d'ordre partiel $\overset{DC}{\rightarrow}_H$. Si pour chaque objet $X$, $H|_X$ est NTR-linéarisable, alors on peut construire une relation d'ordre partiel $\overset{R}{\rightarrow}_H$ telle que*
*(1) $\overset{DC}{\rightarrow}_H \subseteq \overset{R}{\rightarrow}_H$*
*(2) pour tout objet $X$, $\overset{DC}{\rightarrow}_{S_X} \subseteq \overset{R}{\rightarrow}_H$.*

PREUVE :
Nous allons montrer que $\overset{R}{\rightarrow}_H = \overset{DC}{\rightarrow}_H \cup \bigcup_X \overset{DC}{\rightarrow}_{S_X}$ est un candidat.

La démonstration se fait par contradiction et l'hypothèse à réfuter est la suivante : $\overset{R}{\rightarrow}_H$, défini ci-dessus, n'est pas une relation d'ordre, en d'autres termes il existe au moins un cycle dans les opérations de $H(\mathcal{O}, \overset{R}{\rightarrow}_H)$.

Soit $\mathcal{C}$ un cycle minimal, $\mathcal{C} = O_1 \overset{R}{\rightarrow}_H O_2 \overset{R}{\rightarrow}_H \ldots \overset{R}{\rightarrow}_H O_n \overset{R}{\rightarrow}_H O_1$.

Si toutes les opérations concernent le même objet $X$, alors $\overset{DC}{\rightarrow}_{S_X}$ n'est pas une relation d'ordre ce qui est faux. Donc les opérations $O_i$ mettent en jeu au moins deux objets.

Par réindexation, soient $O_1$ et $O_2$ deux opérations mettant en jeu des objets différents. Supposons que l'objet manipulé par $O_1$ soit $X$; nous allons prouver que les opérations $O_2, \ldots, O_n$ ne sont pas des opérations sur $X$. Cela est vrai pour $O_2$ par construction, nous allons prouver que cela est vrai pour les autres.

Soit $O_i$ avec $i \neq 2$ une opération mettant en jeu $X$. Les opérations $O_1$ et $O_i$ sont ordonnées par $\overset{DC}{\rightarrow}_{S_X}$ donc sont en relation par $\overset{R}{\rightarrow}_H$ d'après la définition de $\overset{R}{\rightarrow}_H$. Ce qui implique que $O_1 \overset{R}{\rightarrow}_H O_i \overset{R}{\rightarrow}_H \ldots \overset{R}{\rightarrow}_H O_n \overset{R}{\rightarrow}_H O_1$ est un cycle plus petit que $\mathcal{C}$ ce qui est contraire à l'hypothèse que $\mathcal{C}$ est un cycle minimal. Donc $O_2, \ldots, O_n$ ne sont pas des opérations sur $X$.

Comme $O_n \overset{R}{\rightarrow}_H O_1$ et $O_1 \overset{R}{\rightarrow}_H O_2$ par définition de $\mathcal{C}$, et que $O_n$, $O_1$ et $O_2$ ne sont pas en relation par $\overset{DC}{\rightarrow}_{S_X}$ nous avons forcément par construction de $\overset{R}{\rightarrow}_H$ les relations suivantes : $O_n \overset{DC}{\rightarrow}_H O_1$ et $O_1 \overset{DC}{\rightarrow}_H O_2$. La relation d'ordre $\overset{DC}{\rightarrow}_H$ est par définition transitive donc nous avons $O_n \overset{DC}{\rightarrow}_H O_2$. Ce qui implique que $O_2 \overset{R}{\rightarrow}_H \ldots \overset{R}{\rightarrow}_H O_i \ldots \overset{R}{\rightarrow}_H O_n \overset{R}{\rightarrow}_H O_2$ est un cycle plus petit que $\mathcal{C}$ ce qui est en contradiction avec l'hypothèse que $\mathcal{C}$ est un cycle minimal.

Il ne peut donc pas exister de cycle $\mathcal{C}$ et par conséquent $\overset{R}{\rightarrow}_H$ est une relation d'ordre □

PREUVE : (de la proposition)
La partie implication ($\Rightarrow$ ) est évidente, il reste donc à démontrer que
$\forall$ objet $X$, $H|_X$ est NTR-linéarisable $\Rightarrow$ $H$ est NTR-linéarisable.

Puisque chacun des objets $X$ est NTR-linéarisable, il existe pour chacun d'eux une histoire séquentielle $S_X$ équivalente à $H|_X$ telle que (1) $S_X$ est légale et (2) $\overset{DC}{\rightarrow}_{H|_X} \subseteq \overset{DC}{\rightarrow}_{S_X}$.

Considérons l'histoire $H'(\mathcal{O}, \overset{R}{\rightarrow}_H)$, où $\overset{R}{\rightarrow}_H = \overset{DC}{\rightarrow}_H \cup \bigcup_X \overset{DC}{\rightarrow}_{S_X}$ et $\mathcal{O}$ est l'ensemble des opérations de $H$. D'après le lemme 1 et l'hypothèse que les $H_X$ sont NTR-linéarisables, $\overset{R}{\rightarrow}_H$ est une relation d'ordre partiel donc :
(1) par construction, $H'(\mathcal{O}, \overset{R}{\rightarrow}_H)$ est une histoire équivalente à $H$.
(2) $\overset{R}{\rightarrow}_H$ étant une relation d'ordre, alors il est possible de l'étendre dans un ordre total $\overset{R}{\rightarrow}_S$ et ainsi définir $S(\mathcal{O}, \overset{R}{\rightarrow}_S)$, une histoire séquentielle équivalente à $H$.
(3) $S$ est légale car si $S$ n'est pas légale alors par définition il existe une histoire $S_X$ qui n'est pas légale ce qui est contraire aux hypothèses
(4) $\overset{R}{\rightarrow}_H \subseteq \overset{R}{\rightarrow}_S$ par construction.

Donc $H$ est NTR-linéarisable □

**Illustration**

Si l'on reprend la figure $H_4$ de la figure 3.6, l'histoire $H|_X$ n'est pas NTR-linéarisable. En effet, $H|_X$ qui est donnée dans la figure 3.8, ne peut être équivalente à aucune histoire séquentielle légale respectant $DC$. Cela est dû aux dépendances $DCd$ entre les opérations sur $X$ induites par les opérations sur les registres $Y$, $Z$ et $T$.

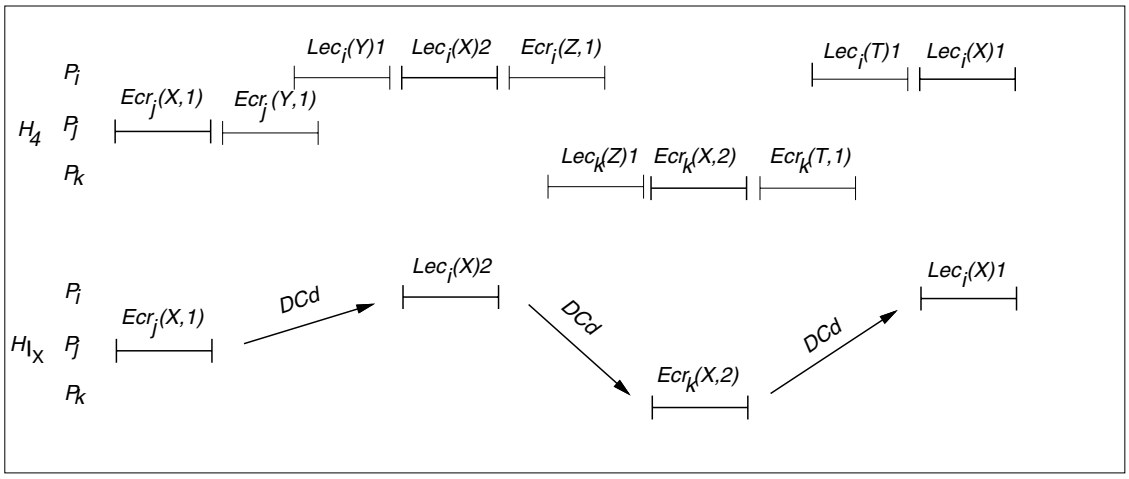


Figure 3.8 : La NTR-linéarisabilité est plus contraignante que la cohérence séquentielle

Ainsi, dans l'histoire $H_4$, les histoires des objets sont séquentiellement cohérentes, mais pas NTR-linéarisables. Ce sont les dépendances $DC$ qui confèrent à la NTR-linéarisabilité la propriété de localité.

# 4 Influence du choix d'un critère de cohérence sur la mise en œuvre

Si l'on considère les définitions précédentes, la linéarisabilité assure la NTR-linéarisabilité, qui implique elle-même la cohérence séquentielle. Les trois critères sont suffisants pour garantir un comportement séquentiel légal d'une exécution, donc suffisants pour garantir un comportement acceptable par l'utilisateur, puisqu'il aurait pu être obtenu si le programme avait été exécuté sur une machine unique. La logique voudrait que, parce que plus permissive, la cohérence séquentielle soit un critère plus facile à mettre en œuvre.

## 4.1 Linéarisabilité vs. cohérence séquentielle

Les définitions précédentes laissent clairement apparaître que la linéarisabilité garantit la cohérence séquentielle, plus la conservation de l'ordre $\stackrel{TR}{\rightarrow}$ entre les opérations.

Deux constatations sont à faire. D'une part, la linéarisabilité est plus contraignante que la cohérence séquentielle, et donc a priori plus difficile à garantir. Toutefois, la cohérence séquentielle n'est pas une propriété locale, et assurer la cohérence séquentielle sur chacun des objets n'assure pas la cohérence séquentielle au niveau du système global. En revanche, la linéarisabilité assure une telle propriété.

L'intérêt de la localité dans un système à grande échelle est très important. Potentiellement, un système à grande échelle peut comprendre un grand nombre d'objets dispersés sur des sites géographiquement éloignés. Assurer un critère de cohérence sur le système entier uniquement en assurant ce même critère sur chacun des objets pris séparément, permet de réduire la complexité du problème de plusieurs ordres de grandeur. La mise en œuvre est plus simple, plus performante et plus facile à tester, puisque modulaire. Le critère de linéarisabilité est donc tout à fait indiqué pour les systèmes à grande échelle, puisqu'il est indépendant de la taille du système. Il est à remarquer, pour étayer cette affirmation, que bon nombre de mises en œuvre de la cohérence séquentielle mettent en fait en œuvre la linéarisabilité. Et cela bien avant que Herlihy et al. en donnent la définition.

### 4.2 NTR-linéarisabilité vs. linéarisabilité

La linéarisabilité est un critère local, car il est possible sur chaque site de construire un ordre total, fonction du temps-réel, sur lequel tous les sites sont d'accord. Cette information globale disponible localement sur chaque site permet un traitement local de la cohérence. La linéarisabilité est donc un critère local parce qu'une information globale est disponible localement : le temps réel. Cependant, ceci présente un défaut majeur car la notion de temps-réel n'est pas disponible dans un système distribué, et recréer le temps-réel par des synchronisations fait perdre tout l'avantage de la linéarisabilité lorsque les synchronisations sont coûteuses (e.g. dans le cas des systèmes distribués sur une grande échelle).

La NTR-linéarisabilité se base sur un ordre partiel qui est inclus dans l'ordre temps-réel. Cet ordre partiel est induit par les dépendances causales à l'intérieur du système. Considérer cet ordre partiel est intéressant, car les informations qui permettent de le garantir sont véhiculées par les interactions entre les objets du système. En effet, pour que deux événements (au sens large) dépendent causalement l'un de l'autre, il faut qu'il y ait une interaction entre les deux, et c'est justement grâce à cette interaction que les informations causales vont être propagées. En d'autres termes, l'information globale suffisante pour assurer l'ordre partiel est disponible localement. Il suffit dès lors de définir une extension linéaire de cet ordre partiel sur lequel tous les sites soient d'accord, pour que la NTR-linéarisabilité soit assurée.

## 5 Conclusion

Nous avons défini dans ce chapitre le modèle de système que nous considérons. Ce modèle nous a permis de présenter et de donner les définitions formelles de deux critères de cohérence : la cohérence séquentielle et la linéarisabilité. Nous nous sommes limités à ces deux critères, car ils nous paraissent plus particulièrement utilisables dans le cadre de l'édition coopérative. Pour de plus amples détails sur les critères non abordés dans ce chapitre, le lecteur peut se

référer à [Dubois 86, Goodman 89, Hutto 90, Gharachorloo 90, Scheuring 89, Ahamad 91] et au travail de synthèse présenté dans [Sinha 93].

Concernant la linéarisabilité, nous proposons une variation de la linéarisabilité classique introduite par Herlihy et al. Dans la définition classique, il est fait mention de temps-réel, notion qui n'est pas exploitable dans les systèmes distribués à cause de l'impossibilité de définir un temps global. Notre approche se base sur la notion de dépendance causale, et permet malgré ces restrictions de conserver les propriétés intéressantes de la définition originelle, en particulier la propriété de localité qui permet de gérer de façon indépendante chacun des objets du système.

# Chapitre 4

# Critères de cohérence : mises en œuvre dans la littérature

## 1 Introduction

Le chapitre précédent a présenté les différents critères de cohérence proposés dans la littérature. Ce chapitre a une approche beaucoup plus pratique du problème de contrôle de cohérence puisqu'il présente les mises en œuvre que l'on trouve dans les différents environnements d'édition coopérative existants.

Dans un premier temps, nous présentons les mécanismes et les environnements d'édition coopérative qui les ont adoptés. Pour chacun de ces mécanismes, nous précisons le critère de cohérence assuré lorsque cela est possible.

Dans un second temps, nous décrivons les problèmes qui sont soulevés par de telles approches en général, et en particulier dans un réseau à grande échelle. Pour chacun des problèmes évoqués nous décrivons l'approche retenue dans DUPLEX, pour pallier aux défauts mis en évidence.

## 2 Mécanismes mis en œuvre dans la littérature

Les protocoles utilisés dans le cadre de l'édition, pour assurer la cohérence du document partagé, sont légion [Ellis 91]. Ils peuvent être décomposés en deux catégories. Les premiers assurent un critère de cohérence fort (e.g., cohérence séquentielle ou linéarisabilité), mais par une mise en œuvre pessimiste du contrôle de concurrence, et les seconds proposent une mise en œuvre optimiste, mais garantissent une sémantique de cohérence beaucoup plus complexe à appréhender.

## 2.1 Mécanismes pessimistes

### Copie primaire (ou duplication passive)

Pour un objet dupliqué $X$, on distingue deux catégories de duplicas. Un duplica particulier est appelé *copie primaire* et les autres *copies secondaires*. Toutes les opérations modifiant un objet $X$ sont réalisées séquentiellement sur la copie primaire, garantissant ainsi la cohérence de l'objet. Les copies secondaires sont mises à jour périodiquement et peuvent par conséquent contenir des informations obsolètes. Au niveau de l'utilisateur, suivant la cohérence souhaitée, les lectures peuvent se faire sur la copie primaire, ou bien sur une copie secondaire présente localement sur le site de l'utilisateur. Les éditeurs coopératifs SharedBook [Lewis 88] et Mace [Newman-Wolfe 91a] sont basés sur ce principe.

Ce protocole garantit la linéarisabilité si tous les accès (écriture et lecture) se font sur les copies primaires. En revanche, si des lectures sont effectuées sur la copie secondaire locale à l'utilisateur[1], ni la linéarisabilité, ni la cohérence séquentielle ne peuvent être garanties. En effet, une lecture d'une copie secondaire a pour effet de lire une valeur dans le passé, et ainsi violer l'ordre temps-réel des opérations requis par la linéarisabilité et la causalité requise par la NTR-linéarisabilité. De plus, la cohérence séquentielle est violée dans le cas suivant :

Le processus $P_i$ réalise les trois opérations $Lec_i(X)0, Lec_i(Y)1, Lec_i(X)0$ et le processus $P_j$ les opérations $Ecr_j(X)1, Ecr_j(Y)1$. Initialement $X$ et $Y$ valent 0. Pour le processus $P_i$ les écritures ne peuvent être effectuées que dans l'ordre $Ecr(Y)1$, puis $Ecr(X)1$. Pourtant le processus $P_j$ les exécute dans l'ordre inverse. Cette histoire n'est pas séquentiellement cohérente bien qu'elle puisse se produire si par exemple les deux lectures de $X$ se font dans la copie secondaire.

### Verrous implicites

Des verrous sont automatiquement acquis et relâchés par le système en fonction des opérations invoquées par les utilisateurs. DistEdit [Knister 93] fonctionne de cette manière, et un verrou doit être acquis avant toute opération accédant à un objet. Cette acquisition étant exclusive, il n'est pas possible d'accéder au même objet tant que le verrou n'a pas été relâché. Ce protocole met en œuvre la linéarisabilité car les verrous permettent de séquentialiser les écritures sur chacun des objets, et d'ordonner de façon correcte les lectures concurrentes par rapport à une écriture. La linéarisabilité de chaque objet est assurée, et par conséquent celle du système.

Certaine mises en œuvre autorise une lecture alors qu'un objet est verrouillé. Dans un con-

[1] C'est généralement dans ce but que le protocole de copie primaire est utilisé.

texte dupliqué, il n'est plus possible de garantir ni la linéarisabilité, ni la NTR-linéarisabilité, ni la cohérence séquentielle puisque cela permet de réaliser une lecture dans le passé. En effet, si les lectures accèdent à des duplicas différents, le même cas de figure que pour les copies secondaires peut se produire.

### Quorum

C'est une méthode basée sur un vote qui impose qu'un quorum (généralement une majorité de duplicas) soit accessible pour pouvoir effectuer une opération (le quorum dépend de l'opération). L'acquisition d'un tel quorum avant de réaliser une opération empêche deux opérations pouvant engendrer un conflit de s'exécuter en même temps. Chaque duplica $x_i$ possède un poids $p_i$. Par exemple, si $N$ est la somme des $p_i$, alors le quorum pour une opération sera de $N/2+1$. Dans ce cas, deux opérations ne pourront être concurrentes. En spécifiant un quorum de $N$ pour les écritures et 1 pour les lectures, les écritures seront exclusives mais des lectures concurrentes seront autorisées. Dans les deux cas de figure, ce protocole met en œuvre la linéarisabilité puisque l'utilisation des quorums a pour conséquence une séquentialisation des opérations.

En spécifiant un quorum de $N/2+1$ pour les écritures et 1 pour les lectures, deux lectures, ou une lecture et une écriture pourront se dérouler concurremment. En revanche, deux écritures ne pourront pas avoir lieu en même temps. Malgré cela, aucun des trois critères ne peut être garanti car des lectures dans le passé sont possibles.

Deux possibilités sont à considérer pour la détermination du quorum : (1) le vote *statique* où le nombre de duplicas est fixe, et le quorum à obtenir est le même y compris si des duplicas ne sont plus atteignables, et (2) le vote *dynamique* qui permet de prendre en compte les pannes de duplicas. Lorsqu'un duplica n'est plus atteignable (ou considéré comme tel), il ne participe plus aux votes pour les opérations suivantes et le quorum nécessaire est diminué en conséquence. Le vote statique peut entraîner des blocages lorsque par exemple plus de la moitié des duplicas sont inaccessibles. Iris [Borghoff 93] utilise un vote dynamique.

### Contrôleur centralisé

Un serveur centralisé pour tout le système est responsable de l'ordre dans lequel les opérations sont effectuées sur chacun des objets. Dans le cas de Mace [Newman-Wolfe 91a] où les données sont centralisées, la mise à jour de la seule copie est effectuée par le serveur. Dans le cas de Mule [Pendergast 90] où les données sont dupliquées et réparties sur les sites utilisateurs, le serveur indique l'ordre dans lequel les opérations doivent être réalisées par les sites stockant les copies.

Ce protocole met en œuvre la linéarisabilité puisque le serveur centralisé séquentialise les

opérations sur les objets soit directement en modifiant l'objet, soit indirectement en définissant l'ordre dans lequel il faut traiter les opérations.

#### Ordonnancement des opérations par objet

C'est une méthode utilisant des mécanismes non centralisés pour déterminer l'ordre dans lequel les opérations doivent être effectuées par chacun des sites contenant des copies. Cet ordre n'est pas forcément un ordre total et plusieurs critères de cohérence peuvent être obtenus suivant l'ordre imposé. Ces protocoles utilisent des boîtes à outils offrant des primitives de diffusions (*multicast*) ordonnées (e.g., ISIS [ISIS 91], PHENIX [Malloth 94]). Par exemple, DistEdit [Knister 93] utilise une diffusion totalement ordonnée (i.e., abcast offerte par ISIS ) pour assurer la mise à jour des copies réparties sur les sites utilisateurs.

Ce protocole met en œuvre la linéarisabilité puisqu'il ordonne totalement les opérations sur chacun des objets.

### 2.2 Mécanismes optimistes

Les méthodes optimistes plus permissives sur le plan de la cohérence sont utilisées dans le cadre des éditeurs en temps-réel. Elles autorisent des incohérences momentanées qui sont corrigées au fur et à mesure de la disponibilité de nouvelles informations.

#### Opérations réversibles

A la réception d'une requête par un des duplicas, une opération est effectuée immédiatement, tout en sauvegardant un certain nombre d'informations afin de pouvoir défaire cette opération le cas échéant. Un mécanisme de synchronisation indique régulièrement à chacun des duplicas dans quel ordre les opérations doivent (plus justement auraient dû) être exécutées. A la réception d'une telle information, chaque duplica défait les opérations qui ne sont pas correctement ordonnées puis les applique dans le bon ordre. Ainsi les sites peuvent avoir momentanément des états incohérents, mais ceux-ci sont rapidement corrigés. Cette méthode a été proposée par [Sarin 85] pour les applications faisant intervenir des utilisateurs multiples. La justesse des données est directement liée à la granularité des opérations et à la fréquence des synchronisations.

Sur le plan de la cohérence, ce protocole assure la convergence des copies : il garantit qu'à terme toutes les copies seront identiques même si elles sont passées par des états différents.

**Transformation d'opération**

C'est une méthode basée sur les priorités et la sémantique des opérations, et suppose que chaque utilisateur possède un duplica local. Un duplica possède un *vecteur d'état* contenant le nombre d'opérations en provenance de chacun des autres duplicas qu'il a déjà effectués. La requête d'un utilisateur est immédiatement appliquée sur le duplica local, puis envoyée aux autres duplicas accompagnée du vecteur d'état du duplica local. A la réception de cette opération, chaque duplica distant compare son vecteur d'état avec celui de l'opération. Si les deux vecteurs coïncident, l'opération peut être appliquée, dans le cas contraire l'opération est transformée en conséquence avant d'être appliquée. Cette technique est employée dans GROVE [Ellis 91] et le lecteur pourra se référer à [Ellis 89] pour plus de détails sur l'algorithme de transformation d'opérations. Cette méthode peut être considérée comme une détection de dépendance avec une résolution automatique des conflits pour laquelle il n'y a pas de retour en arrière, mais une modification de l'opération à appliquer.

Ce protocole garantit également la convergence des copies.

# 3 Faiblesse des mécanismes présentés pour une utilisation dans un réseau à grande échelle

Dans la littérature, les problèmes de critère de cohérence, ne sont pas réellement abordés ou du moins ne prennent en compte ni les spécificités du système distribué sous-jacent, ni celles de l'application considérée. Ainsi, dans la plupart des précédentes approches, le contrôle de concurrence, pour éviter les opérations conflictuelles entre les différents auteurs, est imposé par le système. Les conséquences sont (1) un manque de souplesse puisque l'utilisateur ne peut pas réellement choisir sa manière de travailler, (2) le critère de cohérence assuré par le système est en général beaucoup trop fort pour l'édition coopérative, et (3) la mise en œuvre en est par conséquent beaucoup trop contraignante.

Le premier point oblige l'utilisateur à adapter sa manière de travailler aux mécanismes imposés par l'environnement d'édition coopérative. Cette approche est une entrave à l'utilisation généralisée de ce genre d'outils.

Le deuxième point conduit à une restriction de la concurrence potentielle, car la sémantique des objets considérés (partie de document) n'est pas prise en compte. En effet, dans l'édition coopérative, la sémantique du contenu d'une partie de document est plus important que le contenu lui-même, et une cohérence faible est souvent tolérable; ce qui n'est pas le cas d'un registre. Par exemple, le contenu d'une partie du document peut avoir changé alors que la sémantique reste la même. En d'autres termes, lire une vieille version de la section 1 est tout à fait acceptable lors de la rédaction de la section 2.

Finalement, le troisième point pénalise les utilisateurs quand les performances du réseau ne sont pas au rendez-vous. Par exemple, lorsqu'un réseau à grande échelle est considéré, il devient difficile de gérer la duplication des données avec les mécanismes proposés, soit parce que ces derniers deviennent extrêmement coûteux, soit parce qu'ils ne prennent pas en considération les pannes qui peuvent survenir dans un tel réseau. Un réseau à grande échelle n'est pas l'extrapolation d'un réseau local.

Les modèles considérés dans la littérature sont mal adaptés à l'édition coopérative dans un réseau à grande échelle. Ce point de vue est illustré dans les sections suivantes où nous mettons en évidence les faiblesses de ces modèles et où nous proposons une alternative.

## 3.1 Lecture et écriture vs. lecture-modification-écriture

La première constatation est que la démarche communément utilisée dans ces environnements est biaisée par le fait qu'ils considèrent comme opération de base des opérations du type lecture-modification-écriture. C'est à dire, charger la partie du document dans l'éditeur (lecture), éditer (modification), et sauvegarder (écriture). Pour garantir l'atomicité de telles opérations, il est nécessaire que l'état de l'objet ne soit pas modifié par un autre utilisateur avant que l'écriture ne soit effectivement réalisée, i.e., entre la lecture et l'écriture.

Dans de telles conditions, une alternative est possible : (1) l'atomicité des opérations lecture-modification-écriture est assurée par une politique pessimiste et la cohérence est assurée à un prix élevé en terme de restriction de la concurrence, ou (2) l'atomicité des opérations lecture-modification-écriture n'est pas assurée et il est impossible de garantir un quelconque critère de cohérence.

Si l'approche consistant à considérer les opérations de type lecture-modification-écriture est envisageable dans un environnement local (un certain nombre de mises en œuvre en sont le témoignage), il n'en est pas de même lorsque l'on se place dans un réseau à grande échelle. Cela pour deux raisons essentielles. La première raison est liée à la multiplicité des utilisateurs, à l'éloignement géographique de ces derniers (différents fuseaux horaires, non connaissance des activités annexes de chacun), et au type d'interaction fortement asynchrone. Cela impose une mise en œuvre optimiste car il n'est pas envisageable, par exemple, qu'un utilisateur ne puisse pas travailler à cause d'un autre utilisateur aux antipodes qui a oublié de relâcher le verrou sur un document avant de regagner son domicile. La deuxième raison est que la difficulté pour chacun des utilisateurs d'appréhender le système dans sa globalité requiert une sémantique claire quant au critère de cohérence considéré.

Pour notre part, nous considérons qu'il faut garantir le critère de cohérence au niveau des opérations de lecture et d'écriture, ce qui peut être réalisé sans entraîner des délais au niveau des utilisateurs. Garantir l'atomicité des opérations lecture-modification-écriture (plus coû-

teuses en temps, à cause de la phase de modification) est certes utile, mais non nécessaire en permanence. De plus, quand elle est nécessaire, elle peut être assurée de manière optimiste pour peu que les connaissances des utilisateurs soient prises en compte. En effet, plusieurs méthodes de travail permettant cette approche optimiste peuvent être considérées par les membres de la coopération. Par exemple, en réglant les conflits a posteriori par une phase de conciliation. Il est clair qu'une approche pessimiste doit également être envisagée dans certains cas. Cependant, il ne faut pas que ce soit le prix à payer systématiquement.

En effet, en garantissant la cohérence des opérations lecture, écriture au niveau du système il est possible d'avoir une mise en œuvre efficace, car les critères de cohérence considérés sont bien adaptés à ce genre d'opération. En revanche, garantir l'atomicité des opérations lecture-modification-écriture ne peut être fait par le système car elles requièrent beaucoup plus de connaissances, que seuls les utilisateurs sont en mesure de fournir. L'utilisateur a besoin de mécanismes qui lui sont proposés, et non imposés.

## 3.2 Décomposition dynamique du document dirigée par les utilisateurs

Un autre problème posé dans le cadre du contrôle de concurrence est la granularité des données, en d'autres termes la taille des parties du document lorsque celui-ci est décomposé. En fragmentant le document, il est possible d'augmenter la concurrence en permettant d'accéder séparément aux parties indépendantes; réciproquement en regroupant des parties indépendantes, il est possible de séquentialiser des opérations qui autrement auraient été réalisables de façon concurrente.

Bien que la décomposition du document soit utilisée dans la littérature (CES [Greif 92], Iris [Borghoff 93], Quilt [Fish 88, Leland 88], Griffon [Decouchant 93], SharedBook [Lewis 88] et MultimETH [Lubich 90]), aucune approche ne prend en compte cette propriété dans le processus de contrôle de concurrence. La plupart du temps, le découpage est statique ou du moins prédéfini (e.g., dans SharedBook ou Quilt les sections sont des parties indépendantes), et l'utilisateur n'a aucun moyen de contrôle sur le découpage. Les approches utilisant une décomposition dynamique (Iris, CES, Griffon et MultimETH) ne laissent pas l'utilisateur maître de la décomposition, puisque c'est le système qui s'en charge. Par exemple, dans Griffon [Decouchant 93] le découpage est lié à la notion de rôle et une partie indépendante est une partie pour laquelle tous les utilisateurs ont un rôle uniforme. Ainsi il suffit qu'un utilisateur change de rôle sur une section pour que cette dernière devienne indépendante, et ce quel que soit l'état de maturité du document.

Pour notre part, nous pensons que la décomposition du document doit être utilisée pour adapter la concurrence autorisée à la maturité du document. Par exemple, considérons une section contenant un théorème et la preuve du théorème. Si le théorème et la preuve sont deux parties indépendantes, la rédaction peut être menée de front par deux utilisateurs.

Ceci est souhaitable puisque uniquement le sens du théorème et de sa preuve est utile aux deux rédacteurs, et non les termes exacts de leur rédaction. Une fois la démonstration du théorème achevée, il convient de faire converger les notations et la formulation définitive du théorème et de sa preuve. Par exemple, dans l'état initial de la figure 4.1 le nom de l'ensemble n'est pas le même. Si deux utilisateurs décident de faire la mise à jour et que l'utilisateur $U_1$ remplace F par E dans le théorème et que l'utilisateur $U_2$ remplace E par F l'incohérence subsiste. Si pour réaliser cette unification, le théorème et la preuve sont dans une même partie, empêchant tout accès concurrent au théorème et à sa preuve, l'incohérence sera évitée.

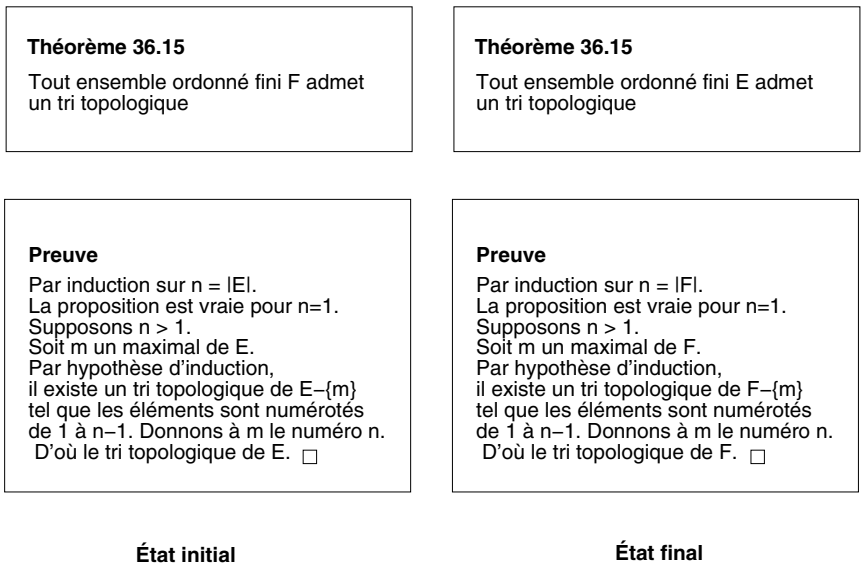


Figure 4.1 : Régulation de la concurrence en utilisant la décomposition du document

En imposant un critère de cohérence tel que la NTR-linéarisabilité au niveau de chaque partie du document, et en jouant sur la décomposition du document, un large spectre de critères de cohérences est disponible En d'autres termes, la cohérence du document est la plus forte (NTR-linéarisabilité du document) lorsque le document est un bloc monolithique, la cohérence est la plus faible lorsque chacun des paragraphes est indépendant (NTR-linéarisabilité de chacun des paragraphes). Plus le document est fragmenté plus sa cohérence est faible.

# 4 Conclusion

Nous avons décrit dans ce chapitre les différents protocoles mis en œuvre dans les systèmes existants, en mettant l'accent sur le type de cohérence et en distinguant les mises en œuvre optimistes et pessimistes.

La majorité des mises en œuvre assurent la linéarisabilité (copie primaire, verrous, quorum, serveur centralisé, ordonnancement des opérations), mais d'une manière pessimiste. En effet, des opérations peuvent être bloquées en attente de la terminaison d'autres opérations. Ceci peut être pénalisant dans un domaine de coopération à grande échelle, où le problème des pannes, empêchant la terminaison de certaines opérations, peuvent bloquer le système. Les mises en œuvre optimistes (transformation d'opération et exécution réversible) ne garantissent ni la linéarisabilité, ni même la cohérence séquentielle, et n'ont un intérêt que dans le cadre d'éditeurs temps-réel, où le flux d'information est suffisamment important pour corriger rapidement les informations momentanément incohérentes.

Cette constatation nous a amené à proposer dans DUPLEX une solution qui permette de pallier à ces inconvénients. Deux concepts ont été introduits dans le modèle de DUPLEX : (1) Une décomposition dynamique du document contrôlée par les utilisateurs, qui reflète la maturité du document et qui permet de définir un large spectre de cohérence, et (2) plusieurs politiques de contrôle de concurrence optimistes et pessimistes offertes aux utilisateurs pour garantir l'atomicité des opérations de type lecture-modification-écriture sur une même partie de document.

# Chapitre 5

# Mise en œuvre de la NTR-linéarisabilité

## 1 Introduction

Comme nous l'avons vu dans le chapitre précédent, la NTR-linéarisabilité est un critère de cohérence qui autorise un traitement modulaire au niveau de chacun des objets. Cependant, la duplication complique le problème, puisqu'il faut dès lors considérer l'ensemble de duplicas correspondant à un même objet. En effet, si un objet logique $X$ est dupliqué, les $n$ duplicas $\{x^1, ..., x^n\}$ devront être maintenus comme une seule et même entité logique. Ceci impose de coordonner les opérations individuelles sur ces duplicas, ce qui peut s'avérer coûteux quand les duplicas sont dispersés dans un système à grande échelle.

Ce chapitre propose un protocole prenant en compte les propriétés des objets manipulés afin de permettre un traitement ad-hoc moins coûteux qu'un protocole générique. Ce qui permet d'envisager de façon réaliste une mise en œuvre efficace sur un réseau à grande échelle.

La section deux présente le modèle dans lequel le protocole se place. La section 3 présente brièvement le protocole générique. La section 4 propose un protocole amélioré pour le type d'objets manipulés dans DUPLEX. La section 5 conclut le chapitre.

## 2 Présentation du modèle

### 2.1 Présentation du contexte dupliqué

La figure 5.1 présente une superposition de la vue physique (représentée en minuscules) et de la vue logique (représentée en majuscules) d'une exécution. Nous considérons un objet

logique $X$ constitué de 3 duplicas $x^1$, $x^2$ et $x^3$. A chaque invocation logique émise par le processus $P_i$ correspondent 3 invocations physiques de chacun des duplicas $x^1$, $x^2$ et $x^3$. La duplication est active[Barrett 90], tous les duplicas jouent le même rôle. Ainsi le processus $P_i$ recevra 3 réponses à sa requête. La réponse logique est fonction des réponses physiques[1].

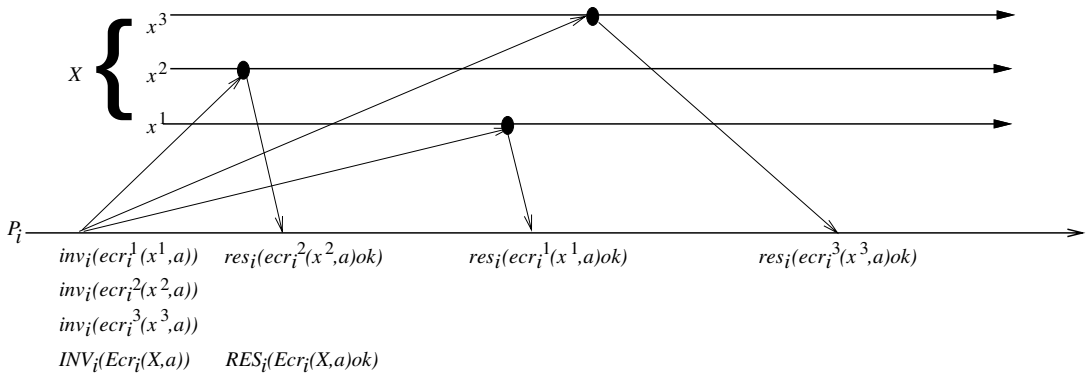


Figure 5.1 : Protocole primaire

Si l'on se place dans le cadre de l'édition coopérative, un processus correspond à un utilisateur, et un objet à une partie du document.

Les protocoles que nous allons décrire ci-dessous utilisent les notions introduites par le *modèle virtuellement synchrone* présenté brièvement ci-après.

## 2.2 Présentation du modèle virtuellement synchrone

Dans le *modèle virtuellement synchrone* [Birman 93, Schiper 93], il est possible de regrouper des processus dans des *groupes*. Les membres d'un groupe reçoivent une séquence de *vues* qui représentent chacune la composition du groupe du moment. Ainsi, chaque fois qu'un processus joint ou quitte le groupe $G$, tous les processus du groupe $G$ reçoivent la composition de la nouvelle vue. Un *multicast* à un groupe $G$ peut être émis aussi bien par un processus du groupe $G$ que par un processus externe. Les multicasts envoyés au groupe sont ordonnés par rapport aux changements de vues. Ainsi, tous les messages correspondant à un multicast sont reçus soit avant le changement de vue, soit après, par tous les membres du groupe. Plusieurs types de multicasts peuvent être distingués en fonction (1) de la qualité de service assurée, et (2) de l'ordonnancement que l'on veut obtenir entre les différents multicasts.

[1]Suivant ce qu'assure le gestionnaire de cohérence, une ou plusieurs réponses doivent être considérées pour que la réponse logique soit donnée.

Plusieurs qualités de service pour la livraison d'un multicast $m$ sont disponibles :

**fiable :** si un des membres du groupe $G$ reçoit un message $m$ et qu'il ne tombe pas en panne, alors tous les membres du groupe reçoivent le message $m$.

**uniforme :** si un des membres du groupe $G$ reçoit un message $m$, alors tous les membres du groupe reçoivent le message $m$

Il est possible de garantir un certain ordre entre les multicasts :

**aucun :** aucun ordre n'est garanti a priori.

**fifo :** si deux multicasts $m1$ et $m2$ sont envoyés par le même processus, dans l'ordre $m1$ puis $m2$, alors les processus du groupe $G$ recevront $m1$ puis $m2$.

**causal :** si deux multicasts $m1$ et $m2$ sont liés par une dépendance causale ($m1$ précède causalement $m2$), alors les processus du groupe $G$ recevront $m1$ puis $m2$.

**total :** deux multicasts, $m1$ et $m2$, émis par deux processus distincts, seront reçus par tous les processus de $G$ dans le même ordre.

**total causal :** deux multicasts, $m1$ et $m2$, émis par deux processus distincts, seront reçus par tous les processus de $G$ dans le même ordre, et cet ordre est compatible avec la causalité.

Un certain nombre de boîtes à outils assure tout ou partie de ces services (e.g., ISIS [ISIS 91], TRANSIS [Amir 92], DELTA4 [Barrett 90] et PHENIX [Malloth 94]). Nous avons utilisé ISIS pour la première mise en œuvre; et PHENIX, qui est développé au sein du laboratoire, sera utilisé pour la mise en œuvre définitive.

# 3 Protocole générique

Nous allons dans un premier temps présenter la manière la plus simple de garantir la NTR-linéarisabilité, en utilisant les primitives de communication introduites ci-dessus. Dans un deuxième temps, nous apporterons des améliorations visant à rendre le protocole plus performant.

Les duplicas d'un même objet sont regroupés au sein d'un groupe[2]. Il existe donc un groupe par objet logique. Les processus $P_i$ sont externes.

Afin de rendre la duplication des objets transparente aux processus et aux duplicas, leur code est encapsulé dans une coquille (*shell*) qui sert d'interface. Ainsi, l'introduction de la

[2]Un duplica est un objet actif, il est donc géré par un processus. Par groupe de duplicas nous entendons groupe de processus gérant un duplica du même objet.

duplication ne nécessite pas la modification du code des duplicas et des processus. En particulier, la coquille traduit les invocations et réponses physiques en invocations et réponses logiques et réciproquement.

Une requête émise par un processus $P_i$ à destination d'un objet logique $X$ sera donc transformée en multicast à destination du groupe de duplicas de $X$ par la coquille . En retour, bien que chaque duplica traite la requête et retourne une réponse, la coquille du processus $P_i$ ne considérera qu'une seule réponse (la première) qu'elle transmettra à $P_i$. La coquille d'un duplica $x^j$ est responsable des éventuels traitements dus à la duplication comme nous le verrons ci-dessous.

Le protocole générique consiste à utiliser un multicast totalement ordonné causal (abcast d'ISIS [ISIS 91]) aussi bien pour les lectures que pour les écritures. Ceci garantit que tous les duplicas sont toujours dans le même état, et que par conséquent lors d'une lecture, toutes les réponses seront identiques.

# 4 Protocole optimisé

Le protocole optimisé se base sur la sémantique des objets manipulés pour proposer un protocole ad-hoc plutôt qu'un protocole générique comme le précédent. Ce protocole, utilisant la sémantique des objets manipulés, permet de ne plus utiliser un multicast totalement ordonné causal, mais un multicast fiable complété par un contrôle explicite des dépendances causales. Ainsi, l'idée de départ est de considérer un multicast fiable pour les requêtes d'écriture et d'ordonner les requêtes d'écriture par rapport aux requêtes de lecture. Ceci est développé ci-dessous.

### Relaxation des contraintes d'ordonnancement

Les objets que nous considérons sont de type atome, l'extension de la notion de registre à des objets plus complexes comme par exemple les fichiers ou les parties de document. Un objet atome, à l'instar du registre, n'admet que deux opérations :

**l'opération d'écriture** qui consiste en un écrasement (*overwrite*) de la valeur précédente de l'objet par la nouvelle valeur : l'objet entier est modifié.

**l'opération de lecture** qui retourne la valeur courante de l'objet. La définition de la valeur courante est liée à la spécification séquentielle des objets de type atome, et peut être définie de la même manière que les registres [Misra 89].

Supposons qu'il existe un ordre, global à tous les duplicas, dans lequel les opérations d'écritures doivent être traitées. Nous verrons par la suite comment chaque duplica peut

obtenir localement cette information. Avec des objets de type atome il est possible d'exécuter les opérations d'écritures dès réception, et de ré-ordonnancer, a posteriori, les écritures arrivées en retard, comme le montre la figure 5.2.

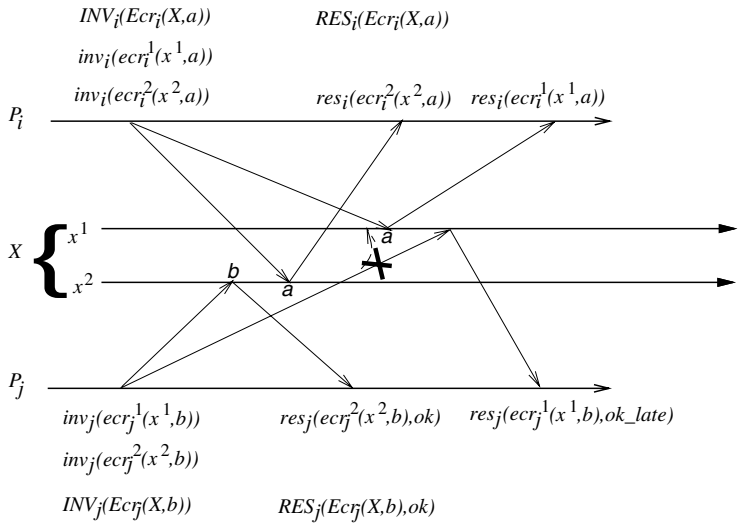


Figure 5.2 : Ré-ordonnancement a posteriori

Les deux processus $P_i$ et $P_j$ écrivent respectivement les valeurs $b$ et $a$ sur l'objet logique $X$. Les deux écritures sont reçues dans des ordres différents par les deux duplicas $x^1$ et $x^2$.

Supposons que l'ordre des écritures impose l'écriture de $b$ comme la première. Le duplica $x^2$ reçoit les écritures dans le bon ordre à savoir l'écriture de $b$, puis celle de $a$, et les applique dans cet ordre. En revanche, le duplica $x^1$ reçoit d'abord l'écriture de $a$, l'exécute immédiatement, puis reçoit celle de $b$. Le duplica $x^1$ s'aperçoit que l'écriture de $b$ aurait dû être exécutée avant celle de $a$. La requête d'écriture va tout simplement être ignorée, et la réponse retournée sera *ok*. En fait, cette écriture est considérée comme étant arrivée juste avant l'écriture de $a$, et immédiatement écrasée par l'écriture de $a$, avant qu'aucune lecture ne puisse en prendre connaissance. Pour un observateur externe il est impossible de distinguer ces deux scénarii. Cette solution rend les écritures plus efficaces.

Reste le problème des lectures. Il faut éviter qu'une requête de lecture retourne plusieurs résultats différents, pouvant conduire à des lectures dans le passé. Il faut donc faire un traitement à la réception d'une requête de lecture qui assure aux duplicas d'être dans un état cohérent. La solution présentée ici consiste à garantir que tous les duplicas aient traité le même ensemble de requêtes d'écriture avant de traiter la requête de lecture.

## 4.1 Intérêt du protocole

Paradoxalement, une requête de lecture va être plus coûteuse qu'une requête d'écriture. L'intérêt du protocole est que les écritures sont peu coûteuses. Une requête d'écriture est traitée immédiatement, ne nécessitant ni latence, ni stockage. La synchronisation ne se faisant que pour les requêtes de lecture. Et dans le cas des lectures, ne sont attendues que les requêtes d'écriture qui auraient de toutes façons été attendues avec le protocole générique. Ce protocole est particulièrement intéressant lorsque les écritures sont plus nombreuses que les lectures. Par exemple, lorsqu'un utilisateur réalise une lecture pour obtenir la partie de document à modifier et réalise plusieurs écritures au fur et à mesure que son travail progresse.

## 4.2 Structures de contrôle

La mise en œuvre proposée nécessite les structures de contrôle suivantes. Chaque processus $P_i$ maintient une horloge logique [Lamport 78] $C_{P_i} = (T, I_{P_i})$ où $T$ est un compteur et $I_{P_i}$ l'identificateur unique associé au processus $P_i$. L'évolution de $T$ reflète les précédences causales [Lamport 78]. Le champ $I_{P_i}$ est une constante entière utilisée pour ordonner totalement les valeurs d'horloge qui auraient un champ $T$ identique[3]. Chaque processus $P_i$ maintient également pour chaque objet atome $X$ qu'il connaît une horloge logique $C(X)$ du type précédemment défini, et reflétant l'évolution de ce dernier. Cette structure de données représente donc une horloge vectorielle [Mattern 89] notée $V_{P_i}$, contenant une entrée par objet logique.

Pour chaque objet logique $X$, chacun des duplicas $x^j$ possède également les mêmes structures de contrôle $V_{x^j}$ et $C_{x^j}$.

### Évolution des structures de contrôle

Les invocations, les réponses, et les communications entre processus, sont les événements responsables de l'évolution des structures de contrôle $C$ et $V$, mais également de leur propagation à travers le système. Ainsi, chaque message (invocation, réponse ou communication) sera accompagné des structures $C$ et $V$ de l'entité émettrice, et à la réception les structures locales seront modifiées en fonction de celles qui accompagnaient le message.

La coquille dans ce cas là a deux fonctions, (1) la traduction des invocations et réponses physiques en invocations et réponses logiques et réciproquement, et (2) la gestion des structures de contrôles $C$ et $V$ de l'entité qu'elle encapsule.

[3] E.g., l'identification unique de $P_i$.

## 4.3 Protocole d'un processus $P_i$

Les structures de contrôle maintenues par la coquille du processus, évoluent de la manière suivante.

**i)** *Lors de la réception des réponses.* Supposons que la réponse prise en considération (la première) émane du duplica $x^j$, et notons $C_{x^j}$ et $V_{x^j}$ les deux structures de contrôle associées à la réponse. La coquille de $P_i$ va prendre en compte $C_{x^j}$ et $V_{x^j}$ pour modifier les structures de contrôle locales à $P_i$ (i.e., $C_{P_i}$ et $V_{P_i}$). L'horloge logique $C_{P_i}$ prend la valeur $max(C_{P_i}, C_{x^j} + 1)$ (Cf. [Lamport 78]); l'horloge vectorielle $V_{P_i}$ prend la valeur $max(V_{x^j}, V_{P_i})$, où l'opération $max$ est appliquée à chaque composant du vecteur (Cf. [Mattern 89]). Cela permet de notifier au processus $P_i$ les informations connues par le duplica $x^j$ ayant produit la réponse considérée. L'horloge logique $C_{x^j}$ correspond à l'instant logique dans lequel se trouve le duplica $x^j$ au moment où il poste la réponse. L'horloge vectorielle $V_{x^j}$ correspond aux instants logiques des dernières écritures effectuées sur les objets logiques connus par le duplica $x^j$.

**ii)** *Lors de la réception d'une communication d'un autre processus $P_j$.* L'horloge logique $C_{P_i}$ est mise à jour à $max(C_{P_i}, C_{x^j} + 1)$ et $V_{P_i}$ est mise à jour à $max(V_{x^j}, V_{P_i})$. Signifiant que les deux processus sont désormais au même instant logique.

## 4.4 Protocole d'un duplica $x^j$

Tous les duplicas d'un même objet logique effectuent le même traitement (duplication active). Après un contrôle préliminaire, une invocation reçue par la coquille du duplica $x^j$ est soit répercutée sur le duplica lui-même et exécutée de façon transparente, soit, dans certains cas, directement générée par la coquille du duplica. Ce contrôle préliminaire est fonction du type de requête, il est décrit ci-dessous.

### Requête d'écriture.

Lorsque la coquille d'un duplica $x^j$ reçoit une requête d'écriture provenant d'un processus $P_i$, elle compare la valeur de l'horloge $C_{P_i}$ avec la valeur de sa propre horloge $C_{x^j}$. Deux cas de figure sont à considérer :

$C_{P_i} < C_{x^j}$

L'instant logique du processus $P_i$ initiateur de la requête est en retard par rapport à l'instant logique de $x^j$. La requête est considérée comme arrivée en retard et est simplement ignorée (Cf. Section 4). Ce traitement est compatible avec le critère de cohérence que l'on veut assurer (la NTR-linéarisabilité). Il faut toutefois récupérer les

informations qui sont transmises avec l'invocation de $P_i$. Le vecteur $V_{x^j}$ prend la valeur $max(V_{x^j}, V_{P_i})$ et la coquille génère une réponse indiquant un statut $ok\_en\_retard$[4]. Aucune mise à jour n'est soumise au duplica $x^j$.

$C_{P_i} \geq C_{x^j}$

L'instant logique du processus est supérieur à l'instant logique du duplica $x^j$. La requête va donc pouvoir être considérée. Il faut mettre à jour les structures de contrôle $C_{x^i}$ et $V_{x^i}$. L'horloge $C_{x^j}$ prend la valeur $C_{P_i}$ indiquant que le processus et le duplica sont désormais au même instant logique. La composante $V_{x^i}[X]$ prend également la valeur $C_{P_i}$, indiquant que cette écriture doit être considérée comme la dernière écriture réalisée sur l'objet logique $X$. L'horloge vectorielle $V_{x^j}$ prend la valeur $max(V_{x^j}, V_{P_i})$, notifiant le duplica des actions les plus récentes sur chacun des objets logiques. La coquille invoque le duplica $x^j$ avec les paramètres de la requête (i.e., la valeur à écrire) et attend la réponse du duplica. Lorsque la réponse arrive à la coquille, cette dernière la transmet à $P_i$ accompagnée des nouvelles valeurs de $C_{x^j}$ et $V_{x^j}$.

Dans les deux cas, le traitement d'une requête d'écriture est peu coûteux.

**Requête de lecture.**

Lorsque la coquille d'un duplica $x^j$ reçoit une requête de lecture de la part d'un processus $P_i$ il faut éviter que la réponse donnée ne constitue une lecture dans le passé. Il faut donc que les duplicas se synchronisent afin de rétablir un état consistant.

Plusieurs solutions sont possibles. Celle qui a été choisie se base sur les propriétés du modèle virtuellement synchrone. Un changement de vue va être déclenché au niveau du groupe de duplicas. Lors d'un changement de vue, les différents membres du groupe se mettent d'accord (consensus) sur (1) les membres du groupe qui constitueront la nouvelle vue, et (2) les messages qui doivent être délivrés avant le changement de vue (i.e., tous ceux qui ont été vus par au moins un des membres de l'ancienne vue participant à l'établissement de la nouvelle vue).

Le changement de vue considéré ici est un changement de vue "allégé" puisque la composition de la vue reste la même, et en général, tous les duplicas auront reçu toutes les invocations d'écriture (vu le protocole utilisé, un message manquant ne peut être qu'une invocation d'écriture). Cependant, il peut arriver qu'une invocation d'écriture soit manquante. Le changement de vue sert à garantir que les invocations d'écriture manquantes soient délivrées sur tous les duplicas. Ceci, additionné au protocole gérant les écritures décrit ci-dessus, permet de garantir que les duplicas sont tous dans le même état (même contenu, même

[4]Cela signifie pour le processus qui recevra cette réponse que l'écriture a été exécutée en respectant le critère de cohérence mais qu'elle a été écrasée par une autre écriture.

structures de contrôle) lorsque le pseudo changement de vue est achevé. La réponse à la requête de lecture est la valeur courante du duplica. Elle est interceptée par la coquille et transmise au processus $P_i$ accompagnée de $C_{x^j}$ et de $V_{x^j}$, mises à jour comme précédemment.

# 5 Traitement des pannes

Trois types de pannes sont à considérer, (1) la panne d'un processus $P_i$, (2) la panne d'un duplica $x^j$, et (3) la partition du réseau.

## 5.1 Panne de processus $P_i$

Si le processus tombe en panne pendant qu'une opération est en cours deux cas de figure sont à envisager suivant qu'il y ait eu ou non propagation de l'information causale (Cf. Figure 5.3)

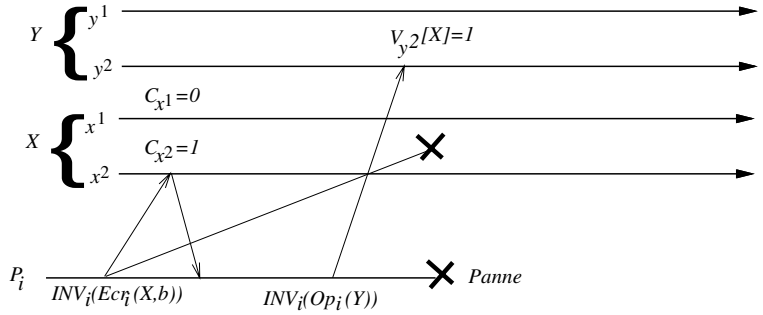


Figure 5.3 : Propagation de l'information causale

La requête d'écriture $Ecr_i()$ du processus $P_i$ n'a atteint que le duplica $x^2$ qui a répondu. La seconde requête $Op_i()$ atteint le duplica $y^2$ d'un autre objet puis le processus $P_i$ tombe en panne. L'existence de la première requête est transportée par la seconde; en particulier $V_{y^2}[X] = 1$, ce qui signifie que l'objet logique $X$ a traité une requête. Or, le duplica $x^1$ n'a pas reçu la requête.

Voyons le comportement en cas de requête ultérieure sur $X$. En cas de lecture de $X$ par un processus $P_j$, le changement de vue occasionné fera que le duplica $x^1$ recevra l'écriture manquante par l'intermédiaire de $x^2$, puis la requête de lecture sera traitée; la réponse envoyée par les deux duplicas sera correcte.

En cas d'écriture de $X$ par un autre processus $P_j$, l'écriture sera soit ordonnancée après celle de $P_i$ ce qui aura pour effet d'effacer les traces de l'écriture de $P_i$, soit ordonnancée avant celle de $P_i$, ce qui laissera les duplicas de $X$ dans un état similaire.

A terme, une requête de lecture ou d'écriture finira donc par effacer la trace de l'écriture man-

quante. L'écriture manquante, ne peut en aucune manière bloquer un processus, puisqu'au moins un duplica a vu l'écriture. La panne éventuelle du duplica (lorsqu'un seul a vu l'écriture) est discuté dans la section suivante.

## 5.2 Panne de duplica $x^j$

Ce type de panne nécessite plus d'attention car il peut conduire au scénario catastrophe suivant :

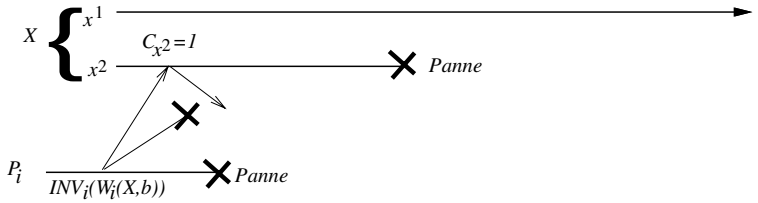


Figure 5.4 : Cas extrême pour une panne de duplica

Le processus $P_i$ émet une requête d'écriture sur l'objet $X$ qui atteint le duplica $x^2$. Le duplica $x^1$ n'est jamais atteint car $P_i$ tombe en panne. Bien que $P_i$ reçoive une réponse de $x^2$, si $x^2$ tombe en panne, il n'y aura pas de trace de l'écriture dans le système. Deux solutions sont possibles : (1) utiliser un multicast uniforme pour l'envoi des requêtes d'écriture, ce qui sur le plan théorique est satisfaisant mais qui peut s'avérer coûteux à l'usage, ou (2) décider que dans ce cas de figure extrême l'écriture ne peut être garantie.

Au niveau du groupe de duplicas, la cohérence est assurée dans les deux cas. Au niveau des autres utilisateurs, les deux cas de figure sont également satisfaisants. La seule différence visible est pour le processus qui est tombé en panne.

La deuxième solution a été choisie pour la première mise en œuvre. Parce que la boîte à outils (ISIS [ISIS 91]) n'offre pas de multicast uniforme. La première solution sera mise en œuvre grâce à la boîte à outils PHENIX [Malloth 94] qui offre un multicast uniforme. En terme de coût, il est trop tôt pour évaluer si cela vaut la peine ou non de l'employer. Des mesures de performance (e.g. temps nécessaire pour réaliser une requête) permettront de faire un choix définitif, entre favoriser la performance dans le cas général, et favoriser la sûreté dans un cas très particulier.

## 5.3 Partition du réseau

Le traitement d'une partition du réseau n'est pas très différent d'une panne de duplica, si la partition minoritaire est considérée comme étant en panne, et que les duplicas la composant sont stoppés. Le cas de figure le plus problématique est celui où seuls les duplicas de la

partition minoritaire ont vu une écriture. Ce traitement entraîne la perte de certaines requêtes.

Cette solution est actuellement mise en œuvre, car ISIS [ISIS 91] ne permet pas de traiter autrement une partition minoritaire. De plus, elle est cohérente avec le choix fait pour traiter le problème précédent (panne d'un duplica). La deuxième solution est de considérer que les deux partitions peuvent coexister, et d'envisager un protocole de reconnexion. Cette seconde mise en œuvre, si elle s'avère utile, souhaitable et réalisable, pourra être mise en œuvre en utilisant PHENIX [Malloth 94].

# 6 Conclusion

Ce chapitre a montré qu'il était possible de mettre en œuvre de façon efficace un protocole assurant la NTR-linéarisabilité. L'originalité de ce protocole est l'efficacité des écritures, une synchronisation n'étant nécessaire que lors des lectures.

Le problème des pannes a été pris en considération, et deux propositions sont offertes suivant la qualité de service dont les utilisateurs ont réellement besoin. La première basée sur un multicast uniforme est la plus satisfaisante sur le plan théorique puisqu'elle empêche toute perte d'opérations. Sur le plan pratique cependant elle peut s'avérer coûteuse. La seconde n'empêche pas la perte d'informations mais est peu coûteuse.

Au niveau des mécanismes mis en œuvre, il est à noter que l'utilisateur n'est pas pénalisé par la duplication, puisque seule la première réponse est considérée, et que le reste du protocole peut s'effectuer en parallèle. Ainsi même si certains mécanismes peuvent s'avérer coûteux au niveau système, l'utilisateur n'est pas pénalisé.

Au niveau des structures de contrôle, le surcoût à payer pour la quantité d'informations échangées dans le système est proportionnel au nombre d'objets logiques, et non au nombre de duplicas. L'utilisateur n'est ainsi pas pénalisé par le taux de duplication.

Finalement, signalons les propriétés suivantes : (1) la duplication est transparente à l'utilisateur, y compris au niveau des performances, (2) le code des processus et des duplicas n'a pas besoin d'être modifié puisque seule une coquille servant d'interface est requise, (3) le protocole est non bloquant en cas de panne, (4) le temps de réponse est optimal pour les requêtes d'écriture, et aussi bon pour les lectures qu'un protocole utilisant des multicasts totalement ordonnés. Ce dernier point est particulièrement intéressant dans le cadre de DUPLEX puisque dans la pratique, les requêtes d'écriture sont plus nombreuses que les requêtes de lecture.

# Partie III

# Architecture

# Chapitre 6

# Présentation de l'environnement DUPLEX

## 1 Introduction

Un environnement d'édition coopérative est la combinaison de plusieurs outils offrant des facilités non seulement pour l'écriture conjointe de documents, mais encore pour l'échange d'informations entre les coauteurs (Cf. Chapitre 2) [Miles 93]. C'est pour cette raison que l'environnement DUPLEX intègre en plus d'un *éditeur coopératif* permettant de maintenir l'état d'un document partagé, des *facilités de communications* permettant l'échange d'informations entre les coauteurs dans le cadre de la coopération. Concernant ces communications, nous distinguons les *communications directes* où les destinataires d'un message sont explicitement spécifiés par l'émetteur, des *communications indirectes* où les destinataires d'un message sont ceux qui ont un intérêt pour un sujet particulier. Concernant les interactions, on distingue communément les *interactions synchrones* (où synchrone prend le sens de temps-réel) et les *interactions asynchrones* [Posner 92]. Lors d'une interaction synchrone, les modifications produites peuvent être observées en temps-réel par tous les membres de la collaboration. Dans le cas asynchrone, une divergence de vue est possible.

L'éditeur coopératif de DUPLEX offre des interactions asynchrones, puisque dans le contexte du travail coopératif à grande échelle, l'interaction entre utilisateurs travaillant dans des fuseaux horaires différents suggère plutôt ce mode opératoire. Cependant, en certaines occasions, les interactions entre les utilisateurs peuvent nécessiter un couplage plus fort. C'est pour cette raison que les facilités de communications directes ou indirectes supportent des interactions synchrones et asynchrones. Par exemple, si l'on considère les différentes phases dans l'élaboration d'un document (Cf. Chapitre 1), on peut remarquer que la phase initiale de planification du document, ou la phase finale de polissage, exigent des interactions

en temps-réel entre les auteurs. Cependant, il nous semble que ces interactions mettent plus volontiers à contribution des outils de communications directes ou indirectes intensives plutôt que des facilités d'édition en temps-réel. Une interaction synchrone, quand elle est nécessaire, est réalisée grâce aux facilités de communication directe ou indirecte, puis les résultats de ces interactions sont finalement fusionnés avec le document en utilisant l'éditeur coopératif. En restreignant ainsi l'éditeur coopératif à des interactions asynchrones, un environnement puissant peut être offert aux utilisateurs, sans pour autant les pénaliser par l'inhérente inefficacité de la communication dans un réseau à grande échelle.

Ce chapitre présente une vue d'ensemble de l'environnement d'édition coopérative DUPLEX. La section 2 décrit les modes opératoires offerts par cet environnement. La section 3 présente le système distribué sur lequel DUPLEX peut fonctionner, ainsi que les contraintes introduites par un tel système distribué. Finalement la section 4 conclut par les spécificités de DUPLEX qui seront développées dans les chapitres suivants.

# 2 Mode opératoire dans DUPLEX

L'environnement DUPLEX se compose de deux types d'entités : le noyau et l'environnement d'un utilisateur. Deux types d'interactions sont donc à prendre en compte : (1) l'interaction entre un utilisateur et le noyau, et (2) l'interaction entre deux utilisateurs (i.e., les communications).

## 2.1 Interaction entre l'environnement d'un utilisateur et le noyau

Le *noyau* est partagé par tous les partenaires de la collaboration, il pourvoit à la persistance et à la disponibilité de l'état le plus récent du document. *L'environnement d'un utilisateur*, en revanche, offre à l'utilisateur un espace de stockage indépendant et autonome.

Un document est décomposé en parties indépendantes et chacune d'entre elles constitue ce que nous appelons un *objet* dans la suite de cette partie.

On distingue les *objets du noyau* qui sont accessibles par tous les utilisateurs, des *objet locaux* qui sont des copies locales, stockées sur le site de l'utilisateur, qui ne sont accessibles que par lui. Un objet du noyau est dupliqué (au sein du noyau) afin d'assurer sa disponibilité, et d'offrir un meilleur accès. Une copie locale n'est pas maintenue comme un duplica d'un objet du noyau, mais comme une copie propre à l'utilisateur.

L'ensemble des objets locaux contenus dans l'environnement d'un utilisateur constitue sa *vue* locale du document. L'utilisateur travaille localement à partir des objets de sa vue; il est déconnecté du noyau jusqu'à ce que (1) une copie d'un objet noyau soit exigée pour continuer le travail (elle est alors copiée dans la vue locale à partir du noyau), ou (2) l'utilisateur

considère qu'un objet de sa vue locale peut être partagé (l'objet noyau correspondant est mis à jour à partir de la copie locale). Ainsi les utilisateurs travaillent indépendamment dans leur environnement local, et n'interagissent avec le noyau que lorsque cela est nécessaire.

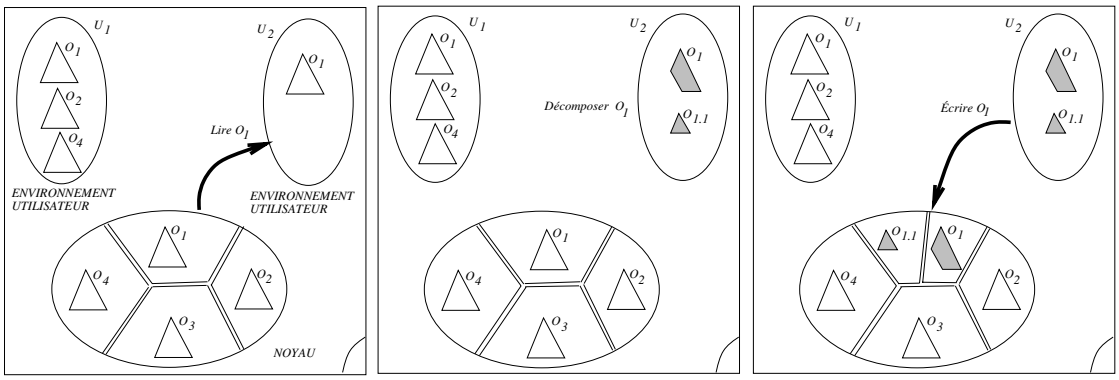


Figure 6.1 : Représentation logique du noyau et d'un environnement utilisateur

La figure 6.1 illustre ce mode d'interaction. La vue du document pour l'utilisateur $U_1$ est composée des copies des objets $O_1$, $O_2$, $O_4$. L'utilisateur $U_2$, désirant travailler sur l'objet $O_1$, copie l'objet depuis le noyau dans son environnement local (fig. 6.1-a). L'utilisateur $U_2$ modifie l'objet $O_1$ en le décomposant en deux objets locaux $O_1$ et $O_{1.1}$ (fig. 6.1-b). La décomposition du document est décrite plus en détail dans la suite de ce document (Cf. Chapitre 7). Pour le propos qui nous intéresse ici, admettons que l'objet $O_1$ est simplement divisé en deux parties. Finalement $U_2$ met à jour $O_1$ au niveau du noyau mettant ainsi à disposition les deux objets partagés $O_1$ et $O_{1.1}$ (fig. 6.1-c).

La vue locale d'un utilisateur peut donc différer de l'état réel du noyau de plusieurs façons. Premièrement, elle peut être incomplète dans le sens où certains objets du noyau sont absents de la vue de l'utilisateur (e.g., la vue de $U_1$ ne contient pas $O_3$). Deuxièmement, un utilisateur peut localement modifier les copies des objets (e.g., vue de $U_2$ dans la fig. 6.1-b). Finalement, une modification de l'état du noyau peut rendre obsolètes les vues locales des autres utilisateurs (e.g., vue de $U_1$ dans fig. 6.1-c).

Les approches décrites dans la littérature pour le partage du document sont soit basées sur un stockage centralisé (MultimETH [Lubich 90], Prep [Neuwirth 90], Quilt [Fish 88, Leland 88] ou Mule [Pendergast 90]), soit à travers une duplication locale des données sur le site de chaque utilisateur (Iris [Borghoff 93] ou CES [Greif 92]). DUPLEX propose une solution hybride permettant les avantages de chacune de ces deux approches :

> Le noyau dupliqué maintient les informations partagées indépendamment de la localisation des utilisateurs, afin de ne pas pénaliser l'ensemble des utilisateurs à cause des mauvaises performances matérielles du site de l'un d'entre eux. Les modifications des données sont effectuées localement sur le site de chaque utilisateur, et le contrôle de concurrence intervient, à la demande de l'utilisateur, lors de la mise à jour du noyau et

non automatiquement à chaque modification des données locales. La notion de copie locale, et les politiques de traitement de ces dernières, permettent d'optimiser le contrôle de concurrence et le temps de réponse. Autoriser une divergence entre l'état du noyau et les vues locales est adapté aux interactions asynchrones entre les utilisateurs dans un environnement à grande échelle.

## 2.2 Communication entre utilisateurs

La partie communication est une partie où certains outils déjà existants ont été mis à profit. En effet, certains outils correspondaient exactement à nos besoins et avaient le mérite d'être disponibles, efficaces et facilement intégrables dans l'environnement DUPLEX.

Les différents types de communication sont mis en œuvre ainsi :

**communication directe et indirecte synchrone :** fournie par l'outil Ytalk[1] qui permet d'échanger en temps-réel des messages entre plusieurs utilisateurs connectés sur Internet. Brièvement, chaque utilisateur impliqué dans la communication possède sur son écran (1) une fenêtre de dialogue par interlocuteur où apparaissent les messages tapés par ce dernier, et (2) une fenêtre dans laquelle il peut taper un message à destination des autres utilisateurs. Dans le cas d'une communication indirecte, la liste des utilisateurs impliqués est définie par DUPLEX et correspond par exemple aux utilisateurs travaillant sur la même partie du document. Dans le cas d'une communication directe, cette liste est explicitement donnée par l'utilisateur qui est à l'origine de la discussion.

**communication directe asynchrone :** basée sur les fonctionnalités offertes par le courrier électronique (i.e., E-Mail) disponible en standard sur Internet. Une première interface permet à l'émission (1) d'associer à un message une date de péremption afin qu'il ne perdure pas plus que nécessaire, (2) d'associer des informations causales utiles pour la gestion de la cohérence, (3) de qualifier un message (e.g. urgent, avertissement, normal, etc.), et (4) de définir le ou les destinataires d'une manière plus conviviale que les adresses de courrier électronique classiques. Une seconde interface permet à la réception (1) de traiter automatiquement certains messages, (2) de réordonner les messages (restitution de la causalité), et (3) de trier les messages par leur qualifiant.

**communication indirecte asynchrone :** forum semblable aux News sur Internet. Le sujet est une partie de document, et chaque personne travaillant sur cette partie a accès au forum. Ces communications permettent par exemple d'apporter des propositions de corrections et de révisions.

[1] Logiciel écrit par B. Yenne et disponible en *freeware*. ytalk@austin.eds.com

# 3 Le modèle de système distribué

Le système distribué que nous considérons (schématisé dans la figure 6.2) est composé d'un ensemble de sites dans un réseau à grande échelle. Un site est soit la station de travail d'un utilisateur (étiqueté U), soit un site noyau (étiqueté N) participant au stockage des données correspondant à une partie du document, soit les deux.

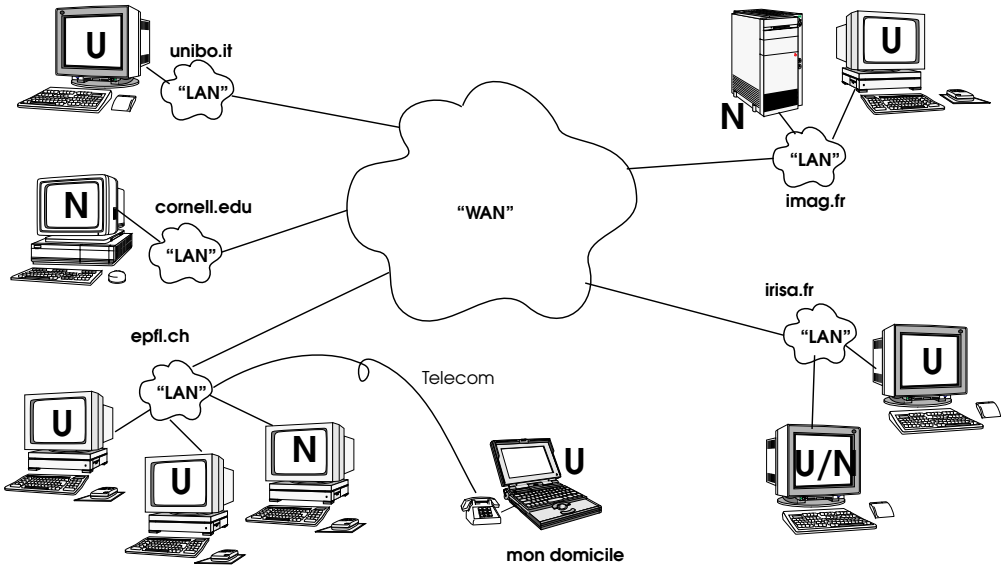


Figure 6.2 : Exemple de système distribué cible

Chaque équipe impliquée dans une coopération met à disposition un ou plusieurs sites pouvant offrir les fonctionnalités du noyau. Ces sites sont choisis en fonction des performances du réseau qui les relie. Ainsi, des domaines n'ayant pas de performances acceptables n'auront pas de site noyau (e.g., unibo.it sur la figure 6.2) et en revanche des domaines ayant un réseau de bonne qualité peuvent fournir un site noyau bien qu'ils n'aient pas d'utilisateurs impliqués (e.g. inesc.pt sur la figure 6.2).

L'ensemble des sites évolue dynamiquement au fur et à mesure que les utilisateurs rejoignent ou quittent la collaboration, ou que des parties du document sont ajoutées ou enlevées. Les sites que nous considérons, dans notre modèle, sont ceux qui sont impliqués par l'édition d'un même document. Les ensembles de sites correspondant à différents documents peuvent être disjoints ou bien présenter une intersection non vide : ils sont gérés comme des systèmes distribués indépendants.

La sémantique de panne d'un site que nous considérons est de type "*crash-failure*" [Cristian 91] (i.e., toutes les activités du site, y compris la communication, cessent lors d'une panne). La communication entre les sites est point-à-point, asynchrone (délais de transmission non bornés) et non fiable. En particulier, les liens de communication peuvent être inopérants et le réseau peut être partitionné en sous-réseaux déconnectés. Plus précisément, nous con-

sidérons un système dont les sites sont connectés à travers Internet, ce qui constitue, dans l'état actuel de la technologie, le plus grand réseau possible.

L'intérêt d'une telle architecture est qu'un utilisateur n'est pas gêné par les mauvaises performances réseau d'un autre utilisateur. Par exemple, considérons un ordinateur portable branché sur le réseau Internet par une ligne offrant de piètres performances (i.e., modem + ligne téléphonique + protocole SLIP [Romkey 88]). Si cet ordinateur devait maintenir un objet du noyau, tous les utilisateurs en seraient pénalisés. En offrant un noyau avec de bonnes performances, seul l'utilisateur utilisant ce portable est pénalisé par les mauvaises communications.

Le deuxième intérêt d'une telle architecture est que les coûts d'utilisation sont réduits puisque l'utilisateur du portable ne sera effectivement connecté au réseau que pour les opérations de transfert. En plus de ce mode de fonctionnement domestique de plus en plus fréquent, il faut noter que bon nombre d'entreprises disposent d'un réseau Ethernet local, mais dépendent de passerelles coûteuses pour se relier à Internet ou à d'autres succursales distantes. Cette liaison est généralement non permanente car elle utilise un support de type PTT qui est facturé à la durée et non au débit.

## 4 Spécificité de l'environnement DUPLEX

L'environnement DUPLEX propose trois concepts fondamentaux : (1) une décomposition dynamique du document conduite par les utilisateurs qui permet d'agir sur le contrôle de concurrence et les contraintes imposées pour la cohérence, (2) un noyau de stockage fiable distribué sur un réseau à grande échelle, et (3) un environnement utilisateur riche permettant un travail local la plupart du temps, et une interaction avec le noyau réduite au minimum.

Ces trois concepts sont spécifiques et originaux dans le contexte de l'édition coopérative. Ils font l'objet d'un développement plus approfondi dans les chapitres suivants.

# Chapitre 7

# Exploitation de la décomposition du document

## 1 Introduction

Le chapitre 5 de ce manuscrit présente le contrôle de la cohérence dans l'environnement DUPLEX. L'idée maîtresse en ce qui concerne ce contrôle de cohérence est de laisser un maximum de liberté aux utilisateurs. En effet, les utilisateurs, avant l'apparition d'outils dédiés à l'édition coopérative, avaient déjà une manière de travailler en groupe. Le but de DUPLEX est de faciliter ce processus de coopération, et non d'imposer ses propres règles. Ainsi, ce chapitre a pour but de montrer comment les utilisateurs peuvent utiliser les différents mécanismes proposés par DUPLEX pour adapter l'environnement d'édition coopérative à leurs besoins.

Comme il a été précédemment écrit, le contrôle de concurrence dans DUPLEX est basé sur (1) une décomposition dynamique du document en parties indépendantes qui permet de réduire les conflits sur les objets du noyau, (2) un critère de cohérence permettant de garantir la cohérence du document grâce à une approche basée sur un traitement local et indépendant de chacune de ces parties, et (3) un ensemble de politiques de contrôle de concurrence (optimiste, pessimiste, hybride) offrant à l'utilisateur un large choix de méthodes de collaboration pour assurer l'atomicité des opérations du type lecture-modification-écriture.

La combinaison de ces trois aspects permet de simplifier le contrôle nécessaire pour assurer la cohérence des données partagées, puisqu'elle permet une approche modulaire (au niveau de chacune des parties du document) d'un problème qui s'avère coûteux dans un système à grande échelle si l'on considère une approche globale. En outre, il est possible d'adapter le contrôle de concurrence en fonction de la cohérence réellement nécessaire par (1) une modification dynamique de la décomposition du document qui modifie la granularité des objets

partagés, et (2) une gestion par objets du contrôle de concurrence qui permet d'adapter le mode de coopération en fonction de l'état du document, la partie du document concernée, ou encore des personnes impliquées dans sa réalisation.

La section 2 présente en détail le modèle utilisé pour la décomposition du document. La section 3 présente comment les opérations de base de DUPLEX (à savoir écriture et lecture pour l'édition, et séparation et regroupement pour la décomposition du document) peuvent être réalisées de façon optimiste en accord avec le critère de cohérence choisi. La section 4 conclut ce chapitre en mettant en évidence les avantages de notre approche par rapport à celles de la littérature.

# 2 Décomposition du document

## 2.1 Motivations

Considérons les opérations de base de DUPLEX : lecture, écriture, séparation et regroupement. Si on gère le document comme un objet unique, chaque paire d'opérations peut entrer en conflit, soit potentiellement $\frac{N(N-1)}{2}$ conflits à gérer pour $N$ opérations. En effet, $N$ opérations sur une partie peuvent entrer en conflit avec $N-1$ autres. D'où à cause de la symétrie du problème : $\frac{N(N-1)}{2}$ conflits potentiels. En revanche, si le document est décomposé en $k$ parties ($k \ll N$) et que l'on considère que la probabilité d'accéder à chacune de ces parties est distribuée uniformément, le nombre des conflits tombe à $\frac{N(N-k)}{2k}$. S'il y a équirépartition des opérations sur une des $k$ parties il y aura $\frac{N}{k}$ opérations qui entreront en conflit avec $\frac{N}{k}-1$ autres, soit $\frac{N}{k}\frac{N-k}{k}\frac{1}{2}k$ à cause de la symétrie du problème et du nombre de parties $k$, d'où le résultat $\frac{N(N-k)}{2k}$. L'hypothèse de la répartition uniforme se justifie puisque l'intérêt de la décomposition est justement de réaliser une distribution des opérations.

## 2.2 Décomposition du document

La décomposition du document dans DUPLEX est dynamique et dirigée par les auteurs eux-mêmes, puisqu'ils sont probablement les plus aptes à connaître les conflits potentiels. En effet, dans une coopération, une multitude d'informations annexes au document ne peuvent être prises en compte que par les coauteurs, puisque par essence ces informations ne transparaissent pas dans le document. Par exemple, les coauteurs peuvent déterminer, en fonction de l'état présent du document, les endroits où les conflits potentiels peuvent, ou au contraire ne peuvent pas être évités. Le mécanisme de décomposition de DUPLEX assure que le partitionnement (opérations séparation et regroupement) sera exécuté d'une manière cohérente vis-à-vis de l'édition (opérations lecture et écriture).

La décomposition est basée sur la structure hiérarchique du document : un document est

composé de parties, composées de chapitres, sections, sous-sections, sous-sous-sections, ... , paragraphes, figures, ...

Même si certains de ces éléments peuvent être absents de la hiérarchie, tout document possède une structure hiérarchique intrinsèque, (e.g., si le document est un livre, la présence de parties et chapitres se justifie; dans le cas d'un article la hiérarchie commence généralement au niveau des sections).

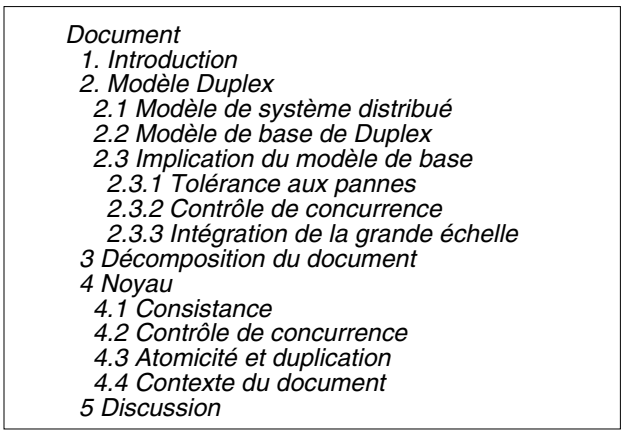


Figure 7.1 : Table des matières de document servant d'exemple

La table des matières d'un document est une bonne approximation de sa structure hiérarchique même si seulement une sous-partie de la hiérarchie y apparaît. Nous utiliserons le document dont la table des matières est représentée dans la figure 7.1 pour illustrer les exemples dans la suite de ce chapitre.

Du point de vue hiérarchique, le document est un segment de *niveau* 0 composé de plusieurs segments de niveau 1 disjoints, qui peut être modélisé comme une expression parenthésée bien formée (Cf. Figure 7.2) où l'index de la parenthèse ouvrante représente le niveau du segment dans la hiérarchie du document.

$$\{_0\{_1 1\ \textsc{Introduction}\}\ \{_1 2\ \textsc{Modèle Duplex}\}\ \{_1 3\ \textsc{Décomposition du document}\}$$
$$\{_1 4\ \textsc{Noyau}\}\ \{_1 5\ \textsc{Discussion}\}\}$$

Figure 7.2 : Expression parenthésée bien formée définissant un document

A son tour, chaque segment de niveau 1 est lui-même une séquence bien formée de segments de niveau 2. Dans notre exemple, le segment 2 Modèle Duplex de niveau 1 est une séquence composée des segments de niveau 2 suivants : 2.1 Modèle de système distribué, 2.2 Modèle de base de Duplex, 2.3 Implication du modèle de base. Il en est ainsi, récursivement jusqu'aux segments du document non structurés comme par exemple une phrase,

un paragraphe non numéroté, ou encore une figure. Ces segments sont appelés segments *élémentaires* et ne peuvent plus être à leur tour décomposés.

Nous définissons ci-dessous un certain nombre de notions associées aux segments.

> Définition 7.1 : *segment père*
> *Si un segment $S$ est de niveau $k$, son segment* père *est le segment de niveau $k-1$ qui contient $S$.*

> Définition 7.2 : *segment frère*
> *Si un segment $S$ est de niveau $k$, ses segments frères sont les segments de niveau $k$ contenus dans le segment père de $S$.*

> Définition 7.3 : *segment fils*
> *Si un segment $S$ est de niveau $k$, ses segments fils sont les segments de niveau supérieur à $k$ qu'il contient.*

Dans Duplex, chaque segment (y compris les segments élémentaires) peut être défini (1) comme un segment *indépendant* du document, et maintenu comme un objet du noyau, ou bien (2) comme un segment *dépendant* de son segment père, et inclus dans ce dernier.

Dans la figure 7.3 chaque rectangle représente un segment. Les segments dépendants sont représentés imbriqués dans leur segment père. Les segments indépendants de niveau $k$ sont représentés dans les colonnes correspondant à leur niveau dans la hiérarchie. Un segment peut être indépendant bien que son segment père ne le soit pas : par exemple le segment 4.2 Contrôle de concurrence est indépendant alors que son segment père 4 Noyau est dépendant du segment Document. De plus, un segment peut être indépendant alors que ses segments frères ne le sont pas: par exemple, le segment 2 Modèle Duplex est indépendant alors que les autres segments frères 1 Introduction, 3 Décomposition du document, 4 Noyau et 5 Discussion sont dépendants du segment père Document.

Considérons à un instant donné une partition $\Pi$ du document suivant ce type de frontière structurelle; $\Pi = \{S_i | S_i \textit{ est un segment indépendant}\}$. Une description complète d'un segment indépendant $S_i$ de niveau $k$ est fournie par : (1) une description complète des segments fils dépendants de $S_i$, (2) la position relative dans $S_i$ des segments fils dépendants et indépendants, et (3) les références aux segments fils indépendants. Par exemple, la description complète du segment indépendant 2 Modèle Duplex est fournie par : (1) la description complète de 2.2 Modèle de base de Duplex, 2.3 Implication du modèle de base, où la description de 2.3 Implication du modèle de base fait apparaître les segments dépendants 2.3.1 Tolérance aux pannes, 2.3.2 Contrôle de concurrence et 2.3.3 Intégration de

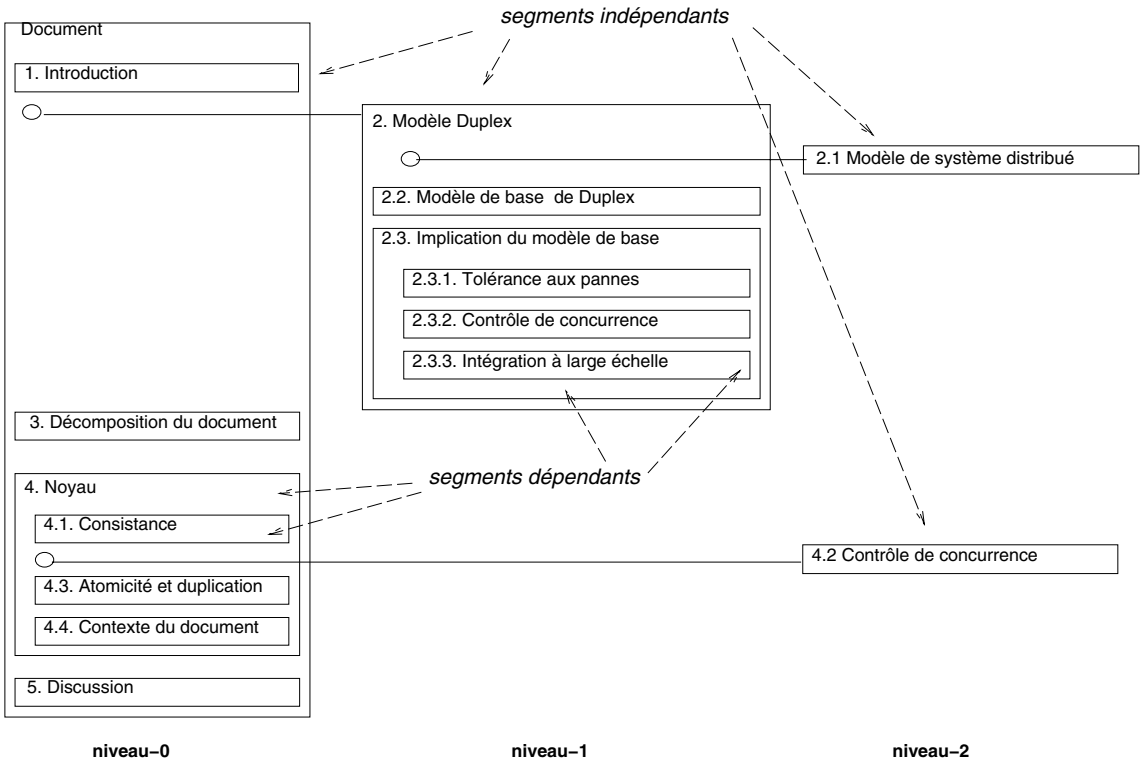


Figure 7.3 : Exemple de décomposition

la grande échelle, (2) la position de 2.1 Modèle de système distribué, 2.2 Modèle de base de Duplex, 2.3 Implication du modèle de base dans 2 Modèle Duplex, à savoir l'ordre dans lesquelles ces sections apparaissent dans le texte, et (3) la référence du segment indépendant 2.1 Modèle de système distribué.

Deux écritures sur $S_i$ sont sujettes au contrôle de cohérence, car elles représentent un conflit potentiel. Par contre, une écriture sur $S_i$ n'entre pas en conflit avec une écriture sur un autre segment indépendant du document (y compris les segments fils indépendants de $S_i$).

## 2.3 Aspect dynamique de la décomposition

En permettant une décomposition dynamique du document, il est possible d'autoriser plus ou moins de concurrence en fonction de l'état courant du document. En effet, dans les phases initiales de rédaction, il est souhaitable d'augmenter la concurrence afin de permettre aux différents coauteurs impliqués d'apporter leurs idées de façon informelle. Des contraintes fortes au niveau du contrôle de concurrence peuvent nuire à la bonne marche de la collaboration. Au fur et à mesure que le document mûrit, il convient de diminuer la concurrence en figeant certaines parties de celui-ci. Ceci est réalisé en augmentant la taille des objets indépendants par des opérations de regroupement.

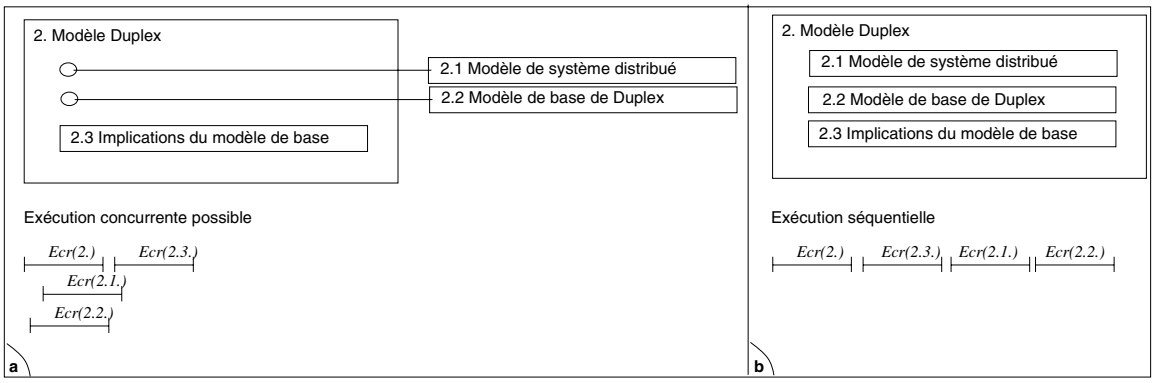


Figure 7.4 : Adaptation du contrôle de concurrence par modification du grain

Par exemple, considérons la cohérence séquentielle comme critère de cohérence dans les exemples de la figure 7.4. Le critère de cohérence est appliqué à chacun des objets indépendants, et a pour effet de séquentialiser les opérations effectuées sur un objet. Considérons la figure 7.4-a : les deux sous-sections 2.1 Modèle de système distribué et 2.2 Modèle de base de Duplex de la section 2 Modèle Duplex sont indépendantes, alors que la sous-section 2.3 Implication du modèle de base est dépendante de la section 2 Modèle Duplex. Supposons qu'une écriture soit réalisée sur chacun de ces segments. Il est possible de réaliser concurremment une écriture sur 2.1 Modèle de système distribué, et une sur 2.2 Modèle de base de Duplex, ou encore, une écriture sur 2 Modèle Duplex et une sur 2.1 Modèle de système distribué. Par contre les écritures sur 2 Modèle Duplex et 2.3 Implication du modèle de base devront être séquentialisées. Ce qui rend l'histoire de la figure 7.4-a acceptable pour l'utilisateur.

En revanche si 2.1 Modèle de système distribué et 2.2 Modèle de base Duplex sont dépendants de la section 2 Modèle Duplex comme dans la figure 7.4-b, alors toutes les écritures seront séquentialisées au niveau de la section 2 Modèle Duplex, ce qui impose une histoire séquentielle comme celle de la figure 7.4-b.

Augmenter le nombre de parties permet d'augmenter le nombre d'histoires possibles pour un ensemble d'opérations et diminue les contraintes d'ordonnancement entre les opérations d'écriture.

# 3 Opérations de décomposition du document

Deux opérations sont considérées pour la décomposition du document : (1) la séparation qui consiste à rendre un segment indépendant, et (2) le regroupement qui consiste à rendre dépendant un segment préalablement indépendant. Ces opérations mettent en œuvre deux objets et peuvent se dérouler concurremment à des opérations de lecture et d'écriture.

L'atomicité des opérations de regroupement et séparation doit être garantie afin de ne pas violer le critère de cohérence choisi.

Considérant les opérations de séparation et de regroupement, deux problèmes sont à résoudre : (1) l'atomicité aux pannes (l'opération est réalisée complètement ou pas du tout) et, (2) l'atomicité de concurrence (les effets de l'opération ne sont visibles qu'une fois l'opération terminée).

## 3.1 Opération de séparation

L'objet correspondant au segment initial est noté $P$ (pour père) et l'objet correspondant au segment qui est rendu indépendant est noté $F$ (pour fils).

L'état initial du noyau et de l'environnement de l'utilisateur est donné par la figure 7.5-a. Le contenu des objets est donné (quand nécessaire) en haut à droite de chacun des cadres bordant les 3 figures 7.5-a-b-c.

Dans son état initial l'objet $P$ est la section 2 Modèle Duplex qui contient trois segments dépendants : les sous-sections 2.1 Modèle de système distribué, 2.2 Modèle de base de Duplex et 2.3 Implication du modèle de base. L'opération de séparation va consister à extraire la sous-section 2.1 Modèle de système distribué qui deviendra l'objet $F$.

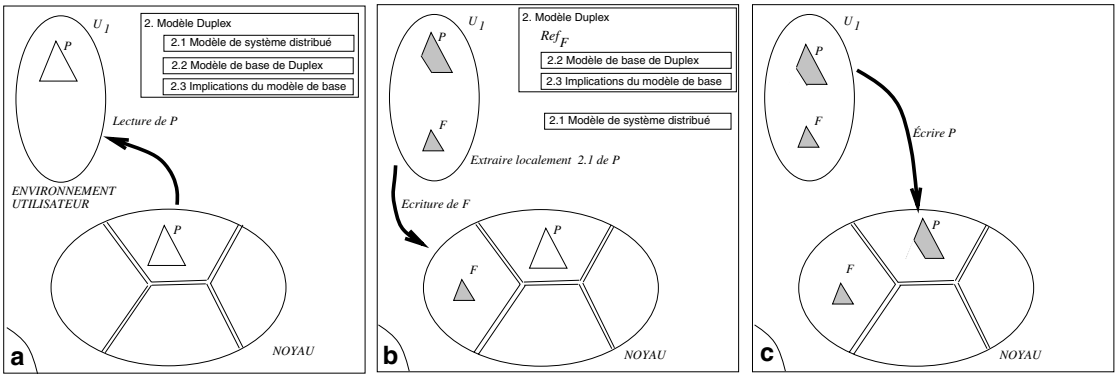


Figure 7.5 : Opération de séparation

Une propriété de l'opération de séparation est que le nouvel objet $F$ ne sera connu de la communauté que par l'intermédiaire de l'objet $P$ qui contient la référence vers $F$. Ainsi, comme nous allons le voir, une opération de séparation peut être réalisée localement au niveau de l'environnement de l'utilisateur, et réduite à une écriture de l'objet $P$ comme décrit ci-dessous :

Une lecture de l'objet $P$ est réalisée par l'utilisateur voulant procéder à une opération de séparation. Une fois dans l'environnement local de l'utilisateur l'objet $P$ va être modifié. Un nouvel objet $F$ est créé. Le contenu du segment 2.1 Modèle de système distribué est copié

**var**
    $seg$ : segment à rendre indépendant
–
lecture de l'objet noyau $P$;
création de l'objet $F$ localement;
copie du contenu du segment $seg$ de $P$ dans l'objet local $F$;
remplacement du segment $seg$ de $P$ par une référence sur l'objet $F$ ($Ref_F$);
création de l'objet $F$ au niveau du noyau;
écriture l'objet local $F$ au niveau du noyau;
écriture l'objet local $P$ au niveau du noyau;

Algorithme 7.1 : Séparation d'un segment en deux segments indépendants

dans l'objet local correspondant au nouvel objet $F$. Le contenu du segment 2.1 MODÈLE DE SYSTÈME DISTRIBUÉ à l'intérieur de l'objet $P$ est remplacé par la référence à l'objet $F$ (noté $Ref_F$ dans la figure 7.5-b). Bien que $F$ soit connu du système, il ne peut être utilisé par les autres utilisateurs car la seule référence à l'objet $F$ est dans la version locale de l'objet $P$. L'objet $F$ peut être transféré au niveau du noyau (figure 7.5-b) sans risquer d'être utilisé par un autre utilisateur. Ceci garantit l'atomicité aux pannes et à la concurrence, car toute l'opération de séparation est finalement réduite à l'opération d'écriture de $P$ (figure 7.5-c).

A ce niveau-là, si une approche pessimiste a été choisie par l'utilisateur en se réservant une écriture exclusive[1]) de $P$, garantissant que $P$ ne peut être écrit par un autre utilisateur, avant de procéder à la séparation, l'opération écriture sur $P$ est sûre d'aboutir. En revanche, si une approche optimiste a été choisie avec une détection de conflit, alors deux cas de figure sont à considérer :

- l'écriture de $P$ réussit (opération non conflictuelle), et au niveau du noyau l'opération de séparation sera validée puisque $P$ peut être lu par les autres utilisateurs qui découvrent ainsi l'existence de $F$.

- l'écriture échoue, et au niveau du noyau l'opération de séparation ne sera pas validée et l'objet $P$ du noyau n'aura aucune trace de $F$. La trace de $F$ ne subsiste que dans l'environnement local de l'utilisateur initiateur de l'opération de séparation.

Il est à noter que cette méthode permet de rendre indépendant localement plusieurs segments avant de répercuter les effets au niveau du noyau. Ceci est très important car cela permet une décomposition très rapide du document.

Les deux problèmes suivants restent à résoudre :

(1) Le premier problème est celui de la destruction de l'objet $F$ en cas d'échec de l'opération d'écriture finale. En effet, plus aucune référence ne pointe sur l'objet $F$ qui devient ainsi

[1]Grâce au mécanisme de capacité, (Cf. Chapitre 8

inaccessible, et donc monopolise des ressources pour rien. Plusieurs méthodes sont utilisables : la première est basée sur une requête de destruction de $F$ explicite émanant du client $F$. Cette solution n'est pas optimale, puisqu'en cas de panne du client, cette requête peut ne jamais être émise. La seconde consiste en l'utilisation d'un ramasse miettes dont la fonction est de détruire les objets qui ne sont plus référencés. On peut utiliser ici un algorithme basé sur un marquage des objets accessibles en partant de la racine du document. Les objets marqués ne doivent pas être détruits, les autres peuvent éventuellement l'être après une vérification plus poussée. La dernière solution est celle que nous avons retenue, elle utilise un mécanisme d'activation/passivation. Elle consiste à désactiver un objet non utilisé, et permet de réduire les ressources consommées à un minimum. Par exemple, en mode passif, un objet ne consomme qu'un fichier par duplica et aucune ressource CPU. Le mécanisme d'activation/passivation de DUPLEX est décrit dans le chapitre concernant le noyau. Les données stockées sur disque seront libérées dans la phase finale lorsque le document est achevé et que la collaboration prend fin.

(2) Le deuxième problème est celui de l'unicité du nom logique du nouvel objet $F$ créé. En effet, au niveau de la décomposition du document, chacun des objets indépendants se voit associer un nom logique qui permet aux utilisateurs de l'identifier. Dans l'exemple précédent, $P$ et $F$ peuvent être considérés comme les noms logiques des segments indépendants section 2 MODÈLE DUPLEX et sous-section 2.1 MODÈLE DE SYSTÈME DISTRIBUÉ. Un nom logique doit avoir un sens pour les utilisateurs mais également être unique pour le système, afin de désigner sans ambiguïté le segment correspondant. Ceci est réalisé par la concaténation au nom donné par le créateur de l'objet (le responsable de l'opération de séparation) d'une identification assurant l'unicité, par exemple : <identité du créateur,date,heure>.

## 3.2 Opération de regroupement

L'opération de regroupement présente plus de problèmes puisque les deux objets $P$ (père) et $F$ (fils) sont potentiellement manipulables par les membres de la collaboration pendant la durée de l'opération de regroupement.

La solution retenue est de faire exécuter l'opération de regroupement par l'objet père $P$ au niveau du noyau.

La première garantie est que l'objet $P$ étant un objet dupliqué, l'atomicité aux pannes est garantie puisque l'objet $P$ n'est pas sensible aux pannes.

L'atomicité de concurrence est réalisée de la manière suivante (Cf. Alg. 7.2) : (1) l'objet $P$ n'accepte aucune autre requête durant l'opération de regroupement (ce qui est le fonctionnement standard lors du traitement d'une requête quelconque), (2) l'objet $P$ va se réserver

**envoyer** "requête d'accès exclusif" à $F$;
**si** requête acceptée
**alors**
    lecture de $F$;
    remplacement de $Ref_F$ par le contenu de $F$;
    destruction de $F$;
    envoi de $P$ au client;
**sinon**
    notification de l'échec au client;
**fin si**

Algorithme 7.2 : Regroupement de deux segments

un accès exclusif[2] sur l'objet $F$, garantissant que $F$ ne peut être ni lu, ni a fortiori modifié, par un utilisateur.

Le succès de l'opération de regroupement est alors réduit au succès de la demande d'accès exclusif à $F$.

Après obtention de l'accès exclusif à $F$, l'objet $P$ peut alors lire l'objet $F$ en étant sûr d'avoir une version correcte, et remplacer la référence à $Ref_F$. Finalement $P$ détruit l'objet $F$ et envoie la nouvelle version de $P$ au client.

# 4 Discussion

Ce chapitre a décrit les règles de décomposition dynamique d'un document dans DUPLEX. Cette approche permet (1) de définir un grain variable pour les objets partagés, et ainsi d'améliorer la concurrence au niveau des accès (Cf. Chapitre 2, remarques 1 et 2), et (2) de faire évoluer les contraintes liées au critère de cohérence en fonction de la maturité du document.

La décomposition d'un document en parties indépendantes est un procédé utilisé par d'autres environnements (e.g. Iris [Borghoff 93], Griffon [Decouchant 93],CES [Greif 92], Quilt [Fish 88, Leland 88] SharedBook [Lewis 88] et MultimETH [Lubich 90]). La plupart de ces environnements proposent une décomposition statique (e.g. SharedBook ou Quilt), ou une décomposition dynamique Iris, CES, Griffon, et MultimETH, sur laquelle l'utilisateur ne peut pas intervenir directement. Par exemple, la décomposition dans Griffon est liée au mécanisme de rôle, dans le sens où un segment indépendant est un segment pour lequel le rôle de chacun des utilisateurs est uniforme. Quand un utilisateur modifie son rôle sur une partie d'un segment indépendant, la décomposition est automatiquement recalculée pour satisfaire la contrainte citée plus haut. La taille des segments indépendants est induite par les rôles des utilisateurs mais à l'insu de ces derniers. Il est difficile dès lors d'utiliser ce mécanisme pour

[2]Grâce au mécanisme de capacité, (Cf. Chapitre 8)

agir sur le contrôle de concurrence. Notre approche en ce sens est originale, puisque les utilisateurs peuvent agir directement sur la taille des parties indépendantes, et ainsi moduler les contraintes liées à la cohérence du document entier.

Finalement, signalons que toutes les opérations, y compris celles modifiant la décomposition du document, peuvent être réalisées de façon non bloquante, et cela sans mettre en défaut le critère de cohérence choisi. Ceci est très important dans un environnement qui, compte tenu de la dispersion géographique des utilisateurs, ne peut être qu'asynchrone, et très sensible aux performances des communications et des pannes.

# Chapitre 8

# Le Noyau

## 1 Introduction

Le noyau est la clé de voûte de l'environnement DUPLEX puisque c'est lui qui permet d'offrir et de contrôler l'accès aux informations relatives à un document entre les différents utilisateurs impliqués dans la collaboration.

La responsabilité du noyau est (1) de maintenir la cohérence globale du document décomposé, (2) d'assurer l'accès au document ainsi qu'aux informations qui y sont relatives et (3) d'offrir l'infrastructure nécessaire aux politiques de contrôle de concurrence. Le premier point a été longuement abordé dans les chapitres précédents concernant le problème de la cohérence. En revanche, les deux derniers points seront détaillés ci-dessous dans les section 2 et 3.

## 2 Description logique du noyau

Comme nous l'avons vu dans le chapitre 6, les partenaires d'une coopération mettent à disposition de la communauté un certain nombre de sites pouvant abriter les fonctionnalités du noyau. L'ensemble des *sites du noyau* peut évoluer au cours de la coopération, il est maintenu par le service d'activation décrit ci-dessous.

Pour assurer les différentes fonctionnalités qui lui incombent, le noyau utilise et offre un certain nombre de services :

**Le service d'objet :** C'est le service qui permet d'accéder à un objet du noyau. Afin de permettre une tolérance aux pannes, un service d'objet est un *groupe de duplicas* répartis sur différents sites du noyau. La duplication est active; chaque duplica joue le

même rôle : il reçoit une requête en provenance d'un utilisateur, la traite et y répond. Un service d'objet, comme nous le verrons, consomme des ressources même lorsqu'il n'est pas soumis à des requêtes en provenance des utilisateurs. Ainsi, il est utile qu'il puisse se désactiver lorsqu'il n'a pas été soumis à une requête pendant un certain laps de temps.

**Le service d'activation :** Le rôle premier de ce service est de pouvoir créer un duplica sur un site du noyau. Pour ce faire, il doit maintenir la liste $L_g$ de tous les sites mis à disposition, par la communauté, pour le noyau. Ce rôle de création est utilisé dans les trois tâches suivantes :

- **activation d'un service d'objet :** Un utilisateur ne peut pas accéder directement à un service désactivé. Il faut donc un mécanisme permettant de réactiver chacun des duplicas d'un groupe, afin que le service d'objet correspondant puisse être accédé par les utilisateurs.
- **régulation du taux de duplication :** Un duplica peut tomber en panne. Si l'on veut conserver un taux de duplication constant, il faut pouvoir mettre à disposition un duplica sur un autre site du noyau[1].
- **création d'un nouveau service d'objet :** Dans DUPLEX, l'ensemble des objets est dynamique en raison du processus de décomposition du document. Lorsqu'un nouvel objet est créé, il convient de créer un service d'objet correspondant. La destruction d'un objet, en revanche, n'est pas synonyme de destruction du service d'objet correspondant, puisque certaines informations telles que le journal (Cf. section 3.1) doivent toujours rester accessibles. Dans ce cas, il y a toutefois une réduction des informations stockées au niveau d'un service d'objet.

**Le service de noms :** Il constitue le moyen d'accéder aux services d'objet. Le service de noms dédié à un document permet de lier un segment logique du document (connu des auteurs, le nom étant établi à la création du segment) au service d'objet correspondant.

Nous détaillons ces différents services dans les sections suivantes.

# 3 Description des services

## 3.1 Service d'objet

Pour des raisons de tolérance aux pannes, les objets sont dupliqués, ainsi plusieurs duplicas du même objet existent. Un duplica est un objet actif constitué d'une partie donnée stockée

[1] Il est important de garder un taux de duplication constant pour éviter que des pannes en cascade finissent par rendre le service inopérant.

en mémoire ou sur un support stable (i.e., fichiers sur disque), et d'une partie active (i.e., un flot de contrôle). Un processus encapsule donc les données appartenant à l'objet, qui ne sont plus accessibles que par l'intermédiaire de requêtes qui lui sont adressées. Ce processus reçoit les requêtes, les traite et envoie une réponse en retour au requérant.

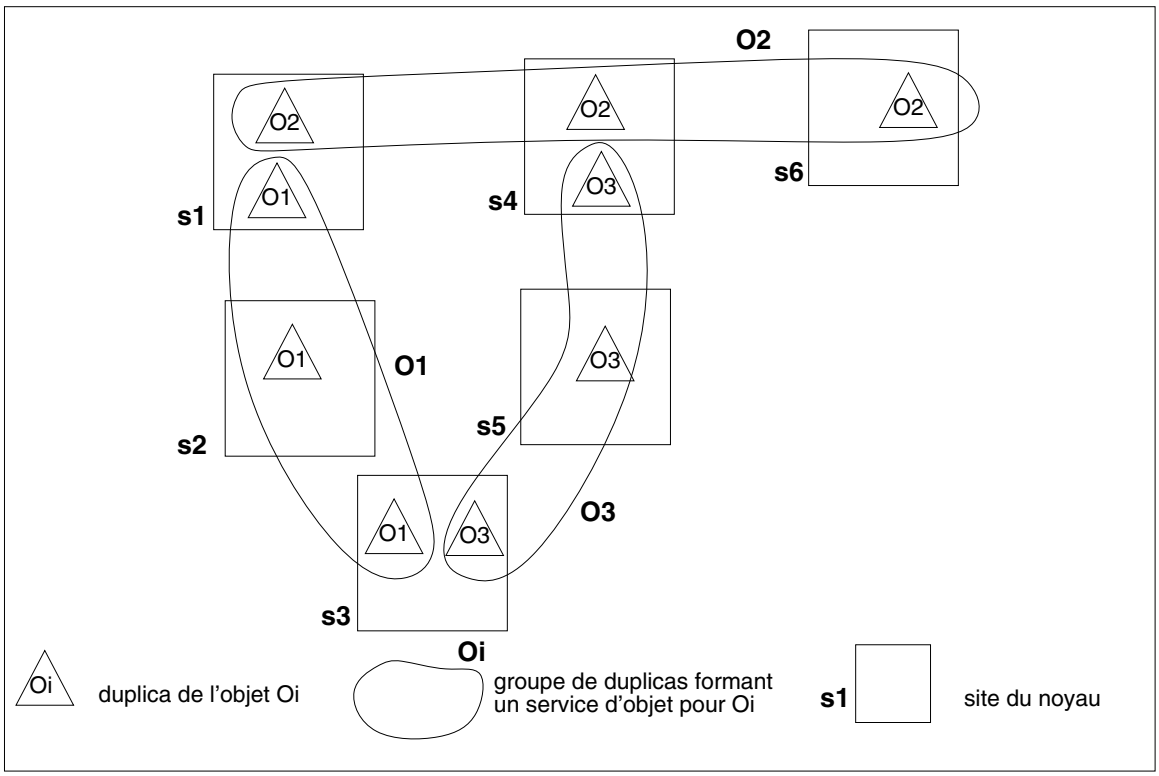


Figure 8.1 : Correspondance service d'objet, groupe de duplicas

Les duplicas d'un même objet sont regroupés au sein d'une même entité logique que nous appelons un *groupe*. Le service d'objet est mis en œuvre par un groupe de duplicas. Il y a une correspondance directe entre un service d'objet et un groupe de duplicas, et nous utiliserons indifféremment les deux termes dans la suite de ce chapitre. La figure 8.1 montre trois services d'objet O1, O2, O3 constitués chacun de trois duplicas. Ces duplicas sont répartis sur les différents sites du noyau (s1 à s6 ).

Un site peut accueillir des duplicas de plusieurs services d'objet et dans ce cas, les flots de contrôle des différents duplicas présents sur un même site sont gérés par un seul et même processus que nous appelons *processus gestionnaire d'objets* (PGO). Un processus gestionnaire d'objets "factorise" la partie active d'un duplica qui sert d'interface à plusieurs jeux de données (un par duplica). Un PGO appartient à autant de groupes que de duplicas qu'il gère.

Un jeu de données contient les informations suivantes :

**L'information textuelle :** Elle décrit complètement le segment du document. C'est par exemple une suite de caractères constituant le texte proprement dit, ou le *bitmap* d'une figure.

**L'information structurale :** Elle décrit la décomposition du segment en sous-segments dépendants et indépendants, et maintient les références logiques (par nom) des sous-segments indépendants[2].

**Le bulletin d'information :** Il assure les communications asynchrones indirectes concernant le segment. C'est une liste de messages (persistants ou de durée limitée) causalement ordonnée et cohérente avec le flot des discussions et les mises à jour des segments.

**Le journal :** Il garde une trace des opérations sensibles (mise à jour, destruction, etc.) accomplies sur le segment. Il survit à la destruction du segment et fournit en particulier la cause et le responsable de la destruction.

Le rôle du service d'objet est de permettre aux utilisateurs d'accéder à ces données. Pour ce faire, le service d'objet traite les requêtes en provenance des utilisateurs. Le problème de l'atomicité des requêtes dans le contexte dupliqué exige des *diffusions fiables* au *groupe* de duplicas dans un *modèle virtuellement synchrone* [Birman 93, Schiper 93]. La duplication est active, ce qui signifie que chaque duplica joue le même rôle : il reçoit la requête, la traite, puis envoie la réponse au requérant. Le protocole assurant la cohérence de ce groupe (i.e., l'ordre dans lequel les différents duplicas traitent les requêtes) est décrit dans le chapitre 5.

L'emplacement et le nombre des duplicas assurant le service d'objet sont dictés par deux paramètres : l'emplacement des utilisateurs et le taux optimal de duplication. L'emplacement des utilisateurs joue un rôle, car dans une duplication active tous les duplicas traitent la requête et envoient une réponse à l'utilisateur. Attendre la première réponse est suffisant, et ainsi placer un des duplicas à proximité d'un utilisateur travaillant fréquemment avec l'objet permet d'améliorer les performances. Cependant, il n'est pas possible de mettre un duplica à proximité de chaque utilisateur travaillant sur l'objet. En effet, le coût du protocole permettant de maintenir cohérent le groupe de duplicas est lié au nombre de duplicas. Ainsi, il faut éviter de dupliquer un objet plus que nécessaire[3]. En effet, il est rare que plusieurs sites distants tombent en panne au même moment : un petit nombre de duplicas (e.g. trois dans le cas présent) est suffisant pour assurer la tolérance aux pannes. Il convient toutefois, en cas de panne d'un site, de recréer un duplica sur un autre site, afin de conserver le taux de duplication constant.

[2] Ces références seront traduites en adresses dans le système distribué par le service de noms décrit ci-après.
[3] "entia non sunt multiplicanda praeter necessitatem" Guillaume d'Occam

## 3.2 Maintien du taux de duplication

Le maintien du taux de duplication fait intervenir deux services. Le service d'objet et le service d'activation. Le service d'objet détient deux informations : (1) la composition du groupe, et (2) les utilisateurs qui utilisent le plus l'objet.

---

**var**
    $L_a$ : liste des sites actifs
    *statistiques* : statistiques sur l'utilisation du service d'objet par les utilisateurs
–
**à la réception de** "la nouvelle composition est $L_a$";
**si** je suis le site de plus petit rang
**alors**
    **envoyer** une "requête de régulation" au service d'activation avec $L_a$ et *statistiques*;
**fin si**

---

Algorithme 8.1 : Régulation du taux de duplication

Grâce au modèle virtuellement synchrone, qui permet aux membres d'un groupe d'être informés de tout changement de composition du groupe, lorsqu'un duplica tombe en panne (ou plus naturellement quitte le groupe), les autres membres sont avertis de la nouvelle composition $L_a$ (pour liste des sites actifs). Les statistiques sur l'utilisation du service d'objet par les utilisateurs sont mises à jour à chaque requête. Le service d'activation pour sa part détient la liste des sites du noyau, et peut ainsi mettre à disposition un duplica sur un autre site. La partie concernant le service d'activation sera décrite dans l'algorithme Alg. 8.6. En ce qui concerne le service d'objet, le traitement est décrit dans l'algorithme Alg. 8.1.

La requête de régulation est envoyée au service d'activation par un seul duplica, celui qui est de plus petit rang (chaque membre d'un groupe a un rang qui est déterminé par le mécanisme de changement de vues). Si ce dernier tombe aussi en panne, cela sera détecté par les autres duplicas du groupe. Le nouveau duplica de plus petit rang prendra alors le relai, et les deux pannes seront traitées en même temps. Les statistiques contiennent les fréquences d'utilisation du service d'objet par les différents utilisateurs visant à faire ressortir les sites utilisateurs les plus actifs. Elles servent au service d'activation à déterminer le meilleur site de remplacement.

## 3.3 Migration de duplicas

Un utilisateur peut cesser à tout moment d'utiliser un service d'objet. Il peut donc être judicieux de rendre le placement des duplicas dynamique en permettant la migration d'un duplica d'un site $s_i$ vers un autre site $s_j$ plus adéquat. Le processus gestionnaire d'objets en cause peut simplement quitter le groupe de lui-même, provoquant le même traitement que lors d'une panne.

## 3.4 Désactivation

Lorsqu'un objet cesse d'être utilisé par l'ensemble des utilisateurs, les mécanismes de détection de pannes de duplicas et de régulation du taux de duplication continue de fonctionner, ce qui représente une consommation inutile des ressources. Pour cette raison, il est utile de passiver un service d'objet qui n'est plus utilisé. Passiver un objet consiste à réduire l'objet à son jeu de données sur disque.

L'ordre de désactivation est décidé par le service d'objet lui-même, puisqu'il est conscient de son oisiveté. Cela est réalisé de la manière suivante :

---

**si** dernière requête date de plus de $n$ minutes, et je suis le site de plus petit rang
**alors**
    **envoyer** "requête de désactivation" à mon groupe;
**fin si**

---

Algorithme 8.2 : Détection de l'inactivité

Tous les duplicas (y compris celui de plus petit rang qui a émis la requête) exécutent alors le protocole suivant :

---

**à la réception de** "requête de désactivation"
**faire**
    quitter le groupe;
**fin faire**

---

Algorithme 8.3 : Désactivation d'un service d'objet

Il reste cependant à faire un traitement annexe au niveau du processus gestionnaire d'objets (PGO). Un PGO gère plusieurs duplicas sur un site donc il appartient à autant de groupes qu'il gère de duplicas. Lorsqu'un PGO ne gère plus aucun duplica (i.e., il n'appartient plus à aucun groupe), il doit se terminer afin de libérer les ressources qu'il utilise.

## 3.5 Service d'activation

Le service d'activation est un service dupliqué, afin de tolérer les pannes. Il a trois rôles : (1) la création d'un service d'objet, (2) l'activation d'un service d'objet, et (3) la régulation d'un service d'objet.

Pour ce faire, ce service est responsable de la maintenance de la liste $L_g$ des $p$ sites mis à disposition par la communauté, pour le noyau. Cette liste est dynamique, puisque à tout moment de la coopération un site peut être ajouté. Ce service maintient également, pour chaque objet $X$, deux listes $L_t$ et $L_r$. La liste $L_t$ contient les $n$ sites[4] "titulaires" de la liste

[4] Où $n$ est le taux de duplication de l'objet.

$L_g$ qui abritent un duplica de l'objet $X$ dans sa version active ou passive (i.e., données sur disque). La liste $L_r$ contient les $n - p$ sites "remplaçants" susceptibles d'abriter un duplica dans le cas d'une défaillance d'un des $n$ titulaires.

### Création d'un service d'objet

Le premier rôle du service d'activation, la création d'un service d'objet, permet d'ajouter dynamiquement des objets dans le noyau. Lors de la création d'un nouvel objet (décomposition du document), le service d'activation est invoqué pour qu'il crée un nouveau service d'objet. La création se fait par l'envoi de la part d'un utilisateur d'une requête "activation de X" au service d'activation.

```
var
      L_g : liste des p sites constituant le noyau.
      L_t : liste des n sites titulaires pour l'objet X.                    - (initialement vide)
      L_r : liste des p − n sites remplaçants.
      s_u : site de l'utilisateur demandant la création
–
à la réception de "requête création de X"
faire
      initialiser L_r à L_g;          - (initialiser la liste des sites remplaçants pour l'objet X)
      choisir s de L_r tel que s soit le plus proche de s_u;
      retirer s de L_r;
      mettre s dans L_t;              - (mettre le site favorisant le créateur comme titulaire)
      pour i = 1 to n − 1
            retirer un site s de L_r;
            mettre s dans L_t;
      fin pour      pour tout site s ∈ L_t
      faire                                          - (compléter la liste des sites titulaires)
            si il n'y a pas de PGO sur s
            alors lancer un PGO sur s;
            fin si
            envoyer "requête de jonction du groupe X" au PGO du site s;
      fin faire
fin faire
```

Algorithme 8.4 : Création d'un service d'objet

Le service d'activation choisit $n$ sites opérationnels parmi les $p$ disponibles dans la liste $L_g$, en privilégiant un site proche de l'utilisateur créant le nouvel objet. Pour chacun de ces sites, il va falloir créer un duplica. Si le processus gestionnaire d'objets est déjà présent sur le site choisi (un autre objet actif est déjà géré sur ce site), alors il suffit de demander au processus de joindre le groupe de duplicas. Sinon, il faut lancer un processus gestionnaire d'objets puis lui demander de joindre le groupe de l'objet $X$. Ce dernier après avoir joint le groupe, initialise le jeu de données correspondant à l'objet $X$. Lorsque le groupe existe, les duplicas ayant rejoint le groupe déterminent la liste des duplicas absents. Ils seront considérés comme étant en panne et une requête de régulation sera envoyée au service d'activation qui la

traitera comme décrit dans la suite de cette section (Cf. Alg. 8.6). Cependant, il est à noter que dans un système asynchrone, il n'est pas possible de différencier un processus lent d'un processus en panne [Fisher 85]. Il est donc tout à fait possible (mais rare) que le processus gestionnaire d'objet d'un des sites de $L_t$ déclaré en panne rejoigne finalement le groupe. Il sera alors informé par les autres duplicas du groupe qu'il doit quitter le groupe.

Le placement initial a de grandes chances de ne pas être optimal vu le placement aléatoire, mais il est à noter (1) que le créateur de l'objet aura de bonnes performances, et (2) que les duplicas migreront en fonction de l'utilisation grâce au service de régulation (Alg. 8.6).

### Activation d'un service d'objet

Le second rôle du service d'activation, est l'activation proprement dite d'un service d'objet. Elle consiste à remettre en activité un service d'objet passivé. En effet, un service d'objet passivé ne peut être directement accédé par un utilisateur, puisqu'il n'existe plus aucune entité active associée. L'activation se fait par l'envoi de la part d'un utilisateur d'une requête "activation de X" au service d'activation.

```
var
      L_t : liste des n sites titulaires pour l'objet X
      L_r : la liste des p − n sites remplaçants
–
à la réception de "requête d'activation de X"
faire
      pour tout s ∈ L_t
      faire
            si il n'y a pas de PGO sur s
            alors
                  lancer un PGO sur s ;
            fin si
            envoyer "requête de jonction du groupe X" au PGO du site s;
      fin faire
fin faire
```

Algorithme 8.5 : Activation d'un service d'objet

Le service d'activation procède de manière similaire à la création. La différence est que la liste $L_t$ des sites titulaires existe déjà. Le processus gestionnaire d'objet de chacun des sites de $L_t$ joint le groupe formant ainsi le service d'objet correspondant à $X$.

Il est à noter que dans la pratique, les sites du noyau abritent souvent un processus gestionnaire d'objets car des duplicas d'autres objets sont actifs. Dans ce cas de figure, l'activation d'un service d'objet est extrêmement rapide puisqu'elle consiste simplement à rejoindre un groupe. La motivation, pour ne pas laisser un processus gestionnaire d'objets en permanence sur chacun des sites du noyau, est que l'activité liée à un document peut cesser pendant des périodes de temps importantes.

Grâce à ce mécanisme, un service d'objet passivé est activé avec l'état le plus récent de l'objet, pour peu qu'un seul des sites de la liste $L_t$ soit opérationnel. Dans le cas contraire, il faudra attendre qu'au moins un de ces sites devienne accessible.

### Régulation du service d'objet

Le troisième rôle du service d'activation est la régulation du service d'objet. La régulation consiste à utiliser les statistiques fournies par le groupe de duplicas pour choisir parmi les $p - n$ sites de remplacement, ceux qui conviendraient le mieux pour remplacer les sites défaillants.

Le protocole pour la régulation du service d'objet correspondant à l'objet $X$ est le suivant :

---

**var**
  $L_t$ : liste des $n$ sites titulaires pour l'objet $X$.
  $L_r$ : liste des $p - n$ sites remplaçants.
–
  $L_a$ : liste des sites actifs – (envoyés avec la requête de régulation)
  *statistiques* : statistiques sur l'utilisation du service d'objet par les utilisateurs
  – (envoyées avec la requête de régulation)
–
**à la réception de** "requête de régulation", $L_a$, *statistiques*
**faire**
  **pour tout** site $s_f$ en panne – ($s_f \in L_t$ mais $s_f \notin L_a$)
  **faire**
    choisir $s_r \in L_r$ tel que $s_r$ non en panne et $s_r$ correspond le mieux aux *statistiques*;
    **si** il n'y a pas de PGO sur $s_r$
    **alors**
      lancer un PGO sur $s_r$;
    **fin si**
    **envoyer** "requête de jonction du groupe $X$" au PGO de $s_r$;
    enlever $s_f$ de $L_t$; – (car il n'est plus titulaire)
    mettre $s_f$ dans $L_r$; – (car il pourra être remplaçant après la panne)
    enlever $s_r$ de $L_r$; – (car il est maintenant titulaire)
    mettre $s_r$ dans $L_t$ et dans $L_a$; – (car il est maintenant titulaire)
  **fin faire**
**fin faire**

---

Algorithme 8.6 : Régulation par le service d'activation

En rejoignant le groupe, un processus gestionnaire d'objets correspondant à un duplica de remplacement va récupérer l'état du groupe de duplicas et devenir un duplica de $X$ opérationnel. En fin de traitement, les sites opérationnels seront dans la liste $L_t$.

## 3.6 Service de noms

Le service de noms a été spécialement conçu pour résoudre le problème de partage d'information pour des applications coopératives. Sans rentrer dans les détails, soulignons que ce

genre d'applications nécessite de partager des fichiers, des services, ou tout autre entité que la communauté a décidé de mettre en commun. Le nombre de ces entités est généralement faible (de l'ordre d'une centaine), mais elles sont distribuées sur un réseau potentiellement important par les distances, et probablement hétérogène. De plus, la nature de ces entités échappe complètement au service de noms. Dans ces conditions, il n'est pas possible d'utiliser des services de noms généraux qui sont soit trop coûteux en terme de performances, soit inadaptés au service nécessaire.

Le service de noms que nous utilisons le *"light weight name server"* a été développé au sein même du laboratoire [Lugeon 93], et permet de résoudre exactement le problème posé par ce type d'application en général, et par celui de DUPLEX en particulier. Le terme de *"light weight"* vient du fait qu'un certain nombre de fonctionnalités qui ne sont pas tout le temps nécessaires, ont été retirées du service de noms de façon à améliorer les performances. Ces fonctionnalités sont, si nécessaire, réalisées au niveau de l'application. Par exemple, il n'y a pas de vérification de conflit de nom lors de l'enregistrement d'un nouveau nom. Cela permet dans le cas de DUPLEX, où il est garanti a priori qu'il n'y aura pas de conflit, d'éviter le coûteux protocole de vérification. Ainsi, l'enregistrement peut être réalisé localement.

# 4 Infrastructures nécessaires aux politiques de contrôle de concurrence

Comme nous l'avons précédemment montré, le contrôle de cohérence de DUPLEX assure que les opérations lecture et écriture sont correctement ordonnées en regard du critère de cohérence. Ceci n'assure toutefois pas l'atomicité des opérations lecture-modification-écriture. Les alternatives suivantes sont proposées dans DUPLEX pour assurer ce type d'atomicité, lorsqu'elle est souhaitée par les utilisateurs :

- le droit d'écrire sur une partie de document est réservé à un seul utilisateur;
- le droit d'écrire sur une partie de document est réservé à un groupe d'utilisateurs, les conflits étant détectés a posteriori;
- le droit d'écrire sur une partie de document est autorisé à l'ensemble des utilisateurs, les conflits étant détectés a posteriori;
- le droit d'écrire sur une partie de document est autorisé à l'ensemble des utilisateurs, aucun contrôle n'est fait, ce mode suppose des règles entre les utilisateurs.

Le passage de la politique la plus optimiste à la politique la plus pessimiste et réciproquement, peut se faire graduellement, et ce pour chacune des parties indépendantes du document.

L'infrastructure nécessaire pour mettre en œuvre ces quatre politiques de contrôle de concurrence est offerte par le noyau. Sont proposés : (1) un mécanisme de détection de conflit permettant de détecter quand deux opérations lecture-modification-écriture entrent en conflit et, (2) un mécanisme de capacité, qui permet d'attribuer des droits d'accès à son détenteur. Ces deux mécanismes sont décrits en détail ci-dessous.

## 4.1 Capacités

Une capacité[Lampson 69, Lampson 71] est associée à un objet et détermine la liste d'opérations qu'un utilisateur la possédant peut exécuter sur l'objet. Cette liste d'opérations n'est pas statique, mais évolue en fonction des capacités que les autres utilisateurs possèdent sur le même objet.

**Opérations autorisées avec X quand il existe Y dans le système**

| X \ Y | Défaut | Écriture partagée | Écriture exclusive | Accès exclusif |
|---|---|---|---|---|
| Défaut | Lecture<br>Écriture<br>Acquisition capacité supérieure | Lecture<br>Écriture<br>Acquisition capacité supérieure | Lecture<br>~~Écriture~~<br>~~Acquisition capacité supérieure~~ | ~~Lecture~~<br>~~Écriture~~<br>~~Acquisition capacité supérieure~~ |
| Écriture partagée | Lecture<br>Écriture<br>Acquisition capacité supérieure | Lecture<br>Écriture<br>Acquisition capacité supérieure | Lecture<br>~~Écriture~~<br>~~Acquisition capacité supérieure~~ | ~~Lecture~~<br>~~Écriture~~<br>~~Acquisition capacité supérieure~~ |
| Écriture exclusive | Lecture<br>Écriture<br>Acquisition capacité supérieure | Lecture<br>Écriture<br>Acquisition capacité supérieure | IMPOSSIBLE NON PARTAGEABLE | IMPOSSIBLE |
| Accès exclusif | Lecture<br>Écriture | Lecture<br>Écriture | IMPOSSIBLE | IMPOSSIBLE NON PARTAGEABLE |

Figure 8.2 : Tableau donnant les opérations autorisées par les capacités

Par défaut, tous les utilisateurs possèdent une capacité Défaut sur tous les objets qui leur permet d'exécuter les opérations de lecture, écriture, et d'acquisition d'une capacité supérieure. Les opérations lecture, écriture sont celles qui ont été présentées précédemment, et permettent de réaliser également les opérations sur la décomposition du document (Cf. Chapitre 7). L'acquisition d'une capacité supérieure est propre à la gestion des capacités. Finalement, la restitution d'une capacité n'est pas soumise à la détention d'une capacité particulière, elle est toujours réalisable.

La liste d'opérations autorisées avec une capacité dépend des capacités que possèdent les autres utilisateurs sur le même objet. Pour illustrer notre propos, considérons les quatre capacités suivantes : Défaut, Écriture partagée, Écriture exclusive et Accès exclusif. La figure 8.2 donne les opérations qu'un utilisateur peut réaliser avec une capacité $X$ lorsqu'un autre utilisateur possède une capacité $Y$ pour le même objet. Lors d'une acquisition, la capacité obtenue remplace l'ancienne et lors d'une restitution, la capacité est restituée et remplacée par la capacité Défaut. Ce qui fait qu'un utilisateur ne possède qu'une seule capacité sur un objet à un instant donné.

Les capacités Défaut et Écriture partagée sont partageables ce qui signifie que plusieurs utilisateurs peuvent les posséder en même temps. En revanche, les capacités Écriture exclusive et Accès exclusif ne le sont pas, d'où l'impossibilité que deux exemplaires coexistent. Finalement, lorsqu'une capacité Écriture exclusive existe, c'est la seule qui permette d'acquérir la capacité Accès exclusif, d'où l'impossibilité pour les capacités Écriture exclusive et Accès exclusif de coexister. La capacité Accès exclusif n'est en principe pas destinée aux utilisateurs, mais elle est utilisée pour le mécanisme de regroupement (Cf. Chapitre 7).

Les capacités sont gérées par les services d'objet. Chaque utilisateur associe à sa requête la capacité qu'il détient sur l'objet destinataire de la requête. Si la requête est autorisée, alors elle sera traitée. Dans le cas contraire l'émetteur de la requête sera informé.

Pour réaliser ce traitement, un service d'objets doit maintenir les informations suivantes :

- Un tableau similaire à celui de la figure 8.2
- Le nombre de capacités de chaque type possédées par les utilisateurs
- La liste des utilisateurs qui possèdent chacune des capacités

Les deux premières informations servent à accepter ou non les requêtes adressées par les utilisateurs. La liste des utilisateurs est utile pour contacter les utilisateurs possédant une capacité particulière, afin de leur demander soit de restituer la capacité, soit d'accepter de la partager. Les différentes possibilités sont présentées dans le chapitre 9 décrivant l'interface et les fonctionnalités. L'intérêt du mécanisme de capacité est qu'il est possible d'ajouter de nouveaux types de capacité au niveau des services d'objets sans avoir à modifier le code.

Une utilisation des capacités qui sort du cadre du contrôle de concurrence, est la protection des données. Des capacités particulières pourraient être utilisées pour rendre certaines parties du document confidentielles. Par exemple, dans un cadre moins ouvert que le milieu académique, une capacité de lecture partagée pourrait permettre de donner une confidentialité à certaines données qui ne seraient visibles que pour le groupe d'utilisateurs la possédant.

## 4.2 Détection de conflit pour les opérations lecture-modification-écriture

Comme nous l'avons présenté dans le chapitre 2, le gestionnaire de cohérence de DUPLEX n'assure l'atomicité que des opérations de lecture et écriture. L'atomicité des opérations lecture-modification-écriture n'est pas assurée par défaut. Ainsi il est possible que deux opérations de type lecture-modification-écriture soient concurrentes et entrelacées. Il convient dans ce cas de détecter le conflit, et d'avertir les utilisateurs concernés. Nous distinguons deux types de conflit.

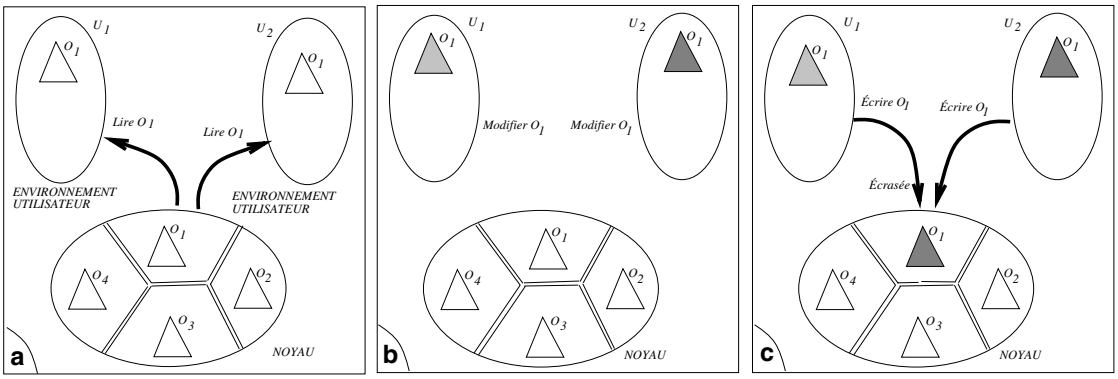


Figure 8.3 : Scénario entraînant une perte d'information

Le premier type de conflit est illustré par la figure 8.3. Supposons que les deux utilisateurs lisent la même version de $O_1$ depuis le noyau puis la modifient localement, et finalement décident tous deux de mettre à jour le noyau; les deux écritures sont concurrentes en regard du contrôle de cohérence et seront donc arbitrairement ordonnées et appliquées à l'objet $O_1$. La dernière écriture écrasera toute trace des précédentes et ainsi, dans l'exemple de la figure 8.3, les modifications de l'utilisateur $U_1$ seront perdues. Bien que ce comportement soit valide du point de vue du contrôle de cohérence au niveau de l'objet $O_1$, l'atomicité des opérations lecture-modification-écriture des utilisateurs $U_1$ et $U_2$ n'a pas été assurée. De telles opérations seront appelées *conflictuelles* avec la définition suivante :

> Définition 8.4 : *Des opérations $Op_1$ et $Op_2$ de type* lecture-modification-écriture *sur un objet $O$ sont conflictuelles, si la* lecture *de $Op_1$ et celle de $Op_2$ ont lu la même version de $O$.*

Le deuxième type de conflit est illustré dans la figure 8.4, où un utilisateur possède localement une vieille version d'un objet $O_1$. Par exemple, il peut avoir lu une version de l'objet $O_1$ pour l'imprimer. Cet objet $O_1$ évolue au niveau du noyau suite aux opérations d'autres utilisateurs, rendant la version locale conservée par $U_1$, obsolète à son insu. Si $U_1$ décide d'effectuer des modifications sur sa version locale de $O_1$, et de soumettre cette version au noyau, il y aura conflit.

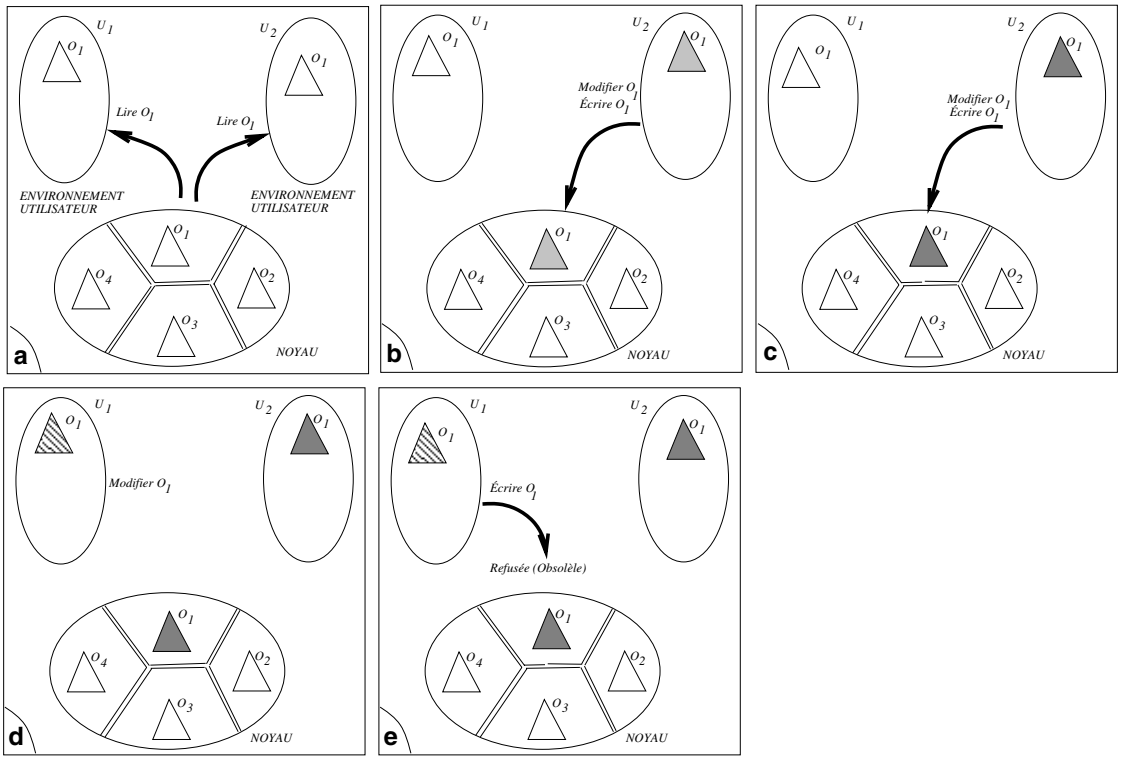


Figure 8.4 : Scénario entraînant une impossibilité de valider ses modifications

Dans ce cas de figure, l'atomicité de l'opération lecture-modification-écriture de $U_1$ n'est pas assurée non plus, mais en outre, l'écriture de $U_1$ ne peut plus être ordonnée que dans le passé (Cf. protocole Chapitre 5), sinon il y aurait violation du critère de cohérence (linéarisabilité ou cohérence séquentielle) sur l'objet $O_1$. L'opération de l'utilisateur $U_1$ est dite *obsolète* avec la définition suivante.

> Définition 8.5 : *Une opération $Op_1$ de type* lecture-modification-écriture *sur un objet $O$ est dite obsolète, si elle est conflictuelle avec une autre opération $Op_2$ de type* lecture-modification-écriture*, et que l'*écriture *de $Op_1$ doit être ordonnée avant l'*écriture *de $Op_2$ pour respecter le critère de cohérence.*

La distinction majeure pour l'utilisateur entre une opération conflictuelle et une opération obsolète est que la deuxième sera obligatoirement écartée par le noyau, alors que la première pourrait être acceptée.

## 4.3 Contrôle de concurrence pour les opérations lecture-modification-écriture

Nous reprenons ici les quatre politiques de contrôle de concurrence que nous avons introduites au début de cette section.

### Politique basée sur un droit d'écriture exclusif

C'est une politique basée sur la capacité appelée exclusive. Cette capacité ne peut être possédée que par un seul utilisateur, elle garantit donc que l'objet ne pourra être modifié que par le possesseur. Cette capacité peut être transmise à un autre utilisateur ou restituée après utilisation. Dans le cas d'une transmission, elle peut être considérée comme un jeton, en cas d'acquisition/restitution, elle peut être considérée comme un verrou.

### Politique basée sur un droit d'écriture réservé à un groupe d'utilisateurs

Cette politique utilise des capacités de type écriture, qui sont partageables. Plusieurs utilisateurs peuvent ainsi posséder une telle capacité en même temps, et par conséquent réaliser des écritures concurrentes entraînant des opérations lecture-modification-écriture conflictuelles et obsolètes. Le mécanisme de détection de conflit est donc utilisé. Suivant les cas, l'utilisateur peut résoudre le problème seul ou bien entrer dans une phase de conciliation dans les cas plus délicats (par exemple, s'il y a conflit d'intérêt entre les utilisateurs ou bien si leur nombre est trop important). Il est ainsi possible après négociation entre les différents auteurs impliqués, de dégager une version consensuelle qui sera soumise au noyau. Il est clair que ce type de fonctionnement n'est réellement exploitable que si les conflits sont rares (Cf. Chapitre 2, remarque 9), hypothèse qui est toutefois réaliste dans l'environnement DUPLEX grâce à la décomposition du document.

Il est par ailleurs garanti que les utilisateurs ne possédant pas de capacité d'écriture ne pourront pas modifier l'objet. Ainsi la possession de la capacité écriture peut être associée au rôle de rédacteur que l'on trouve dans certains environnements (e.g., Quilt [Fish 88, Leland 88], SharedBook [Lewis 88] et Griffon [Decouchant 93]), à la différence près que dans le cas de DUPLEX plusieurs rédacteurs sont autorisés en même temps.

### Politique basée sur une détection des conflits

Cette politique n'utilise pas de capacité particulière, et permet à n'importe quel utilisateur d'écrire. Seul le mécanisme de détection de conflit est utilisé. Les modes de conciliation sont les mêmes que précédemment.

### Politique basée sur l'existence de règles externes

Cette politique se base sur des règles que les utilisateurs établissent afin d'éviter les conflits a priori. Le découpage en tranches horaires peut être utilisé afin de permettre à chacun de travailler sans être perturbé. Cette approche est particulièrement intéressante, puisque

lorsque les utilisateurs travaillent dans des fuseaux horaires différents, ce découpage est quasiment implicite.

# 5 Discussion

Ce chapitre a présenté les différents éléments permettant de partager un document entre une communauté d'utilisateurs. Un service efficace de régulation de la duplication permet de garantir une bonne tolérance aux pannes, ce qui est important dans le système distribué que nous considérons. Les services d'objet se désactivent automatiquement dès qu'ils ne sont plus utilisés afin de limiter les ressources utilisées, et de dispenser la communauté impliquée dans la collaboration de tâches d'administration. Le noyau est autonome, et contrôle lui-même les ressources qu'il utilise.

Le deuxième point développé dans ce chapitre concerne l'infrastructure nécessaire aux politiques de contrôle de concurrence explicite au niveau des opérations lecture-modification-écriture qui permet à l'utilisateur de choisir la politique (exclusive, pessimiste, optimiste, aucune, etc ...) dont il a réellement besoin en fonction de (1) la maturité du document, (2) des personnes avec qui il travaille, et (3) de la partie du document concernée. En effet, le contrôle de concurrence est spécifique à chacun des segments indépendants. Nous avons montré que le mécanisme de capacités est extensible, puisqu'il est aisé de rajouter des capacités particulières. Par exemple, il serait possible de définir à des fins de confidentialité, des niveaux de hiérarchie permettant d'utiliser DUPLEX dans des milieux industriels ou militaires moins ouverts que le milieu de la recherche scientifique. En ce qui concerne les différentes politiques que nous avons détaillées dans ce chapitre, elles recouvrent entre autres les mécanismes proposés dans la littérature. Ainsi, l'environnement d'édition coopérative DUPLEX s'adapte aux utilisateurs, et non l'inverse.

# Chapitre 9

# Environnement de l'utilisateur

## 1 Introduction

L'environnement local d'un utilisateur est plus sensible à l'hétérogénéité que le noyau. En effet, s'il est possible de choisir quels sites formeront le noyau, les sites utilisateurs seront pour leur part imposés. Pour prendre en compte ces considérations et éviter de réaliser des interfaces spécifiques pour chaque architecture, l'environnement local d'un utilisateur de DUPLEX utilise en grande partie des outils standards, puisqu'il se base sur un environnement d'édition autonome sur lequel viennent se greffer toutes les fonctionnalités nécessaires à la coopération.

Le but de DUPLEX est de permettre à l'utilisateur de conserver les outils avec lesquels il a l'habitude de travailler. La seule entorse, dans le prototype actuel, consiste à imposer LaTeX[Lamport 86, Knuth 84] pour l'environnement d'édition. Cette limitation peut être levée, et des propositions en ce sens sont faites dans la section consacrée à l'environnement d'édition autonome.

Ainsi, l'environnement local de DUPLEX intègre :

(1) un environnement d'édition autonome (LaTeX à l'heure actuelle);

(2) des fonctionnalités permettant une fragmentation du document en segments indépendants;

(3) des fonctionnalités pour l'archivage de ces segments sur le site des utilisateurs;

(4) des fonctionnalités pour les communications;

(5) des fonctionnalités permettant d'interagir avec le noyau.

Les points 2 et 4 ont déjà été abordés, respectivement dans les chapitres 1 et 2 de cette même partie. Nous ne développerons ici que les interfaces graphiques correspondantes. En revanche, les autres points seront plus particulièrement décrits dans la suite de ce chapitre.

La section 2 présente l'interface utilisateur intégrant toutes les fonctionnalités de DUPLEX. Cette interface permet de visualiser le document et d'exécuter un certain nombre d'opérations sur les segments indépendants qui le composent. L'interface permet également d'appeler des outils externes offrant une partie des fonctionnalités de l'environnement. La section 3 présente l'environnement d'édition autonome et la façon dont il est possible de le paramétrer selon le besoin de l'utilisateur. La section 4 décrit le module d'archivage et les motivations pour de telles fonctionnalités, ainsi que les aspects d'interrogation du journal associé à chaque partie du document. La section 5 conclut ce chapitre.

## 2 Interface Utilisateur

Le document que nous utilisons pour illustrer notre propos est le même que celui utilisé dans le chapitre 7. Nous en rappelons ici la table des matières.



Figure 9.1 : Table des matières de document servant d'exemple

Pour reprendre le vocabulaire introduit dans le chapitre sur la décomposition du document, nous utiliserons les termes de segments dépendants et indépendants. L'environnement de l'utilisateur contient un certain nombre d'objets qui constituent sa vue locale.

Ces objets représentent les segments du document avec lesquels l'utilisateur travaille, soit activement (modification), soit passivement (consultation). Pour les premiers, il les modifie grâce à des outils d'édition traditionnels avant de les transmettre au noyau. Pour les seconds, ils ne sont utilisés que pour de la visualisation (impression ou affichage à l'écran).

Le nombre de segments indépendants peut être important, ce qui nécessite une interface

graphique permettant à l'utilisateur non seulement d'appréhender le document dans sa globalité, mais également de pouvoir se focaliser plus particulièrement sur la ou les parties qui l'intéressent. De plus, cette interface graphique doit permettre de désigner facilement les parties du document avec lesquelles un utilisateur veut travailler et les actions qu'il veut effectuer. Finalement, en plus de visualiser la vue locale d'un utilisateur, des informations concernant les aspects de coopération doivent également être affichées.

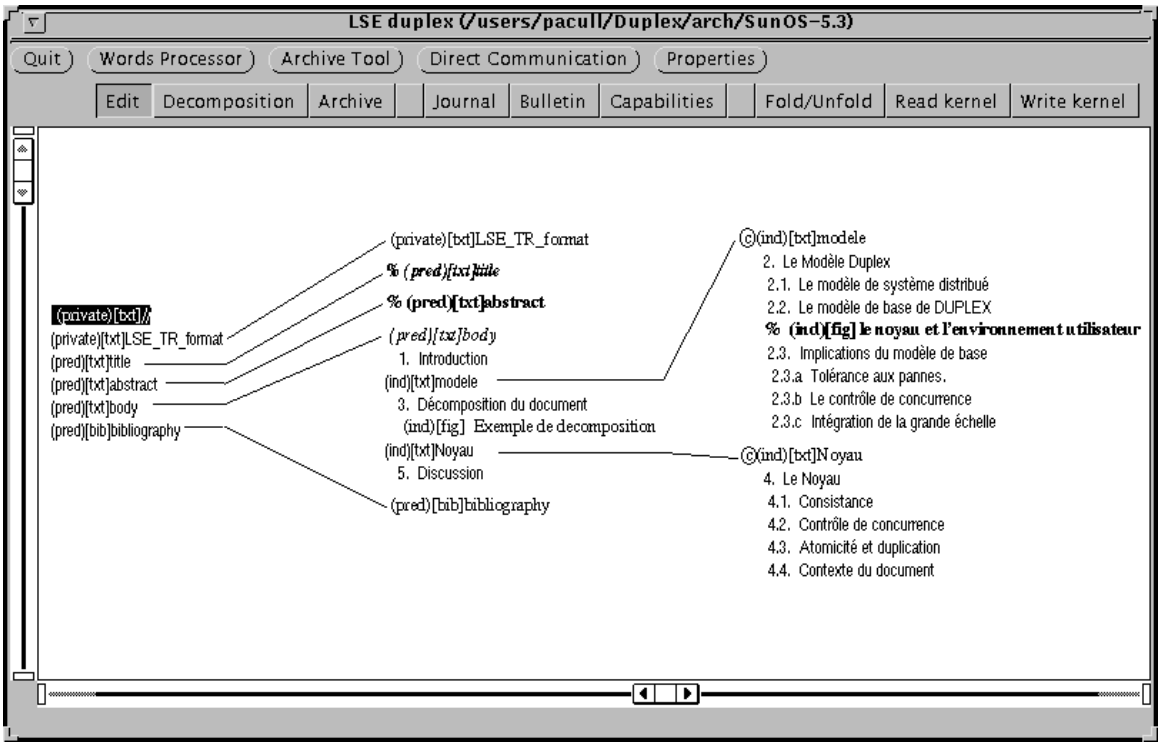


Figure 9.2 : Fenêtre principale de l'interface utilisateur

Le composant principal de l'interface utilisateur est la fenêtre de visualisation représentée dans la figure 9.2. Elle a trois fonctions principales :

**Affichage :** Les segments de document que l'utilisateur possède dans sa vue, ainsi que le placement de ces derniers dans la structure du document, sont affichés sous forme d'arbre, où les noeuds sont les segments indépendants. Un certain nombre d'informations associées à ces segments décorent cet arbre. Par exemple, les préfixes (pred) ou encore [txt] qui donnent des informations sur la nature du segment (Cf. Section 2.1).

**Notification :** Un certain nombre d'informations sont disponibles au sujet des divergences entre les objets du noyau et les objets locaux.

**Tableau de bord :** Tous les outils (communication, archivage, etc ), ainsi que les modes opératoires, sont sélectionnables à partir de cette fenêtre. Les outils sont destinés à réaliser des opérations concernant le document dans son ensemble, alors que les modes opératoires permettent d'agir individuellement sur chacun des segments. Lorsqu'un mode opératoire est choisi, il suffit de sélectionner par un double clic un noeud de l'arbre pour effectuer une action sur le segment correspondant. Par exemple, si le mode opératoire Edit qui définit le mode édition est choisi, l'action consistera à lancer un éditeur permettant de modifier le segment.

Ces fonctionnalités sont décrites plus en détail ci-dessous.

## 2.1 Affichage

Le document est représenté sous forme d'un arbre dont un noeud correspond à un segment indépendant, et où un arc $S_i - S_j$ lie le segment $S_i$ contenant la référence à un segment indépendant $S_j$ au segment indépendant $S_j$ lui-même. Chaque noeud est étiqueté par le nom logique du segment indépendant qu'il représente, ce qui combiné à sa place dans l'arbre, permet de le définir sans ambiguïté.

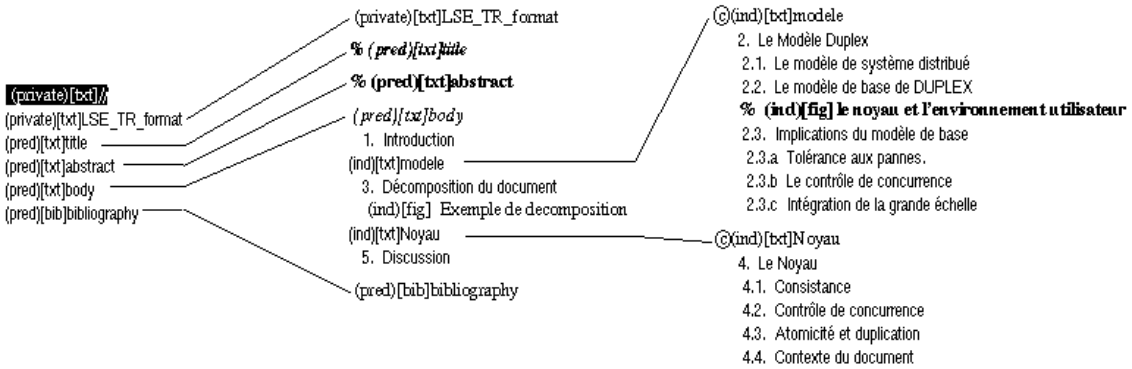


Figure 9.3 : Représentation initiale du document

Pour chaque segment indépendant, les segments dépendants et les références aux segments indépendants sont listés dans l'ordre d'apparition dans la séquence des segments du document. Par exemple dans la figure 9.2, le segment indépendant body contient trois segments dépendants, 1. Introduction, 3. Décomposition du document et 5. Discussion, et les références aux segments indépendants modele et Noyau. Les informations structurales des segments indépendants sont indentées et numérotées comme dans une table des matières.

Les segments indépendants correspondant à des figures sont traités de manière légèrement différente. En effet, une figure étant un segment élémentaire (qui ne peut être décomposé),

elle ne contient pas d'information structurale. Dans ce cas, le noeud indépendant qui correspond est réduit à sa seule étiquette. La référence à un segment indépendant contenant une figure est remplacée par l'étiquette de l'objet indépendant (e.g., [FIG] LE NOYAU ET L'ENVIRONNEMENT UTILISATEUR dans le segment indépendant modele de la figure 9.2). Cela permet de compacter la représentation sous forme d'arbre.

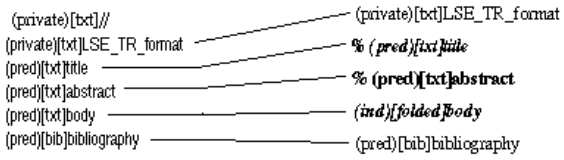


Figure 9.4 : Segment body replié

Ce compactage n'est cependant pas suffisant, car cette représentation arborescente du document peut être rapidement très volumineuse. Pour pallier à ce problème, il est possible de réduire les informations affichées de deux manières orthogonales. La figure 9.3 est la représentation initiale du document.

La première manière consiste à replier un segment indépendant, c'est à dire à ne pas afficher les informations qu'il contient. Par exemple, si le segment body est replié, les segments dépendants, ainsi que les références aux segment indépendants ne sont pas affichés, et par conséquent les segments indépendants eux-mêmes ne sont pas affichés. Cela est équivalent à faire disparaître le segment body de l'espace utilisateur. La figure 9.4 représente le même document que la figure 9.3, mais avec le segment body replié.

La deuxième manière consiste à ne pas afficher les segments correspondant à un certain niveau. Par exemple, on peut choisir de ne pas afficher les paragraphes et sous-paragraphes. Cette option, contrairement à la précédente, concerne toutes les parties indépendantes. La figure 9.5 correspond au même document que la figure 9.3 mais avec uniquement une partie des informations structurales affichées : celles qui sont cochées dans le menu de filtrage. Les sous-sections 2.3.a TOLÉRANCE AUX PANNES , 2.3.b LE CONTRÔLE DE CONCURRENCE, et 2.3.c INTÉGRATION DE LA GRANDE ÉCHELLE ne sont plus affichées.

L'affichage permet également, au niveau de chaque segment indépendant, de visualiser des informations pertinentes sur (1) la nature, (2) le type du contenu et (3) l'état par rapport à la version du noyau correspondante (Cf. Section 2.2 sur les notifications).

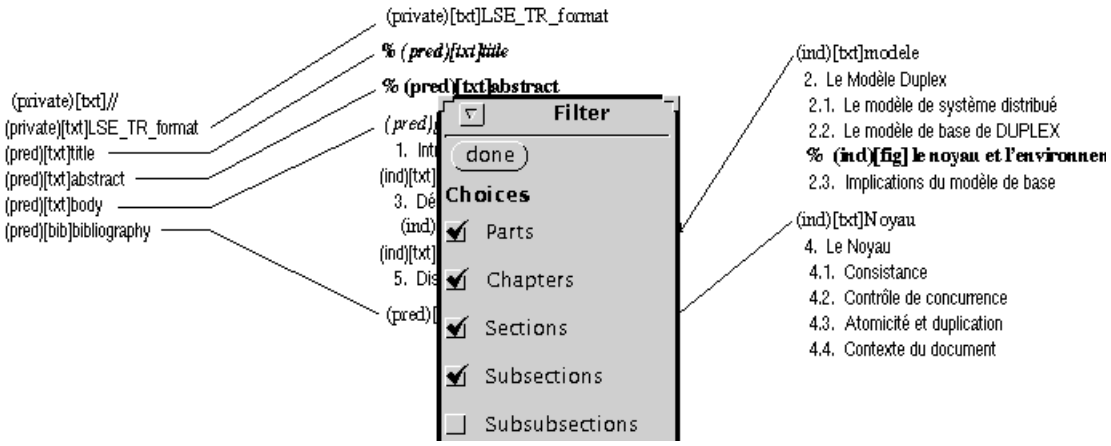


Figure 9.5 : Filtrage des informations structurales

### Nature du segment

Nous distinguons trois catégories de segments indépendants : privé, partagé prédéfini et partagé indépendant, dont l'étiquette est respectivement préfixée par (private), (pred) (ind)

La catégorie "privé" signifie que le segment n'est pas partagé, mais est propre à l'utilisateur. C'est par exemple, la racine du document qui est privée, afin que les utilisateurs puissent paramétrer le document en fonction de leurs préférences. Par exemple, intervenir sur le style d'impression (double ou simple interligne, double ou simple colonne, format A4 ou *"letter"*, recto-verso), y ajouter un logo, une date, ou encore un numéro de version. De la même manière, le style pour les références bibliographiques peut être choisi. En fait, tout ce qui fait la spécificité du format d'impression d'un document est privé à l'utilisateur. En effet partager ce genre de paramètres serait excessivement pénible et responsable de conflits et de perte de temps.

La seconde catégorie concerne les segments partagés prédéfinis qui sont au nombre de quatre : (title, abstract, body et bibliography). Ces segments prédéfinis sont générés lors de la création du document, et sont partagés par la communauté. Le segment body correspond au corps du document : il est le point de départ de la décomposition en segments indépendants du document partagé.

Dans le cas de bibliography, il est préférable de parler d'objet bibliography plutôt que de segment car les informations qu'il contient ne font pas partie intégrante du document, mais servent à générer la section *"références"*. C'est un objet dans lequel toutes les personnes impliquées dans la collaboration peuvent ajouter de nouvelles références bibliographiques, au fur et à mesure des besoins. Cet objet est en quelque sorte "attaché" au document.

La troisième catégorie concerne les segments indépendants qui sont créés par décomposition du segment body. La différence entre un segment (pred) et (ind) est que sur le premier, il n'est pas possible d'effectuer une opération de regroupement avec son objet père. En effet, l'objet père étant la racine, un regroupement aurait pour effet de faire disparaître de la coopération un objet initialement partagé.

### Type du contenu

En termes de contenu, il est préférable de parler d'objets plutôt que de segments, car l'objet mettant en œuvre un segment est plus riche en sémantique.

Nous considérons des objets de cinq types : texte, figure, bibliographie, replié, et manquant, respectivement préfixés par [txt], [fig], [bib], [folded] et [missing].

Le type [txt] est du texte dans un format compréhensible par le traitement de texte utilisé.

Le type [fig] dénote les objets représentant des figures. Ce type d'objet est intéressant, puisque généralement à une figure correspondent deux jeux de données. En effet, un éditeur de figures stocke la figure dans un certain format "source" (e.g., gif, tiff, etc. ) qui n'est pas reconnu par tous les traitements de texte. En revanche le format PostScript est un format reconnu par tous les traitements de texte, mais qui ne permet (en général) pas d'être édité. La solution consiste donc au niveau de l'objet de type [fig] à stocker un jeu de données pour le "source" de la figure, et un pour le traitement de texte. Ici également, la dénomination d'objet est plus juste que segment. Il est à noter que l'éditeur de figures Idraw utilise un format PostScript restreint pour son format de stockage, ce qui permet de n'avoir qu'un seul jeu de données. Cet éditeur est malheureusement incapable de comprendre le PostScript généré par d'autres éditeurs de figures. L'hypothèse que tous les utilisateurs utilisent Idraw ne peut être faite dans le cadre d'un outil général de coopération.

Deux standards pour stocker des informations bibliographiques sont le plus couramment employés : BibTeX[Lamport 86] et refer. L'usage dictant le choix, le format bibtex a été choisi pour stocker les informations bibliographiques dans DUPLEX. Ainsi, l'objet prédéfini bibliography est de type [bib] et contient des informations dans le format bibtex.

Les types [folded] et [missing] sont un peu particuliers. Ils désignent respectivement des objets repliés, décrits précédemment, ou non présents dans l'environnement de l'utilisateur. Dans le dernier cas, seule la référence est connue, et permettra de faire le moment venu une lecture du noyau afin d'en transférer une copie dans l'environnement local.

Bien que seuls cinq types d'objets soient considérés à l'heure actuelle, rien ne s'oppose à l'intégration d'autres types. En fait ce typage n'a pour but que de définir un éditeur particulier pour modifier l'objet. Ainsi, on pourrait rajouter les types "feuille de calcul" ou "code barre". Dans le cadre d'article scientifique, les types [txt], [fig] et [bib] sont suffisants.

## 2.2 Notifications

### État par rapport au noyau

Le modèle DUPLEX autorise une divergence entre l'état du document dans le noyau et dans l'espace de travail de l'utilisateur. Il est possible cependant de notifier à l'utilisateur des informations relatives à cette divergence.

Au niveau de la divergence, deux états sont à distinguer : le premier quand l'utilisateur a modifié localement un segment et ne l'a pas encore répercuté au niveau du noyau, et le deuxième quand la version du noyau a été modifiée par un autre utilisateur.

La première divergence ne dépend que de l'utilisateur, donc l'information est toujours disponible localement. En revanche, le deuxième état ne dépend pas de l'utilisateur, mais d'une action extérieure. Compte tenu du modèle d'interaction asynchrone adopté, cette information n'est disponible que par une interaction avec le noyau. La manière dont cela est traité dans DUPLEX est efficace, car basé sur les dépendances causales entre les opérations effectuées sur les objets. En effet, toute communication entre noyau et environnement utilisateur, ou encore entre environnements utilisateurs, transporte le passé causal de chacun des objets du noyau (en d'autres termes, la connaissance des opérations effectuées sur les autres objets du noyau). Il est ainsi possible de savoir qu'un certain nombre d'objets du noyau ont été modifiés.

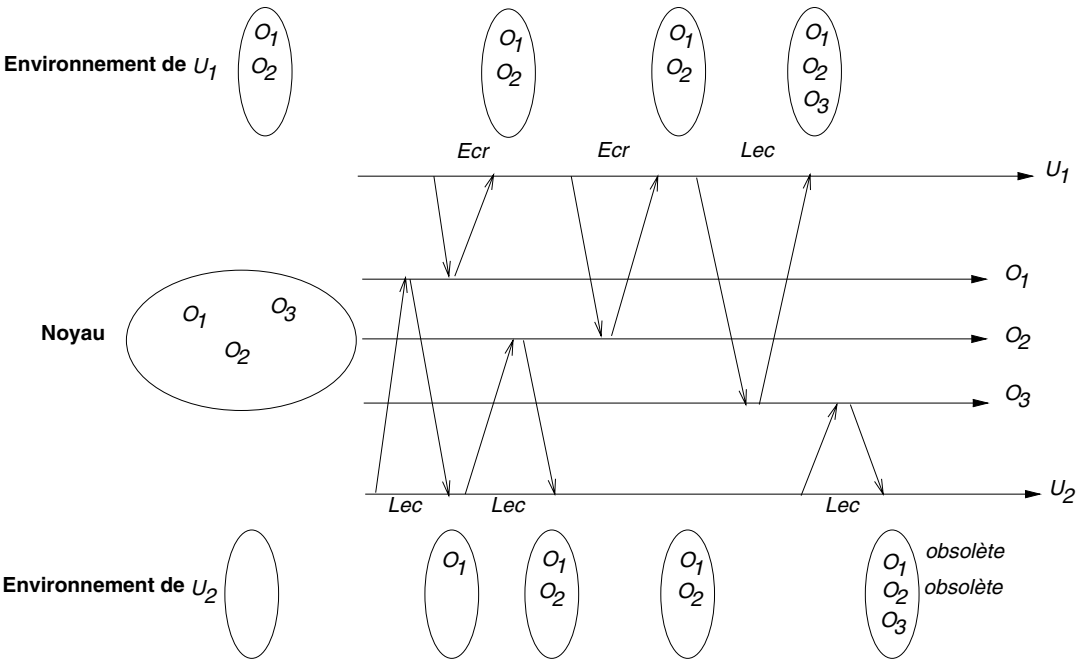


Figure 9.6 : Propagation des informations causales

Ceci est illustré dans la figure 9.6 considérant deux utilisateurs $U_1$ et $U_2$. La figure représente deux informations, la vue logique du noyau et des environnements de $U_1$ et $U_2$, et le flot d'exécution correspondant aux opérations effectuées par $U_1$ et $U_2$. Les opérations exécutées par les utilisateurs sont matérialisées par une invocation à un objet et la réponse de l'objet. $U_1$ réalise la lecture des objets $O_1$ et $O_2$, puis $U_1$ réalise l'écriture de $O_1$ et $O_2$, et finalement la lecture de $O_3$. En faisant une lecture de $O_3$, $U_2$ est informé que les versions locales de $O_1$ et $O_2$ sont obsolètes.

La mise en œuvre d'un tel mécanisme est peu coûteux en terme de structures de contrôle puisqu'un simple vecteur de la taille du nombre d'objets est nécessaire. De plus cette structure de contrôle étant également utilisée pour assurer la cohérence (Cf. Chapitre 5), les notifications sont obtenues gratuitement. Toute requête permet d'obtenir des informations sur le passé causal et au besoin, l'utilisateur peut envoyer une requête n'ayant pour effet que de permettre un rafraîchissement des notifications.

Quatre cas sont donc possibles, ils sont matérialisés au niveau des étiquettes des segments par un codage (Cf. Figure 9.7), les caractères italiques indiquant une version noyau modifiée, le gras une version locale modifiée[1] (Cf. Figure 9.2). Un codage par couleur est possible si l'on dispose d'un écran couleur.

Le cas le plus critique est le cas où à la fois la version locale et la version du noyau ont été modifiées. L'utilisateur devra effectuer une conciliation, soit seul en lisant la version noyau et en y incorporant ses modifications, soit en contactant l'utilisateur responsable de la dernière écriture. Ce contact est réalisé soit en récupérant son identité, stockée au niveau du journal associé à l'objet, soit en réalisant une écriture qui sera forcement conflictuelle ou obsolète et qui retournera l'identité de l'utilisateur.

| | **noyau pas modifié** | **noyau modifié** |
|---|---|---|
| **local pas modifié** | normal | *italique* |
| **local modifié** | **gras** | ***gras italique*** |

Figure 9.7 : Codage des divergences noyau et environnement utilisateur

### Capacités

La capacité du segment indépendant est également affichée. Le symbole "©" est affiché en regard des segments pour lesquels l'utilisateur possède une capacité autre que la capacité **défaut** (e.g., les segments indépendants **modele** et **Noyau** dans la figure 9.2). Les

[1] Le préfixe % est également utilisé avec la même sémantique.

capacités possédées par les autres utilisateurs sont consultables par une requête explicite à l'objet, car le caractère dynamique ne permet pas une notification satisfaisante dans le cadre d'interaction asynchrone.

## 2.3 Tableau de bord

Dans la partie supérieure de la fenêtre principale (Cf. Figure 9.2), on distingue deux rangées de boutons.

La rangée supérieure (boutons arrondis) permet de lancer les outils qui sont intégrés à l'environnement DUPLEX. Parmi ceux-ci nous trouvons (1) l'environnement d'édition autonome, (2) le module d'archivage, et (3) le module de communication directe, synchrone ou asynchrone. Ces trois points font l'objet de la section 3 de ce chapitre.

La rangée inférieure (boutons rectangulaires) permet de sélectionner un mode opératoire. Lorsqu'un segment indépendant est sélectionné (par un double clic), l'opération correspondant à la nature de l'objet et à son type dans ce mode opératoire est déclenchée. Parmi les modes opératoires nous trouvons :

**Edit :** permet de modifier un objet avec l'éditeur ad hoc. L'éditeur est un éditeur standard normalement conçu pour fonctionner avec des fichiers contenus dans le système de fichier local à l'environnement de l'utilisateur. L'éditeur est donc encapsulé dans une boîte de dialogue, afin de permettre des transferts avec le noyau. Ce procédé d'encapsulation est décrit plus en détail dans la suite de ce chapitre.

**Decomposition** : permet d'effectuer les opérations de séparation et regroupement au niveau de la hiérarchie du document. Le type d'opération est déterminé en fonction de la nature de l'objet sélectionné. Si c'est un segment dépendant, l'opération de séparation est effectuée, et donne naissance à un nouveau segment indépendant. Si c'est la référence à un segment indépendant, l'opération de regroupement est effectuée, et le segment devient un segment dépendant du segment père. L'ancien segment indépendant disparaît de l'environnement local de l'utilisateur.

**Archive :** permet d'ouvrir la fenêtre d'archivage pour le segment indépendant sélectionné : e.g. sauvegarder la version courante, ou récupérer une version anciennement sauvegardée. Une description plus détaillée de l'interface est donnée ci-dessous.

**Journal :** permet d'ouvrir la fenêtre liée à la journalisation des opérations sensibles pour le segment indépendant sélectionné. Il est possible de consulter le journal et de savoir l'identité du responsable d'une opération particulière. Par exemple, la destruction à la suite d'un regroupement, ou la modification d'un objet du noyau notifiée à l'utilisateur.

**Bulletin :** permet d'ouvrir la fenêtre d'interface au forum associé à un objet. Ceci concerne les communications indirectes asynchrones en rapport avec un segment indépendant particulier. Les messages sont ordonnés suivant un ordre respectant les dépendances causales entre les messages et les opérations effectuées sur les objets du noyau. Par exemple, si un utilisateur modifie un objet du noyau et poste un message l'annonçant, il n'est pas possible de lire le message et d'obtenir une ancienne version du noyau. Ou encore, il n'est pas possible de lire la réponse à une question posée sans avoir lu la question.

**Capability :** qui permet d'ouvrir la fenêtre d'interface gérant les capacités pour le segment sélectionné. Il est ainsi possible de faire des opérations d'acquisition et de restitution d'une capacité, de consulter les capacités possédées par les autres utilisateurs ou de demander un des traitements spéciaux ci-dessous :

- transmettre une capacité non partageable à un utilisateur particulier, c'est-à-dire la restituer au noyau, mais pour l'usage d'un utilisateur particulier. Une durée pendant laquelle la capacité est "réservée" à cet utilisateur est spécifiable.
- demander à être informé d'une requête d'acquisition d'une capacité de même rang par un autre utilisateur, dans le cas d'une capacité partageable. Deux possibilités : (1) la capacité est attribuée inconditionnellement et le message est uniquement destiné à donner l'information, ou (2) la capacité ne sera attribuée qu'en cas de confirmation.
- demander à être informé en cas d'attribution d'une capacité supérieure par un autre utilisateur.

**Write Kernel :** permet d'écrire l'objet au niveau du noyau.

**Read Kernel :** permet de lire un objet du noyau.

**Fold/Unfold :** permet de replier un objet sur lui même. En d'autres termes, les segments indépendants et les références aux segments indépendants ne sont plus affichés, de même que l'arborescence qui y est associée. Lors de l'impression du document, l'objet ne sera pas non plus affiché. Cela permet de restreindre le document à une partie sur laquelle l'utilisateur travaille.

# 3 Outils annexes

## 3.1 Environnement d'édition autonome

Le moteur dans un environnement d'édition (coopératif ou non) est le traitement de texte utilisé. Le but final de DUPLEX est de permettre l'utilisation de plusieurs traitements de

texte dans les divers environnements locaux. Pour ce faire, la version du noyau devra être dans un format de description de document standardisé tel que SGML [SGML 86] ou ODA [ODA 88]. Les parties indépendantes, formatées dans ce standard, seront traduites dans le format ad hoc avant d'être manipulées par le traitement de texte sur le site utilisateur, et traduites dans le format standard avant d'être envoyées au noyau. Cela laisserait l'utilisateur libre de choisir son traitement de texte préféré (LaTeX[Lamport 86, Knuth 84], Word[Mic 87], Framemaker[Fra 90], Grif [Quint 86], etc.). Cet aspect dépasse toutefois le cadre du premier prototype de DUPLEX, qui se veut une plate-forme de test des concepts proposés dans le contexte de la cohérence et du contrôle de concurrence dans un système à grande échelle. Seul LaTeX est disponible actuellement. Le choix de cet environnement de traitement de texte est lié aux raisons suivantes :

- LaTeX est un format très répandu dans le domaine académique.
- La structure d'un document LaTeX est très proche de SGML. Cela permet dans un premier temps de directement utiliser le format LaTeX au lieu de SGML.
- LaTeX est disponible dans le domaine public et l'environnement, grâce aux outils associés, est très riche.
- Le jeu de macros associé à LaTeX permet de rapidement modifier le comportement du moteur même de LaTeX. Ce qui aurait été difficile à réaliser avec des traitements de texte fermés.

L'environnement d'édition est constitué (1) d'outils classiques, et (2) de fonctionnalités d'interaction avec le noyau. L'utilisation d'outils classiques permet aux utilisateurs de conserver les éditeurs avec lesquels ils ont l'habitude de travailler. Ceci permet une prise en main beaucoup plus rapide de l'environnement DUPLEX. Les paramêtrisations permettent à chacun de modeler l'environnement DUPLEX à ses besoins. De plus DUPLEX n'a besoin que de fonctionnalités et non d'outils pour fonctionner. Par exemple, il a besoin d'une fonction d'édition de texte et non d'un éditeur particulier. Cela permet de considérer Emacs, textedit, voire vi. En fait, l'utilisateur peut conserver ce qu'il utilise déjà dans son environnement de travail LaTeX.

### Intégration de l'environnement LaTeX

L'ensemble des fonctionnalités nécessaires pour LaTeX sont intégrées de façon à ce que l'utilisateur n'ait pas à manipuler de noms de fichiers ou de paramètres. Il suffit de sélectionner une des actions suivantes pour qu'elle soit effectuée directement.

**latex :** permet de générer un fichier au format dvi correspondant aux parties du document présentes dans l'environnement de travail de l'utilisateur.

**bibliography :** permet de générer la section relative aux références bibliographiques du document

**postscript :** permet de générer un fichier PostScript à partir du fichier au format dvi

**postscript preview :** permet de générer un fichier PostScript à partir du fichier au format dvi et de l'afficher à l'écran

**dvi preview :** permet d'afficher à l'écran le document à partir du fichier dvi

**print :** permet d'imprimer le document. Il est à noter que les outils de pré-visualisation PostScript permettent également d'imprimer directement et de façon plus sélective (choix de pages, format du papier, etc.)

### Encapsulation d'outils classiques

L'utilisation d'outils classiques ne permet que de manipuler des fichiers locaux au système de fichiers de l'utilisateur, et non des objets plus complexes comme ceux du noyau. Pour ajouter facilement à n'importe quel outil la possibilité de dialoguer avec le noyau, les outils sont encapsulés dans une boîte de dialogue. Une boîte de dialogue est associée à une paire (outil, objet) et contrôle l'opération réalisée par l'outil sur l'objet. Considérons par exemple que l'outil soit un éditeur de texte. L'encapsulation est réalisée comme nous le montre la figure 9.8. La boîte de dialogue permet de déclencher des transferts entre l'environnement local et le noyau, l'éditeur ne servant qu'à faire les modifications locales.

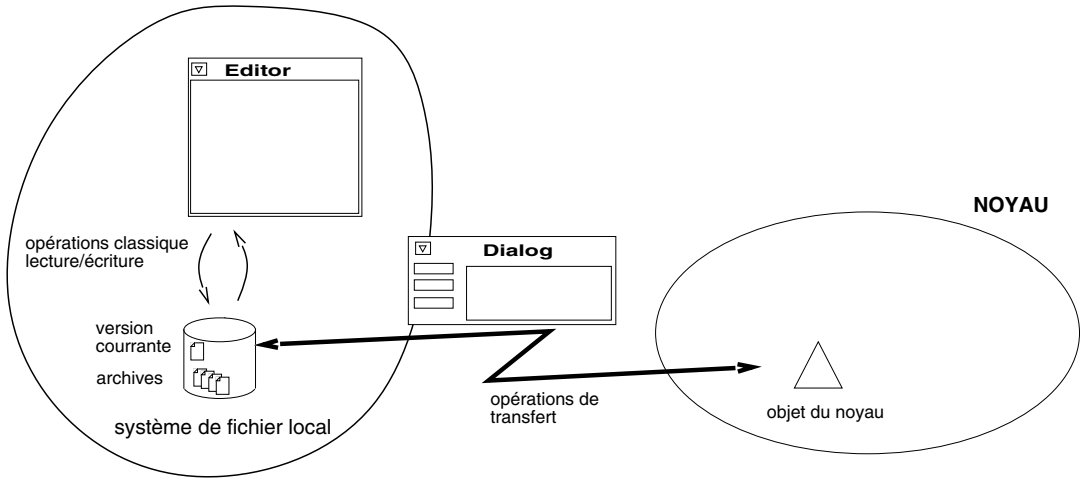


Figure 9.8 : Encapsulation d'un outil classique

La boîte de dialogue permet de fournir un fichier à l'éditeur. Ce fichier peut contenir :

- la version courante de l'objet
- la version courante du noyau (en effectuant une lecture du noyau)
- une version précédemment archivée (Cf. Section 3.2).

L'éditeur permettra alors de modifier le fichier. Les modifications apportées au fichier peuvent être stockées dans :

- la version courante de l'objet (option par défaut)
- la version courante du noyau (en effectuant une écriture du noyau)
- une archive (Cf. Section 3.2).

Ceci peut être réalisé à n'importe quel moment de l'édition et ne nécessite pas de quitter l'éditeur. Ainsi, il est possible de transférer des versions intermédiaires de son travail vers le noyau.

Lors de toutes ces phases, l'utilisateur est averti lorsqu'il y a risque de perte d'information. Par exemple, si la version courante n'a jamais été ni archivée ni envoyée au noyau et qu'une demande de lecture du noyau est effectuée.

## 3.2 Archivage local

L'archivage dans DUPLEX est local, car nous sommes convaincus qu'un utilisateur ne cherche à retrouver dans les archives que des informations qu'il a déjà lues, donc qui ont été à un moment ou un autre présentes dans son environnement de travail.

L'archivage se fait de deux manières :
(1) un archivage de tout le document, où tous les objets actuellement présents dans l'environnement de travail sont archivés, et (2) un archivage ponctuel où seul un objet particulier est archivé. Les deux possibilités sont utiles, car lors d'une recherche d'information archivée, on recherche soit la version du document correspondant à une date précise, soit une version particulière d'un objet.

Quatre fonctionnalités sont donc à considérer :

**L'archivage d'un objet** peut se faire de deux manières différentes : soit grâce à la boîte de dialogue associée à l'outil servant à modifier l'objet, soit à partir de la fenêtre principale, en sélectionnant le mode opératoire "archivage" et en cliquant sur un objet. Un commentaire est demandé à l'utilisateur; il est utilisé pour étiqueter la version de l'objet.

**L'archivage du document complet** peut se faire de trois manières différentes : sur requête de l'utilisateur, sur proposition de DUPLEX lorsque l'utilisateur quitte sa session de travail, ou régulièrement tous les $x$ heures, où $x$ est fourni par l'utilisateur[2]. Dans les deux premiers cas, un commentaire est demandé à l'utilisateur pour étiqueter la version du document.

**La récupération d'une version d'objet** se fait à partir d'une liste de toutes les versions avec le commentaire, la date, l'heure et le type d'archivage (manuel, automatique ou de fin de session). Cette dernière information permet de faire le tri plus facilement.

**La récupération d'une version complète du document** se fait également à partir d'une liste de toutes les versions avec le commentaire, la date, l'heure et le type d'archivage.

# 4 Discussion

L'environnement utilisateur n'est certes pas parfait et manque notamment du "feed back" de la part d'utilisateurs. Il nous satisfait toutefois, puisqu'il est le reflet de notre manière de coopérer.

Insistons sur le fait que l'environnement DUPLEX est facilement paramétrable, puisque l'utilisateur peut conserver les outils qu'il a l'habitude d'utiliser. Il est complet et facilement extensible.

[2] Afin de réduire la place prise par l'archivage du document complet, seul les objets qui ont été modifiés depuis la dernière archive sont archivés à titre individuel et une liste contenant le nom de la dernière version de chacun des objets suffit à stocker l'information correspondant à l'archivage.

Les outils ajoutés par DUPLEX à un environnement local standard, ne font appel qu'à des paquetages standard tels que X11 et TCP/IP [Comer 88]. La partie la plus basse de ces outils (interface avec le noyau) utilise des appels système Unix standard ce qui assure un portage aisé sur toutes plate-formes Unix. Sur des plate-formes autres que Unix, il conviendrait de réécrire cette interface. Ces contraintes sont faibles, ce qui permet de prendre en considération l'hétérogénéité du système distribué sous-jacent. L'environnement DUPLEX est portable. Actuellement, des versions pour Linux, SunOs et Solaris existent et sont utilisées dans le cadre de notre laboratoire.

L'environnement utilisateur par lui-même est peu volumineux, puisqu'il utilise bon nombre d'outils déjà existants. Il est ainsi possible de stocker tout l'environnement sur une disquette, et ainsi de l'installer lors d'un déplacement. Pour pouvoir être opérationnel il suffit de connaître le nom du document et l'adresse du service de noms. Le nom du document est connu par définition et l'adresse du service de noms peut être obtenue facilement sur le site habituel de l'utilisateur.

Finalement, si l'on considère l'aspect panne, signalons que grâce à la séparation en deux niveaux (noyau et environnement utilisateur), les pannes dans le système distribué sont proprement décomposées et leur impact sur l'application d'édition coopérative peut être confiné :

**La panne d'un duplica :** ce type de panne est complètement transparent à l'utilisateur.

**La panne de communication entre noyau et utilisateur :** Deux types de panne peuvent avoir lieu : premièrement, une panne qui survient durant une opération avec le noyau. Elle est traitée par le protocole qui assure l'atomicité des opérations au niveau des objets du noyau. Deuxièmement, une partition est responsable d'une perte de liaison entre le noeud utilisateur et certains noeuds du noyau. Cette panne est détectée au niveau de l'utilisateur qui dans ce cas de figure ne pourra pas accéder à certains objets tant que la partition persiste. Ceci n'empêche nullement l'utilisateur de travailler sur les copies de son environnement ou bien d'atteindre d'autres objets du noyau. Le site utilisateur n'est pas complètement fonctionnel, mais permet quand même de travailler.

**La panne d'un noeud utilisateur :** Ceci a lieu lors d'une panne matérielle ou logicielle du site d'un utilisateur (e.g., son système de fichier est inopérant). Si cela survient durant une opération avec le noyau, la tolérance aux pannes garantit l'atomicité de l'opération. Ainsi, les effets sont confinés au noeud de l'utilisateur et la panne n'affecte pas le travail des autres utilisateurs.

# Conclusion et perspectives

Prendre en considération les réseaux à grande échelle est important à l'heure actuelle où les média déclinent Internet sous toutes ses formes. Seulement, Internet n'est pas l'extrapolation d'un réseau local, de la même manière qu'un pont n'est pas l'extrapolation d'une passerelle.

Il faut en effet tenir compte des pannes de sites ou de liens, des communications de mauvaise qualité et des partitions du réseau. Une fois ces contraintes fixées, l'outil d'édition coopérative se doit de rendre ces désagréments les plus transparents possible à l'utilisateur, ou du moins lui permettre de continuer à travailler, éventuellement dans un mode dégradé. Ceci passe bien entendu par des méthodes de travail différentes de celles utilisées sur un réseau local. En particulier, l'asynchronisme est le mode de fonctionnement privilégié dans un environnement où souvent, les utilisateurs dispersés sur Internet appartiennent à des fuseaux horaires différents.

Le travail effectué dans cette thèse propose différents concepts et mécanismes permettant de prendre en considération la large échelle.

Premièrement un nouveau critère de cohérence a été défini, la NTR-linéarisabilité qui possède la même propriété de localité que la linéarisabilité. La NTR-linéarisabilité considère l'ordre causal plutôt que le temps-réel, notion qui est difficilement exploitable dans un système distribué. La mise en œuvre de la NTR-linéarisabilité est basée sur un protocole non bloquant et efficace. La cohérence est contrôlée par une décomposition dynamique du document en parties indépendantes sur lesquelles est appliqué le critère de cohérence. Il est ainsi possible d'augmenter ou de diminuer la concurrence des actions réalisées sur le document. Comme cette modification est dynamique et dirigée par les utilisateurs, la cohérence peut être plus lâche au début de la collaboration pour se resserrer lorsque la maturité du document le nécessite. Finalement, au lieu de traiter le problème de la cohérence au niveau des opérations lecture-modification-écriture qui, compte tenu de la durée de la phase modification, peuvent s'étendre sur une longue période, la cohérence est traitée au niveau des opérations lecture et écriture.

L'atomicité des opérations lecture-modification-écriture est traitée au niveau de l'application ce qui permet de prendre en considération les connaissances et les désirs réels des utilisateurs. Plusieurs politiques de contrôle de concurrence (pessimiste ou optimiste) sont offertes à l'utilisateur, lui permettant de choisir en fonction de la partie de document, des coauteurs et de la maturité du document, celle qui lui semble la plus adaptée.

Deuxièmement, l'architecture de l'environnement DUPLEX se décompose en un noyau partagé, responsable de la gestion du document partagé, et en des environnements utilisateurs qui sont des espaces de travail privés de chacun des utilisateurs, permettant à l'utilisateur de travailler la majeure partie du temps localement. Les accès au noyau sont effectués pour faire une copie locale d'un objet du noyau, ou pour mettre à jour un objet du noyau à partir d'une copie locale, lorsque l'utilisateur juge ses modifications satisfaisantes. La grande indépendance du format dans lequel les parties d'un document sont stockées au niveau du noyau, et celui dans lequel elles sont manipulées au niveau de l'environnement local de l'utilisateur, permet à ce dernier de quasiment conserver son environnement de travail habituel.

La mise en œuvre de l'environnement d'édition coopérative DUPLEX a permis de valider les mécanismes et concepts cités ci-dessus, prouvant d'une part la faisabilité et d'autre part l'utilité d'un tel outil.

Les mécanismes et concepts de DUPLEX permettent d'entrevoir différentes évolutions du travail présenté dans cette thèse.

En premier lieu, sur le plan de l'édition coopérative, il est possible d'étendre l'indépendance de l'environnement de DUPLEX à des outils particuliers, en permettant la cohabitation[3] de plusieurs environnements d'éditions (e.g., LaTeX[Lamport 86, Knuth 84], Grif [Quint 86], Word[Mic 87], Framemaker[Fra 90], etc.) durant la phase d'édition. La phase de mise en page finale est de toute façon réalisée par un seul utilisateur, donc un seul outil, en fonction de lignes de conduite imposées par la conférence ou la maison d'édition.

En second lieu, il est possible d'utiliser l'architecture de DUPLEX pour d'autres usages que l'édition. En effet, la partie noyau de stockage est suffisamment générique pour permettre d'utiliser cette architecture pour d'autres types d'applications coopératives, comme par exemple le développement de logiciel mettant en jeu plusieurs partenaires.

Finalement, l'utilisation de PHENIX [Malloth 94] à la place de ISIS [ISIS 91], comme boîte à outils utilisée pour la mise en œuvre du protocole assurant la cohérence et la résistance aux pannes, rendra DUPLEX indépendant de tout produit propriétaire afin d'en faciliter la diffusion.

[3] Cf. Chapitre 9, Section 3.1

# Bibliographie


[Adve 90a] S.V. Adve & M.D. Hill. *Implementing Sequential Consistency In Cache-Based Systems.* In Proceedings of the 1990 International Conference on Parallel Processing, pages 47–50, August 1990.

[Adve 90b] S.V. Adve & M.D. Hill. *Weak-Ordering a New Definition and some Implications.* In Proceedings of the 17th Annual International Symposium on Computer Architecture, pages 2–14, May 1990.

[Afek 89] Y. Afek, G. Brown & M. Merritt. *A Lazy Cache Algorithm.* In Proceedings of the 1st Symposium on Parallel Processing, pages 209–222, 1989.

[Ahamad 91] M. Ahamad, P.W. Hutto & R. John. *Implementing and Programming Causal Distributed Shared Memory.* In Proceedings of the The 11th International Conference on Distributed Computer Systems, pages 274–281, 1991.

[Amir 92] Y. Amir, D. Dolev, S. Kramer & D. Malki. *Transis: A Communication Sub-System for High Availability.* In Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS'22), pages 76–84, July 1992.

[Armstrong 92] S. Armstrong, A. Freier & K. Marzullo. *Multicast Transport Protocol.* RFC 1301, 1992.

[Attiya 91] H. Attiya & J. Welch. *Sequential consistency vs. linearizability.* In Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures, pages 304–325, July 1991.

[Bair 85] J.H. Bair. *The need for collaboration tools in offices.* In proceedings of the AFIPS'85, Office Automation Conference, pages 59–68, February 1985.

[Banerjee 87] J. Banerjee, H.-T. Chou, J.F. Garza, W. Kim, D. Woelk, N. Ballou & H.-J. Kim. *Data model issues for object oriented applications.* ACM Transactions on Office Information Systems, pages 3–26, 1987.

[Barrett 90] P. Barrett, P. Bond, A. Hilborne, L. Rodrigues, D. Seaton, N. Speirs & P. Veríssimo. *The Delta-4 Extra performance architecture (XPA).* In Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS'20), Newcastle-UK, June 1990. IEEE.

[Beck 93] E.E. Beck. Computer supported cooperative writing, chapitre Chapter 6: A Survey of Experiences of Collaborative Writing, pages 87–112. Springer-Verlag, 1993.

[Bernstein 81] P.A. Bernstein & N. Goodman. *Concurrency Control in Distributed Database Systems.* ACM Computing Surveys, vol. 13, no. 2, pages 185–221, june 1981.

[Bier 91] E.A. Bier & S. Freeman. *MMM: A User Interface Architecture for Shared Editor on a Single Screen.* In Proceedings of the 4th ACM symp. on User Interface Software and Technology, ACM SIGGRAPH/SIGCHI, pages 79–86, November 1991.

[Bier 92] E.A. Bier, S. Freeman & S. Pier. *MMM: The Multi-Device Multi-User Multi-Editor.* In Proceedings of the ACM CHI'92 Conference on Human Factors in Computing Systems, ACM SIGCHI, pages 645–646, May 1992.

[Birman 93] K. Birman. *The Process Group Approach to Reliable Distributed Computing.* Communications of the ACM, December 1993.

[Borghoff 93] U.M. Borghoff & G. Tegge. *Application of Collaborative Editing to software-Engineering Projects.* ACM SIGSOFT, vol. 18, no. 3, pages 56–64, July 1993.

[Comer 88] D.E. Comer. Internetworking with TCP/IP: Principles, protocols, architecture. Prentice Hall, Stevenage, 1988.

[Conklin 88] J. Conklin. *gIBIS: A hypertext tool for exploratory policy discussion.* In Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'88), pages 140–152, Portland, Oregon, 1988. ACM Press.

[Cristian 91] F. Cristian. *Understanding Fault-Tolerant Distributed Systems.* Communications of the ACM, vol. 34, no. 2, pages 56–78, February 1991.

[Decouchant 93] D. Decouchant, V. Quint, M. Riveill & I. Vatton. *Griffon: A Cooperative, Structured, Distributed Document Editor.* Rapport technique num. R. R. 20, Bull-IMAG, Grenoble, 1993.

[Deering 89] S.E. Deering. *Host extensions for IP multicasting.* RFC 1112, 1989.

[Dubois 86] M. Dubois, C. Scheuring & F. Briggs. *Memory Access Buffering in Multiprocessors.* In Proceedings of the 13th Annual International Symposium on Computer Architecture, pages 434–442, June 1986.

[Dykstra 91] E.A. Dykstra & R.P. Carasik. *Structure and support in cooperative environments: The Amsterdam Conversation Environment.* International Journal of Man Machine Studies, vol. 34, no. 3, pages 419–434, 1991.

[Ede 90] L. Ede & A. Lunsford. Singular texts / plural authors: Perspectives on collaborative writing. Southern Illinois University Press, Carbondale, 1990.

[ElAbaddi 89] A. ElAbaddi & S. Toueg. *The Group Paradigm for Concurrency Control Protocols.* ACM Transaction on Knowledges in Data-bases Engineering, vol. 1, no. 3, April 1989.

[Ellis 89] C.A. Ellis & S.J. Gibbs. *Concurrency Control in Groupware systems.* In proceedings of the ACM SIGMOD '89 Conference on the Management of Data, May 1989.

[Ellis 91] C.A. Ellis, S.J. Gibbs & G.L. Rein. *Groupware - Some Issues and Experiences.* Communications of the ACM, vol. 34, no. 1, pages 38–58, January 1991.

[Fish 88] R.S. Fish, R.E. Kraut & M.D.P. Leland. *Quilt: A Collaborative Tool for Cooperative Writing.* In Proceedings of ACM International Conference on Office Information Systems, volume 9(2-3), pages 30–37, March 1988.

[Fisher 85] M. Fisher, N. Lynch & M. Patterson. *Impossibility of distributed consensus with one faulty process.* Journal of the ACM, vol. 32, no. 2, pages 274–382, April 1985.

[Fra 90] Frame Technology Corporation. *Manuel de référence FrameMaker*, 1990.

[Garreth 86] N.L. Garreth, K.E. Smith & N. Meyrowitz. *Intermedia: Issues, Strategies, and tactics in the Design of a Hypermedia Document System.* In Proceedings of the Conference on Computer Supported Collaborative Work (CSCW'86), 1986.

[Gharachorloo 90] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta & J. Hennessy. *Memory Consistency and Event Ordering in Scalable Shared-Memory Multiprocessors.* In Proceedings of the 17th Annual International Symposium on Computer Architecture, May 1990.

[Goodman 89] J. Goodman. *Cache Consistency and Sequential Consistency.* Rapport technique 61, SCI Committee, April 1989.

[Greif 86] I. Greif, R. Seliger & W. Weihl. *Atomic Data Abstraction in a Distributed Collaborative Editing System.* In Proceedings of the 13th ACM SIGACT/SIGPLAN annual symp. on Principles of Programming Languages, pages 160–172, January 1986.

[Greif 92] I. Greif, R. Seliger & W. Weihl. *A Case Study of CES: A Distributed Collaborative Editing System Implemented with Argus.* IEEE Transactions on Software Engineering, vol. 18, no. 9, pages 827–839, September 1992.

[Herlihy 90] M. Herlihy & J. Wing. *Linearizability A Correctness Condition for Concurrent Objects.* ACM Transactions on Programming Languages and Systems, vol. 12, pages 463–492, July 1990.

[Hutto 90] P.W. Hutto & M. Ahamad. *Slow Memory: Weakening Consistency to Enhance Concurrency in Distributed Shared Memories.* In Proceedings of the The 10th International Conference on Distributed Computer Systems, pages 302–311, 1990.

[ISIS 91] Isis Distributed Systems. *The Isis Distributed Toolkit*, version 3.0, user reference manual edition, 1991.

[Knister 90] M.J. Knister & A. Prakash. *DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors.* In Proceedings of International Conference on Computer-Supported Cooperative Work (CSCW'90), pages 343–355, October 1990.

[Knister 93] M. Knister & A. Prakash. *Issues in the Design of a Toolkit for Supporting Multiple Group Editors.* Computing Systems, vol. 6, no. 2, pages 135–166, 1993.

[Knuth 84] D.E. Knuth. The texbook. Addison-Wesley, 1984.

[Koszarek 90] J.L. Koszarek, T.L. Lindstrom, J.R. Ensor & S.R. Ahuja. *A multi-user document review tool.* In S Gibbs & A. A. Verrighn-Stuart, editeurs, Proceedings of IFIP WG8.4 Conference on Multi-User Interfaces and Applications, Crete, 1990. North Holland.

[Lamport 78] L. Lamport. *Time, Clock and the Ordering of Events in a Distributed System.* Communication of the ACM, vol. 21, no. 7, pages 558–565, July 1978.

[Lamport 79] L. Lamport. *How to Make a Multiprocessor Computer that Correctly Executes Multiprocess Programs.* IEEE Transactions on Computers, vol. C-28, no. 9, pages 690–691, September 1979.

[Lamport 86] L. Lamport. Latex : a document preparation system : user's guide and reference manual. Addison-Wesley, 1986.

[Lampson 69] B.W. Lampson. *Dynamic protection structures.* In Proceedings of AFIPS Fall joint Computer Conference, 1969.

[Lampson 71] B.W. Lampson. *Protection.* In Proceedings of 5th Annual Princeton Conference on Information Science Systems, 1971.

[Leland 88] M.D.P. Leland, R.S. Fish & R.E. Kraut. *Collaborative document production using Quilt.* In Proceedings of International Conference on Computer-Supported Cooperative Work (CSCW'88), pages 206–215, September 1988.

[Lewis 88] B.T. Lewis & J.D. Hodges. *Shared books: Collaborative Publication Management for an Office Information System.* In Proceedings of ACM International Conference on Office Information Systems, volume 9(2-3), pages 197–204, March 1988.

[Lubich 90] H. Lubich & B. Plattner. *A Proposed Model and Functionality Definition for a Collaborative Editing and Conferencing System.* In Proceedings of IFIP WG 8.4 Conference on Multi-User Interfaces and Applications, pages 215–232. North-Holland, September 1990.

[Lugeon 93] J.-C. Lugeon & A. Sandoz. *Sharing a Small Domain in a Large Distributed File System.* Rapport technique DI-LSE 93-6, École Polytechnique Fédérale de Lausanne, Département d'informatique, Lab. de Syst. d'Exploitation, 1993.

[Malloth 94] Ch. Malloth & A. Schiper. *View Synchronous Communication in the Internet.* Tech. Rep. 94/84, École Polytechnique Fédérale de Lausanne, Département d'informatique, October 1994.

[Malone 86] T.W. Malone, K.R. Grant & F.A. Turbak. *The Information Lens: An intelligent system for information sharing in organizations.* In Proceedings of the SIGCHI Human Factors in Computing Systems, pages 1–8, Boston, Mass, 1986. Association for Computing Machinery.

[Malone 87] T.W. Malone, K.R. Grant, K-Y. Lai, R. Rao & D. Rosenblitt. *Semi-structured messages are surprisingly useful for computer-supported coordination.* ACM Transactions on Office Information Systems, vol. 5, no. 2, pages 115–131, 1987.

[Mattern 89] F. Mattern. *Virtual Time and Global States of Distributed Systems.* In Parallel and Distributed Algorithms, pages 215–226. Elsevier Science Publishers B.V. (North-Holland), 1989.

[Mavronicolas 91] M. Mavronicolas & D. Roth. *Sequential consistency and linearizability: Read/Write Objects.* In Proceedings of the 29th Annual Allerton Conference on Communication, Control and Computing, pages 683–692, October 1991.

[McGuffin 92] L. McGuffin & M. Olson. *A Shared Electronic Workspace.* Rapport technique 45, CSMIL, The University of Michigan, 1992.

[Mic 87] Microsoft Corporation. *Manuel de référence de Microsoft Word*, 1987.

[Miles 91] V.C. Miles, C.W. Johnson, J.C. McCarthy & M.D. Harrison. *Supporting prediction in complex dynamics systems.* In Proceedings of the HCI'91 Conference. Springer-Verlag, August 1991.

[Miles 93] V.C. Miles, J.C. McCarthy, A.J. Dix, M.D. Harrison & A.F. Monk. Computer supported cooperative writing, chapitre Chapter 8: Reviewing Designs for a synchronous-asynchronous group editing environment, pages 137–160. Springer-Verlag, 1993.

[Minor 93] S. Minor & B. Magnusson. *A Model for Semi-(a)Synchronous Collaborative Editing.* In Proceedings of the 3rd European Conference on Computer Supported Cooperative Work, September 1993.

[Misra 89] J. Misra. *Axioms for Memory Access in Asynchronous Hardware Systems.* ACM Transactions on Programming Languages and Systems, vol. 8, no. 1, pages 142–153, January 1989.

[Neuwirth 90] C.M. Neuwirth, D.S. Kaufer, R. Chandhok & J. Morris. *Issues in the design of computer support for co-authoring and commenting.* In Proceedings of International Conference on Computer-Supported Cooperative Work (CSCW'90), 1990.

[Newman-Wolfe 91a] R.E. Newman-Wolfe & H.K. Pelimuhandiram. *MACE: A Fine Grained Concurrent Editor.* In Proceedings of the ACM SIGOIS Conference on Organizational Computing Systems, ACM SIGOIS, pages 240–254, 1991.

[Newman-Wolfe 91b] R.E. Newman-Wolfe, C.L. Ramirez, H. Pelimuhandiran, M. Montes, M. Webb & D.L. Wilson. *A Brief Overview of the DCS Distributed Conferencing System.* In Proceedings of the Usenix conference USENIX 91 Summer, pages 437–452, June 1991.

[ODA 88] ODA. *Information Processing - Text and Office Systems - Office Document Architecture (ODA).* ISO, 1988.

[Pendergast 90] M.O. Pendergast & D. Vogel. *Design and Implementation of a PC/LAN-Based Multi-User Text Editor.* In Proceedings of IFIP WG 8.4 Conference on Multi-User Interfaces and Applications, pages 195–206. North-Holland, September 1990.

[Posner 92] I.R. Posner & R.M. Baeker. *How people write together.* In Proceedings of the 25th Hawaii International Conference on System Sciences, volume IV, pages 127–138, January 1992.

[Quint 86] V. Quint & I. Vatton. *Grif: an Interactive System for Structured Document Manipulation.* In Proceedings of the International Conference on Text Processing and Document Manipulation, pages 200–213, 1986.

[Raynal 93] M. Raynal & M. Mizuno. *How to find his way in the jungle of consistency criteria for distributed shared memories.* In Proceedings of the 4th IEEE Workshop on Future Trends of Computing Systems, 1993.

[Romkey 88] J.L. Romkey. *Nonstandard for transmission of IP datagrams over serial lines: SLIP.* RFC 1055, 1988.

[Sarin 85] S. Sarin & I. Greif. *Computer-based real-time conferencing systems.* IEEE Computer, vol. 18, no. 10, pages 33–45, 1985.

[Scheuring 87] C. Scheuring & M. Dubois. *Correct Memory Operation of Cache-Based Multiprocessors.* In Proceedings of the 14th Annual International Symposium on Computer Architecture, pages 234–243, June 1987.

[Scheuring 89] C. Scheuring. *Access Ordering and Coherence in Shared Memory Multiprocessors.* PhD thesis, University of Southern California, May 1989.

[Schiper 93] A. Schiper & A. Sandoz. *Uniform reliable multicast in a virtually synchronous environment.* In Proceedings of 13th IEEE International Conference on Distributed Computing Systems, May 1993.

[SGML 86] SGML. *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML).* ISO, 1986.

[Sheth 90] A. Sheth & M. Rusienkiewicz. *Management of interdependent data: specifying dependency.* Proceedings of the 1st Workshop on the Management Of Replicated Data, November 1990.

[Sinha 93] H.S. Sinha. *Mermera: Non-coherent Distributed Shared Memory For Parallel Computing.* PhD thesis, Boston University, 1993.

[Stefik 87a] M. Stefik, G.G. Bobrow, G. Foster, S. Lanning & D. Tartar. *WYSIWIS revisited: Early experiences with multiuser interfaces.* ACM Transactions on Office Information Systems, vol. 5, no. 2, pages 147–186, April 1987.

[Stefik 87b] M. Stefik, G. Foster, D. Bobrow, K. Kahn, S. Lanning & L. Suchman. *Beyond the chalkboard: Computer support for collaboration and problem solving in meetings.* Communications of the ACM, vol. 30, no. 1, pages 32–47, 1987.

[Weihl 89] W. E. Weihl. *Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types.* ACM Transactions on Programming Languages and Systems, vol. 11, no. 2, pages 249–282, April 1989.

[Winograd 86] T. Winograd & T. Flores. Understanding computers and cognition: A new foundation for design. Ablex, Norwood, New Jersey, 1986.

[Yoder 89] E. Yoder, R. Akscyn & D. McCracken. *Collaboration in KMS, a shared hypermedia system.* In Proceedings of the SIGCHI Human Factors in Computing Systems, pages 37–42, Austin, Texas, 1989. ACM Press.

## Curriculum vitae

Je suis né le 27 octobre 1963 à Perpignan en France. Ma prime scolarité se déroule au Maroc puis en France, où j'obtiens un baccalauréat scientifique en 1982. J'entreprends des études de mathématiques à l'Université St Charles à Marseille où j'obtiens une maîtrise en 1986. Après mon service national, je complète mon cursus par des études en informatique à Grenoble où j'obtiens en 1988 une maîtrise à l'université Joseph Fourrier à Grenoble, puis en 1989 un DEA à l'Institut National Polytechnique de Grenoble.

En septembre 1989, je suis engagé pendant deux ans, par le laboratoire d'informatique théorique à l'EPFL, comme assistant d'un cours postgrade consacré au parallélisme.

Depuis début 1992, je travaille au laboratoire de système d'exploitation à l'EPFL, partageant mes activités entre la tâche d'assistant et la recherche dans le domaine des systèmes répartis.