RS2.7 : un Canevas Adaptable de Services de Duplication

Stéphane Drapeau

Institut National Polytechnique de Grenoble

THÈSE

pour obtenir le grade de

Docteur de l'Institut National Polytechnique de Grenoble

Spécialité : Informatique : Systèmes et Communications

préparée au Laboratoire DTL/ASR de France Télécom R&D et
au Laboratoire Logiciels, Systèmes et Réseaux (LSR)

dans le cadre de l'Ecole Doctorale
Mathématiques, Sciences et technologies de l'information, Informatique

présentée et soutenue publiquement

par

Stéphane Drapeau

le 24 juin 2003

RS2.7 : un Canevas Adaptable de Services de Duplication

JURY :

M. Guy Mazaré	Président
M. Philippe Pucheral	Rapporteur
M. Michel Riveill	Rapporteur
Mme. Christine Collet	Directeur de thèse
Mme. Claudia Lucia Roncancio	Co-encadrant
M. Pascal Déchamboux	Co-encadrant
Mme. Kathleen Milsted	Examinateur

Remerciements

Le travail présenté dans cette thèse a été effectué au sein du département Architecture des Systèmes Répartis de la Direction des Techniques Logicielles (DTL/ASR) de France Télécom R&D et du Laboratoire Logiciels, Systèmes et Réseaux (LSR) de l'Institut d'Informatique et de Mathématiques Appliquées de Grenoble (IMAG). Je tiens à remercier ces deux organismes qui m'ont permis d'effectuer mon travail dans les meilleures conditions matérielles, scientifiques et techniques que l'on puisse espérer.

Je remercie très sincèrement Claudia Lucía Roncancio, maître de conférence à l'École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG), pour la grande disponibilité dont elle a toujours fait preuve, ainsi que pour ses conseils avisés qui m'ont permis d'avancer vers un résultat meilleur. Son encadrement m'aura été précieux tout au long de ces quatre années. Ce manuscrit doit beaucoup à ses suggestions et à ses relectures attentives.

Je remercie également Pascal Déchamboux, ingénieur recherche et développement à France Télécom R&D, pour les discussions que nous avons eues et qui m'ont permis d'avancer.

Je tiens à remercier Christine Collet, professeur à l'École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG), responsable du projet NODS, pour m'avoir accueilli dans son équipe et pour la confiance qu'elle m'a toujours témoignée.

J'exprime ma profonde gratitude à Monsieur Guy Mazaré, professeur à l'École Nationale Supérieur d'Informatique et Mathématiques Appliquées de Grenoble, pour l'honneur qu'il m'a fait en présidant mon jury de thèse. Je souhaite remercier Messieurs Michel Riveill, professeur à l'École Supérieure en Sciences Informatiques (ESSI) de l'Université de Nice Sophia Antipolis et membre de l'équipe Informatique Signaux et Systèmes de Sophia Antipolis (I3S), et Philippe Pucheral professeur à l'Université de Versailles et membre du laboratoire Parallélisme Réseaux Systèmes Modélisation de Versailles (PRiSM), pour avoir acceptés la lourde charge d'être rapporteur et pour leurs précieuses remarques qui m'ont permis d'améliorer ce manuscrit. Je remercie aussi Madame Kathleen Milsted, responsable du laboratoire DTL/ASR de France Télécom R&D, d'avoir acceptée de faire parti de mon jury.

J'adresse mes plus vifs remerciements à Elizabeth Peréz Cortés, maître de conférence à

l'Université Autonome Métropolitaine de Mexico (UAM), pour les nombreuses et houleuses discussions que nous avons eues et qui m'ont permis de comprendre bien des choses. Ses critiques constructives et ses encouragements m'ont permis de mener à bien cette thèse.

Mes remerciements vont naturellement à l'ensemble des gens que j'ai côtoyé durant ces trois années et qui m'ont aidé ou tout simplement rendu le déroulement de cette thèse agréable. Je citerai entre autres : Luciano Garcia Bañuelos pour nos longues discussions sur la duplication et la persistance mais également sur tout et n'importe quoi, Patricia Serrano Alvarado pour m'avoir fait découvrir l'informatique mobile, m'avoir laissé lui présenter à maintes reprises mes travaux et surtout m'avoir permis de découvrir son merveilleux pays, le Mexique, Mikaël Beauvois pour nos réflexions sur la composition et la relecture de ce manuscrit, Edgard Benitez pour m'avoir aidé à écrire mon premier article sur le préchargement à l'aide de techniques d'extraction de données, Alexandre Lefebvre pour avoir bien voulu consacrer de son temps à la relecture de mes articles et de ce manuscrit, Tanguy Nedelec pour le temps qu'il a consacré à la relecture de ce manuscrit et pour ses remarques toujours très constructives, Phuong-Quynh Duong pour nos discussions sur la tolérance aux fautes, Tony, Khalid, Rafael et Gennaro pour les franches rigolades que nous avons pu avoir et également tous les autres : Olivier, Caty, Régine, Manuel, Lizbeth, Achraf, Trinh, Genoveva, Jose Luis, Christelle, Romain, Mourad et la communauté Mexicaine Grenobloise.

Mes pensées les plus chères vont à mes parents, mon frère Thierry et mes amis, dont l'affection, la confiance et le soutien ne me firent jamais défaut.

Je ne saurai terminer sans adresser tous mes remerciements à Patricia pour le précieux soutien et ses constants encouragements qu'elle m'a apportés durant les quatre années de cette thèse.

Le 23 juin 2003.
Grenoble.

Table des matières

Table des figures

Chapitre 1

Introduction

Sommaire

Ce premier chapitre situe brièvement le contexte de ce travail de thèse (section 1.1), en donne la problématique (section 1.2), énonce les objectifs visés (section 1.3), présente la démarche et les principales contributions apportées (section 1.4) et finalement donne l'organisation de ce document (section 1.5).

1.1 De l'importance et de la difficulté de la duplication

L'évolution des systèmes informatiques pendant ces dernières années est caractérisée par une tendance très forte à la décentralisation et les configurations réparties sont de plus en plus répandues. Cette tendance entraîne des besoins en mécanismes assurant la disponibilité et la fiabilité des données pour fournir la tolérance aux fautes et/ou le passage à l'échelle. La duplication d'informations et/ou de services est un de ces mécanismes, car elle apporte essentiellement deux bénéfices :

– D'une part, une amélioration des performances. Si les données se trouvent en un seul lieu, la machine qui les gère ainsi que le réseau pour y accéder constituent un goulot d'étranglement pour les clients. La duplication permet de placer les données de manière

à augmenter les performances et diminuer les coûts d'accès aux données pour chacun des clients.
- Et d'autre part, une meilleure sûreté de fonctionnement. On peut distinguer deux aspects :
  - La disponibilité des données. Plus le nombre de machines indépendantes du point de vue des pannes et possédant une copie d'une donnée est important, plus le temps pendant lequel cette donnée est accessible sera grand.
  - La fiabilité des données. La duplication permet d'éviter de perdre des modifications de données lors d'une panne et permet de ne pas perdre des données en cas de pannes définitives. Cette notion est différente de la disponibilité et est même parfois incompatible lorsqu'on y ajoute les problèmes de cohérence entre copies.

Malheureusement, malgré tous ces avantages, la duplication a un coût car il est nécessaire de maintenir une "certaine cohérence" entre les copies. En effet, la duplication rencontre une contradiction majeure : assurer la cohérence des copies tout en conservant des performances acceptables. Un protocole assurant une cohérence forte est simple mais ne passe pas à l'échelle, d'où des recherches actives sur des cohérences affaiblies. Dans tous les cas de figure, étant donné que chaque copie est indépendante, des protocoles sont nécessaires afin d'assurer la mise en cohérence des copies. Ils doivent également garantir que les accès concurrents sur les différentes copies se déroulent convenablement.

Cinq grands domaines de l'informatique se sont particulièrement intéressés à la duplication : les systèmes à communication de groupe, les mémoires virtuelles réparties, les systèmes de gestion de bases de données réparties, les systèmes de fichiers répartis, ainsi que les plateformes à objets. Malheureusement, nous pouvons remarquer que malgré le nombre important de travaux [GHOS96, KA00a, Kin99, DR01], il n'y a que très peu de rapprochement entre les projets menés dans ces différents domaines. Ainsi, de nombreuses solutions sont proposées au travers de systèmes divers, chacun ayant fait ses propres choix, le plus souvent de manière définitive. Certaines solutions sont orientées vers une catégorie d'applications précise, d'où, finalement, le nombre et la variété des systèmes disponibles. Ces choix particuliers conditionnent entre autre la cohérence entre les copies, les performances, la tolérance aux fautes, la transparence ou bien encore la difficulté de programmation. Ainsi, le choix et la mise en œuvre de la duplication pour une application particulière est une tâche difficile pour le programmeur.

1.2 Un nouveau paradigme de programmation : la séparation des préoccupations

L'étude des techniques de développement classiques comme les méthodes itératives font apparaître qu'une des grandes difficultés de l'informatique vient d'un problème organisationnel lié à un entrelacement des aspects métiers ou fonctionnels (le code applicatif) et des aspects techniques ou non fonctionnels (notamment la duplication) d'une application.

Ainsi, de nombreux travaux de recherche académique ou industriels développent actuel-

lement des outils ou des infrastructures permettant de séparer ces aspects afin que le développeur d'applications puisse se concentrer sur le code fonctionnel plutôt que sur des tâches non-spécifiques à une application. Les intergiciels (*middlewares*) comme CORBA [OMG97, OMG98], EJB [DYK01], COM/DCOM [Mic95], .net [Mic02]), etc, en sont des bons exemples. Un intergiciel est une couche logicielle qui se situe entre le système d'exploitation et les applications. L'objectif d'un intergiciel est de fournir les aspects non fonctionnels sous forme de services (services n'existant pas à l'origine dans le système) et également d'être une couche d'abstraction pour la programmation d'applications. Le premier service fourni est généralement, un système de communication (dit aussi "bus logiciel") entre les applications utilisant l'intergiciel. On peut également trouver, selon les intergiciels, un service de gestion de transactions, un service de persistance ou encore un service de duplication.

Cependant, la séparation des préoccupations n'est pas chose aisée. Par exemple, dans CORBA, les services doivent être explicitement utilisés par le programmeur d'application. La spécification EJB, qui gère les aspects non fonctionnels de persistance et de transaction, permet une gestion implicite de ces aspects. Cependant, le programmeur est obligé d'utiliser ce qui est proposé par la plate-forme. Pour un même aspect non-fonctionnel le choix est limité.

Les travaux cités ci-dessus, que l'on peut qualifier d'empiriques, se font en parallèle de travaux théoriques plus généraux. Ainsi, depuis quelques années, l'intérêt de la communauté scientifique pour la séparation des préoccupations s'est accru de manière considérable et de nombreuses tendances et techniques visant à la résoudre ont vu le jour. Parmi les principales, on peut citer la programmation orientée aspects (*aspect-oriented programming*) [KLM+97, BS99, ASP01], la programmation par composants aspectuels (ou collaborations aspectuelles) [LLM99, LOML01], la programmation orientée sujets (*subject-oriented Programming*) [MHO96, OKK+96, HOT97] ou la programmation par intention (*intentional programming*) [Sim95]. Toutes ces approches visent à appliquer aux langages de programmation le principe bien connu en génie logiciel de séparation des préoccupations (*separation of concerns*) [CE99]. Ces approches suggèrent de concevoir des entités logicielles indépendantes et fournissent des moyens pour les assembler.

En conclusion, la séparation des préoccupations et l'approche par services permettent au développeur d'applications de s'abstraire de l'aspect duplication lors de ses développements afin de lui faciliter la tâche. Cependant, elle souffre d'une limitation majeure. En effet, il semble très difficile, voir impossible, de fournir un service/aspect générique de duplication pouvant être paramètré afin d'être utilisé dans différents contextes d'exécution ou couvrant l'ensemble des protocoles existants.

1.3 De la nécessité d'offrir un support adaptable de la duplication

L'approche **adaptable** prônée par les supports ouverts semble plus prometteuse afin d'obtenir des aspects/services de duplication appropriés au contexte d'exécution et couvrant l'en-

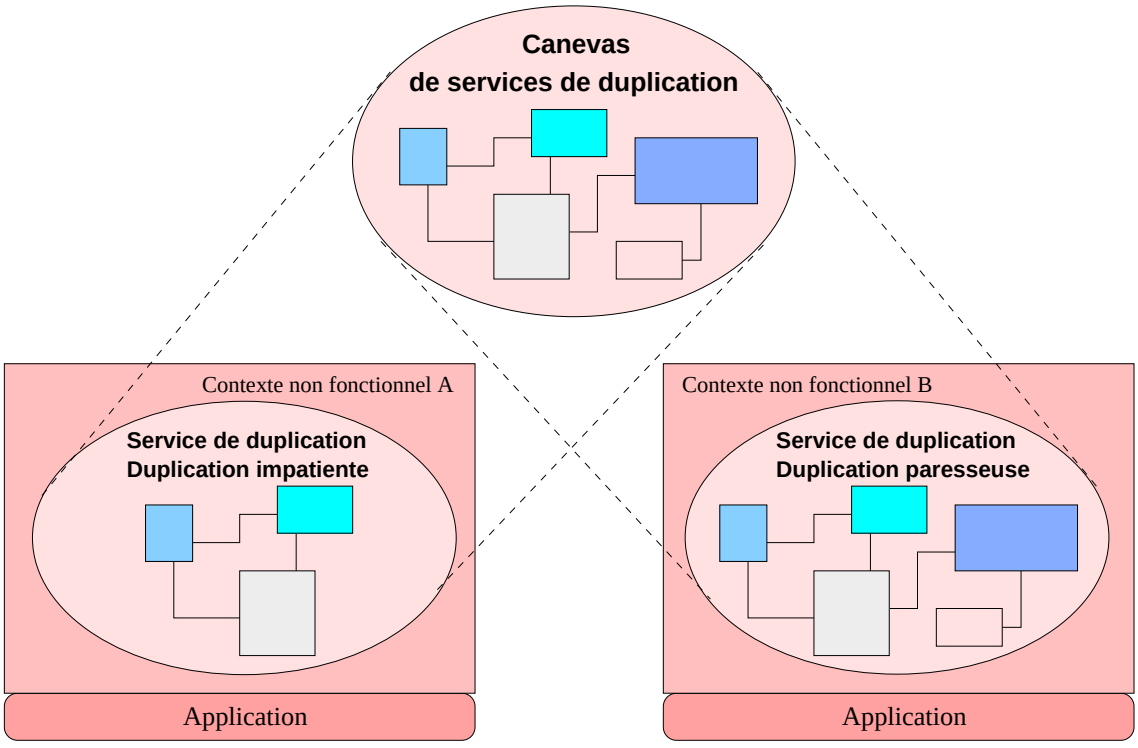


FIG. 1.1 – Développement de services de duplication à partir d'un canevas

semble des protocoles existants. En effet, à l'inverse des supports monolithiques, les supports ouverts présentent une architecture qui permet leur modification en vue de répondre à des besoins particuliers. Ceux-ci peuvent se présenter sous diverses formes [Lob00] :

- Un support est extensible, s'il offre la possibilité d'étendre les services qu'il met à disposition des utilisateurs. Les fonctions déjà existantes ne sont pas affectées par l'extension du système et leur comportement ne peut pas être modifié.
- Un support est configurable, s'il permet de spécifier l'utilisation d'une fonction particulière parmi un ensemble de fonctions disponibles dans le système. Il est possible d'infléchir le comportement du système sans toutefois pouvoir définir de nouveaux comportements.
- Un support est adaptable si les fonctions internes déjà présentes dans le système peuvent être remplacées par de nouvelles fonctions. Il est possible de modifier le comportement du système afin de supporter de nouveaux besoins.

Ainsi, selon nous, un support adaptable de la duplication doit permettre d'obtenir des services de duplication appropriés aux ressources offertes, au contexte non fonctionnel (transactionnel, tolérant aux fautes, persistant, etc) et prendre en compte les contraintes et les protocoles spécifiques à chaque domaine (bases de données réparties, mémoires virtuelles ré-

parties, etc). Pour cela, notre objectif est de définir un canevas[1] de services de duplication, c'est à dire la structure générale d'un service de duplication, pouvant ensuite être instanciée de diverses façons afin d'obtenir le service de duplication adéquate aux ressources, au contexte non fonctionnel et au domaine (figure 1.1).

L'intérêt d'un support adaptable de la duplication nous semble important si l'on considère l'évolution actuelle des systèmes informatiques. En effet, les systèmes informatiques à grande échelle ou mobiles ont introduit de nouvelles problématiques qu'un support adaptable peut aider à résoudre. Ces environnements présentent une grande hétérogénéité, variabilité et évolution, aussi bien au niveau des moyens d'exécution qu'au niveau des besoins à un moment donné. Les ressources offertes peuvent être extrêmement différentes selon que l'on utilise un assistant personnel, un ordinateur portable, une station de travail ou un serveur. De plus, les éléments constitutifs du système et le système même sont soumis à d'importantes variations au cours du temps. Les performances du réseau peuvent également varier. Dans les environnements "classiques" le développement et l'exécution d'applications s'effectuent en supposant que le support d'exécution est connu à l'avance. Dans les environnements où de nombreux changements peuvent intervenir, ceux-ci doivent être pris en compte. Pour cela, un support adaptable de la duplication pouvant être adapté dynamiquement selon les changements survenant dans l'environnement semble tout à fait approprié.

En conclusion, notre objectif est de donner la propriété d'adaptabilité à l'aspect duplication, sous la forme d'un canevas adaptable de services de duplication. Par la suite, ce canevas pourra être utilisé de manière statique ou dynamique afin d'adapter et de construire un service de duplication approprié.

1.4 Démarche et contributions de notre travail

Classification des protocoles de duplication.

Les protocoles de duplication existants dans la littérature sont nombreux et variés. Ils diffèrent suivant le niveau de cohérence souhaité entre les copies (cohérence forte, cohérence plus ou moins affaiblie), l'environnement (duplication de serveurs, duplication dans un contexte mobile composé d'assistants personnels, etc) et le domaine (systèmes de communication de groupe, systèmes de gestion de bases de données réparties, mémoires partagées réparties). Cependant, nous avons dégagé leurs **points communs** afin de pouvoir les comparer. Ainsi, nous proposons une grille de classification des différents protocoles de duplication existant dans la littérature.

Nous avons également dégagé les **spécificités** propres à chaque domaine : par exemple les transactions pour les systèmes de gestion de bases de données et les modèles de cohérence

[1] Un canevas définit la structure générale d'une application générique [Joh97, Rog97]. Il permet de spécifier des décisions de conception et il doit s'adapter à toutes les applications d'un domaine donné. Il est instancié afin d'obtenir une application particulière répondant aux besoins.

pour les mémoires partagées réparties.

RS2.7, un canevas de services de duplication.

Etant donné la difficulté de mettre en œuvre la duplication, des travaux proposent déjà des supports adaptables de duplication. Nous avons mis en lumière certaines **limites** de ces supports :

- D'une part, l'isolation de la duplication par rapport aux aspects non fonctionnels n'est pas claire. Il en résulte, bien souvent, un manque d'adaptabilité du support de duplication par rapport à ces aspects. Il est difficile, voir impossible, d'utiliser le support de la duplication dans divers contextes non fonctionels (contexte transactionel, contexte des mémoires partagées réparties, etc).
- D'autre part, on remarque le manque d'adaptabilité du support dans tout ou partie des protocoles de duplication offerts. Bien souvent, il n'est pas possible de changer uniquement certaines fonctionnalités du protocole afin d'en obtenir de nouvelles plus appropriées.

Ces constatations nous ont conduit vers la définition d'un canevas de services de duplication, nommé RS2.7 [2]. Nos contributions portent sur trois axes : la modélisation des services de duplication pouvant être obtenus à partir de RS2.7, l'adaptabilité du canevas par rapport au contexte non fonctionnel et l'adaptabilité dans tout ou partie des protocoles de duplication.

Au vu de l'état de l'art sur les supports adaptables de la duplication, on note également qu'il n'y a pas consensus sur leur rôle et le service qu'ils doivent rendre. Ainsi, nous retenons comme **noyau minimal** d'un service de duplication deux tâches : la gestion du cycle de vie des différentes copies d'un même objet et la gestion de la mise en cohérence de celles-ci lorsque c'est nécessaire (nous appelons ce deuxième point protocole de cohérence locale dans la suite). Toute autre tâche n'est pas du ressort d'un service de duplication.

Modélisation des services de duplication.

RS2.7 permet d'obtenir des protocoles de cohérence locale variés correspondant à divers besoins. Par exemple, pour mettre en œuvre la tolérance aux fautes il est nécessaire d'avoir une cohérence forte entre les différentes copies, alors que pour améliorer les performances il peut être acceptable de laisser diverger légèrement celles-ci. Afin de caractériser les différents types de services pouvant être offerts par RS2.7 nous définissons la notion de **modèle de cohérence locale**. Un modèle de cohérence locale est la définition de comment l'utilisateur perçoit les différentes copies d'un même objet. Nous avons formalisé quatre types de modèle représentant, à notre avis, l'ensemble des situations envisageables. Ces quatre types de modèle sont : les modèles à copie unique, les modèles à copies divergentes, les modèles à copies

[2] RS sont les initiales de la traduction anglaise de Services de Duplication (**R***eplication* **S***ervices*). RS est également le nom d'un modèle de Porsche 911 appelé aussi RS2.7 (voir annexe A).

convergentes avec lecture sur les copies divergentes et les modèles à copies convergentes avec écriture sur les copies divergentes.

Chacun de ces modèles fait ressortir des besoins particuliers en ce qui concerne d'autres aspects non fonctionnels. C'est le **premier niveau d'interaction** entre la duplication et les autres aspects non fonctionnels. Par exemple, pour le modèle à copie unique, il est nécessaire de garantir que les mises à jour entre les différentes copies peuvent se faire de manière atomique. Ainsi, nous avons clairement définit le rôle de chacun des aspects intervenant lors de la gestion de la cohérence des copies d'un objet logique, ainsi que les interactions existant entre eux, afin de pouvoir les adapter les uns par rapport aux autres selon les besoins.

Adaptabilité de RS2.7 **par rapport au contexte non fonctionnel.**

Traditionnellement, le développeur d'une application s'appuie sur un **modèle de cohérence globale** spécifiant plus ou moins formellement la manière dont se comporte la mémoire ou les données suivant le contexte. Ces modèles sont implantés par des protocoles gérant les objets en prenant en compte la concurrence, la tolérance aux fautes ou bien encore la duplication. Cependant on peut remarquer que ces différents aspects non fonctionnels se trouvent mélangés dans le protocole de cohérence globale. Ceci limite l'évolution du code et sa réutilisabilité. En effet, un protocole de cohérence globale doit être repensé si la duplication est introduite et réciproquement, par exemple, la partie gérant la duplication doit être revue si le contrôle de concurrence change.

Notre objectif étant la séparation des préoccupations, nous avons isolé chacun des aspects non fonctionnels et plus particulièrement l'aspect duplication à l'intérieur des protocoles de cohérence globale. Ainsi, en présence de données dupliquées, nous soutenons qu'un protocole de cohérence globale est constitué, entre autre, d'un protocole de cohérence locale. Cependant, il est nécessaire de définir les interactions possibles entre un protocole de cohérence locale et les autres aspects non fonctionnels participant à la construction du protocole de cohérence globale. En faisant ainsi, il devient possible d'adapter le protocole de cohérence locale au protocole de cohérence globale. Ainsi, notre approche permet d'adapter un protocole de cohérence locale afin qu'il puisse être utilisé dans un contexte transactionnel, de mémoire partagées réparties, etc. De plus, nous proposons une formalisation de la cohérence entre les copies. Cette formalisation nous permet de montrer qu'un même protocole de cohérence locale peut être réutilisé pour mettre en œuvre différents protocoles de cohérence globale.

Cette position est novatrice car dans les différents domaines utilisant la duplication il existe la notion de modèle de cohérence globale (séquentiel, causale, PRAM, paresseuse dans les mémoires partagées réparties, sérialisabilité sur une copie dans les SGBD répartis), mais celle de cohérence locale n'a pas été mise en évidence. Ainsi, dans les propositions actuelles, la gestion de la duplication n'apparaît que dans les protocoles de cohérence globale. Elle se retrouve englobée dans la gestion de données (concurrence, répartition, etc) limitant l'adaptabilité de l'aspect duplication par rapport au contexte non fonctionnel.

Adaptabilité dans tout ou partie des protocoles de duplication.

Notre analyse de l'état de l'art sur les techniques de duplication nous a permis d'extraire différentes phases que l'on retrouve dans de nombreux protocoles. Ainsi, nous proposons une **décomposition structurelle** sous la forme d'un protocole abstrait de cohérence locale composé de cinq phases : une phase d'accès, de coordination, d'exécution, de validation et de réponse. La différence entre les protocoles est due à la manière dont chaque phase est implantée et l'ordre dans lequel elles apparaissent. Dans certains cas, des phases sont inexistantes, ou il peut y avoir des boucles, ou bien encore elles peuvent agir en parallèle.

A partir de la grille de classification des protocoles de duplication, on remarque également que les différents protocoles proposés dans la littérature ont en commun certaines fonctionnalités : moment de déclenchement de la synchronisation, détection et résolution des conflits, gestionnaire de rôle des copies, etc. Ainsi nous proposons une **architecture fonctionnelle** isolant les différents composants intervenant dans un protocole.

Ces factorisations structurelle et fonctionnelle des protocoles de cohérence locale, nous permettent d'obtenir l'adaptabilité et la réutilisabilité dans les protocoles.

En résumé, notre travail porte sur la compréhension de l'aspect duplication et de la proposition de points d'adaptabilité pour celui-ci. Nous proposons l'aspect duplication sous forme de boîtes blanches. Ainsi, notre travail peut être vu comme le point de départ de travaux plus généraux du domaine de la séparation des préoccupations portant sur la composition d'aspects. De part le fait que nous ouvrons l'aspect duplication, il devient possible de composer plus finement les différents aspects non fonctionnels et d'optimiser cette composition.

RS2.7 a été mis en œuvre. Le principe essentiel de la mise en œuvre est la construction de **chaînes de liaison** entre les différentes copies d'un même objet. Ces chaînes de liaison implantent un certain protocole de cohérence locale et une certaine gestion du cycle de vie. Une chaîne de liaison implante donc une instance de RS2.7, et donc un service de duplication. Elles sont construites en prenant en compte :

- Les interactions avec les autres aspects non fonctionnels afin de garantir les modèles de cohérence locale.
- Les interactions avec les autres aspects non fonctionnels afin de participer à la mise en œuvre des modèles de cohérence globale.
- La décomposition structurelle et la décomposition fonctionnelle proposées.

Notre validation porte sur la démonstration des caractéristiques d'adaptabilité offertes par RS2.7. Notre objectif n'est pas d'obtenir une implantation performante, car nous décomposons au maximum les protocoles de cohérence locale. Nous cherchons à montrer que RS2.7 permet d'obtenir des services très variés et convenant pour divers contextes non fonctionnels. Nous avons utilisé des services de duplication obtenus à partir de RS2.7 dans le contexte du projet européen PING (Platform for Interactive Networked Games).

1.5 Plan de la thèse

Ce document est organisé selon trois parties :

1. La première partie assoit la problématique de cette thèse. Elle est composée de deux chapitres. Dans un premier chapitre (chapitre 2), nous présentons les divers concepts et techniques de duplication existants. L'état de l'art embrasse trois principaux domaines de l'informatique répartie : les mémoires partagées réparties, les systèmes répartis à communication de groupe et les systèmes transactionnels. Dans un second chapitre (chapitre 3), nous présentons quelques travaux proposant certaines formes d'adaptabilité dans le support de la duplication. Ceci nous permet de mettre en évidence leurs points positifs, mais surtout leurs limites : le manque d'adaptabilité au contexte non fonctionnel et dans le service rendu.
2. La deuxième partie présente nos éléments de solution. Elle comporte quatre chapitres. Tout d'abord, dans un premier chapitre (chapitre 4) nous définissons le rôle de notre canevas adaptable de duplication. Dans un second chapitre (chapitre 5), nous définissons les différents types de services pouvant être obtenus à partir de RS2.7. Puis, dans le chapitre suivant (chapitre 6), nous présentons de quelle manière RS2.7 peut s'adapter à différents contextes non fonctionnels. Ensuite (chapitre 7), nous montrons comment nous proposons de factoriser la fonctionnalité duplication afin que notre canevas puisse offrir l'adaptabilité dans tout ou partie des services rendus.
3. La troisième partie traite de la validation de nos propositions. Cette partie se compose d'un chapitre (chapitre 8). Tout d'abord, nous présentons la mise en œuvre du canevas proposé, puis nous mettons en valeur le gain obtenu par rapport à l'existant afin de valider nos propositions.

Le dernier chapitre (chapitre 9) dresse un bilan du travail réalisé et évoque un certain nombre de perspectives.

Inventer en toute chose, c'est vouloir mourir à petit feu ; copier, c'est vivre.
Honoré de Balzac - Extrait de Pierre Grassou.

Première partie

Comprendre le problème : état de l'art

Chapitre 2

Concepts et techniques de duplication

Sommaire

Un système réparti, qu'il soit déployé sur un réseau dédié ("clusters", serveurs parallèles), sur un réseau d'entreprise (support au "workflow"), ou sur le réseau mondial (support aux systèmes de réservation ou aux jeux sur Internet) nécessite du partage de ressources par ses différents composants. Les techniques de partage qu'il utilise sont souvent basées sur la duplication d'informations et/ou de services. Dans ce contexte, un même objet[1] peut être dupliqué en autant de copies que nécessaire, chaque copie étant détenue et accédée par un ou plusieurs processus s'exécutant dans le système. Cependant, si la duplication améliore les performances et la tolérance aux fautes, le maintien de la cohérence entre les copies n'est pas chose aisée. Des protocoles se doivent de garantir une certaine cohérence entre les copies selon l'objectif recherché.

Trois grands domaines de l'informatique se sont particulièrement intéressés à la duplication : les systèmes de communication de groupe (SCG), les systèmes de gestion de bases de données répartis (SGBDR) et les mémoires partagées réparties (MPR). Cependant, nous pouvons remarquer que malgré le nombre important de travaux il n'y a que très peu de rapprochement entre eux. Ainsi, l'objectif de ce chapitre est de faire une synthèse des principaux

[1] Nous entendons objet au sens large, un objet peut être une page mémoire, une relation d'une base de données, un serveur de noms, etc.

résultats obtenus dans ces domaines en matière de support à la duplication, afin de mieux dégager les concepts sous-jacents essentiels. Cette étude nous permet de voir quels sont les points communs des différents protocoles, ainsi que les spécificités de chaque domaine. Nous ne traitons pas les systèmes à objets répartis (SOR) dans cet état de l'art, car bien souvent on y retrouve la notion d'action atomique équivalente aux transactions et nous préférons nous focaliser sur les SGBDR. Cependant, nous les traitons dans le chapitre suivant (chapitre 3) pour l'adaptabilité qu'ils offrent.

Le chapitre est organisé de la manière suivante. Nous commençons (section 2.1) par présenter les principales caractéristiques des protocoles de duplication nous paraissant importantes. La mise en évidence de ces caractéristiques nous offre des points de comparaison entre les protocoles proposés dans les différents domaines de la littérature. Ensuite, nous présentons la duplication dans les SCG (section 2.2), dans les SGBDR (section 2.3) puis dans les MPR (section 2.4). Ce chapitre se termine par nos conclusions (section 2.5).

2.1 Points remarquables des protocoles de duplication

Un protocole de duplication gère la cohérence entre les différentes copies d'un même objet tout en ayant pour objectif d'améliorer la fiabilité des données et/ou les performances (tant en écriture qu'en lecture) du système. Malheureusement, ces deux objectifs sont antagonistes (figure 2.1). Pour obtenir une bonne fiabilité, il est nécessaire d'avoir une cohérence forte ce qui pénalise les performances. A l'inverse pour obtenir de bonnes performances il est nécessaire de relâcher la cohérence, ce qui pénalise la fiabilité. Cela est d'autant plus vrai que l'on augmente le nombre de copies. En effet, dans le cas de la cohérence forte, un site accède toujours au dernier élément de la séquence globale des écritures. Ceci peut être réalisé par un protocole basé, par exemple, sur une diffusion atomique des valeurs, ou par le verrouillage global de la donnée avant l'ajout d'un élément à la séquence. Ces méthodes se basent sur un ordre total sur les écritures. Par contre dans le cas de la cohérence faible, on n'assure plus que la valeur lue sur un site (la dernière valeur de la séquence locale) est bien la dernière de la séquence globale. Cela est généralement réalisé par la construction d'un ordre partiel sur les écritures. Il est donc nécessaire de faire des compromis.

Définition 2.1 : *Cohérence forte pour objet dupliqué*

Il y a cohérence forte entre les copies d'un même objet s'il y a ordre total sur les écritures faites sur ces copies.

Définition 2.2 : *Cohérence faible pour objet dupliqué*

Il y a cohérence faible entre les copies d'un même objet s'il y a ordre partiel sur les écritures faites sur ces copies.

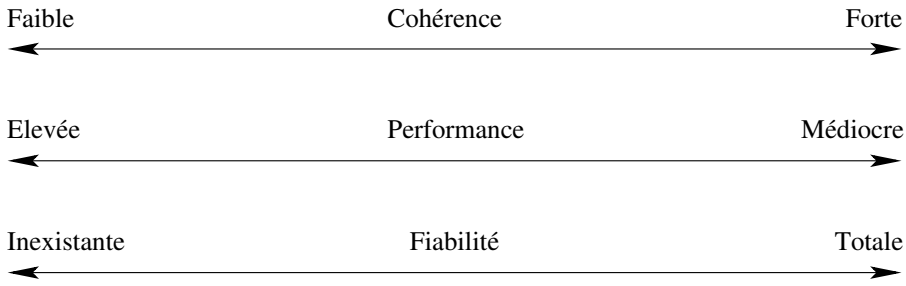


FIG. 2.1 – Compromis performance/fiabilité et répercussions sur la cohérence

De ce compromis performance/fiabilité résulte de nombreux protocoles de duplication mettant en œuvre une cohérence plus ou moins forte entre les copies que ce soit dans les SCG, les SGBDR ou les MPR. Afin de pouvoir comparer les solutions proposées dans ces différents domaines, nous présentons dans cette section les principales caractéristiques des protocoles de duplication nous paraissant importantes : nombre de copies concernées par une lecture ou une écriture (section 2.1.1), droits d'accès (section 2.1.2), moment de la synchronisation (section 2.1.3), initiative de la mise à jour (section 2.1.4), nature des mises à jour (section 2.1.5), topographie de la synchronisation (section 2.1.6), capture des mises à jour (section 2.1.7), gestion des conflits (section 2.1.8), prérequis sur le protocole de communication (section 2.1.9), gestion de la concurrence (section 2.1.10), gestion de la tolérance aux fautes (section 2.1.11), notion de copie (section 2.1.12) et transparence à la duplication (section 2.1.13). En fin de section, nous résumons l'ensemble des caractéristiques présentées sous forme d'une grille (section 2.1.14).

2.1.1 Nombre de copies concernées par une lecture et une écriture

Chaque protocole de duplication a ses propres contraintes sur le nombre de copies à consulter avant de pouvoir répondre à une requête externe de lecture ou d'écriture. Par exemple, certains protocoles font une écriture sur toutes les copies avant de valider une requête externe d'écriture. Dans ce cas, ils n'ont besoin de consulter qu'une copie pour répondre à une requête externe de lecture. D'autres protocoles valident une requête externe d'écriture lorsque n/2+1 copies sont mises à jour (n est le nombre total de copies). Lors d'une requête externe de lecture, il est alors nécessaire de consulter n/2 copies pour pouvoir répondre au demandeur. Les protocoles implantant ces deux approches ont pour objectif d'assurer une cohérence forte entre les copies. Des protocoles relâchant la cohérence peuvent écrire (respectivement lire) sur une copie lors d'une requête externe d'écriture (respectivement de lecture). La mise à jour des différentes copies se fait de manière asynchrone par rapport aux requêtes externes. Ainsi, suivant le protocole celui-ci peut consulter une, plusieurs ou toutes les copies selon le type de la requête externe. De plus le protocole n'interroge pas forcément des copies au hasard, mais par exemple celles qui ont le bon numéro de version (protocoles à base de quorum).

2.1.2 Droits d'accés aux copies

La détermination des copies pouvant être modifiées par des requêtes externes et celles pouvant être uniquement modifiées par le protocole de duplication est un point essentiel. Deux approches extrêmes existent [GHOS96] : l'approche maître-esclaves ou primaire-secondaire (*master-slave* ou *primary-secondary*), et l'approche copies identiques (*update anywhere* ou *peer to peer*). L'approche maître-esclaves se définie comme suit :

Définition 2.3 : *Droits de mises à jour maître-esclave*

Chaque objet dupliqué possède une copie dite maîtresse, les autres étant des copies esclaves. Une requête externe de mise à jour (une écriture) ne peut être faite que sur le maître, celui-ci diffusant ensuite les modifications aux copies esclaves.

Dans la figure 2.2(a) la copie maîtresse C1 reçoit les requêtes d'écriture, puis les diffuse aux copies esclaves C2, C3 et C4. Les copies esclaves ne peuvent recevoir que des requêtes externes de lecture. Cette approche simplifie le contrôle de la concurrence mais elle introduit des encombrements sur la copie maîtresse et fragilise le système. Des variantes où la copie maîtresse peut transférer son rôle à une autre copie au cours du temps permettent d'améliorer les performances (élimination du goulot d'étranglement) ou la tolérance aux fautes (dans le cas où la copie maîtresse est suspectée de mal se comporter). La politique de transfert de la maîtrise peut être décidée de manière déterministe ou indéterministe par des protocoles de vote par exemple. D'autres variantes proposent qu'il y ait plusieurs copies maîtresses en même temps.

L'approche à copies identiques se définie comme suit :

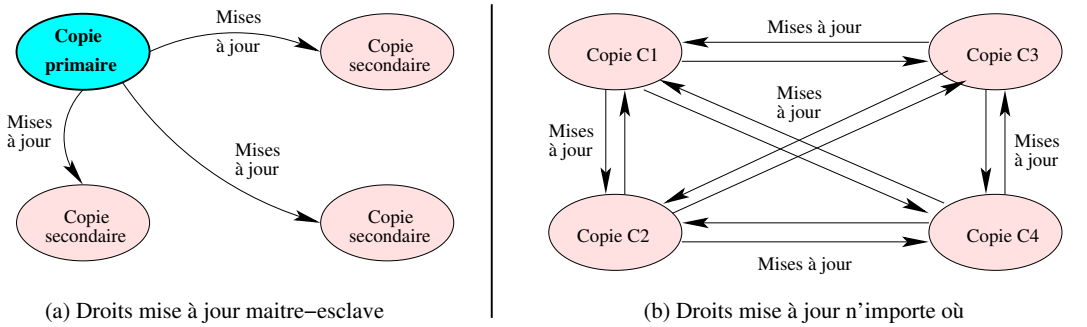
Définition 2.4 : *Droits de mises à jour à copies identiques*

Chaque copie d'un objet dupliqué peut traiter des requêtes externes d'écriture.

Avec cette approche, toutes les copies acceptent les requêtes externes d'écriture et de lecture. Elles ont toutes un comportement de copie maîtresse. Dans la figure 2.2(b) les copies C1, C2, C3 et C4 traitent les requêtes externes d'écriture et diffusent ensuite aux autres copies les mises à jour. Les conflits pouvant survenir suite à deux écritures simultanées sur deux copies différentes est le principal point négatif de cette approche.

2.1.3 Moment de la synchronisation

Les mises à jour des différentes copies peuvent se faire simultanément sur toutes les copies ou d'abord sur une et ensuite sur les autres. [GN95] propose une classification des différents moments de déclenchement de la synchronisation sous le terme "condition de cohérence". Sept



(a) Droits mise à jour maitre–esclave

(b) Droits mise à jour n'importe où

FIG. 2.2 – Approche maître-esclaves versus copies identiques

classes de condition de cohérence sont définies :

Conditions sur le délai. Ces conditions portent sur le temps. Elles expriment la durée maximale que le protocole peut attendre avant la propagation d'une mise à jour. Par exemple, pour un délai maximum de 60 secondes, toutes les mises à jour d'une copie x doivent être propagées vers la copie x' avant l'expiration de celui-ci. Avec cette approche, seule la dernière valeur de x peut être propagée vers x'.

Conditions sur la périodicité. Ces conditions portent également sur le temps. Elles spécifient qu'une copie x' de x doit être mise à jour avec la dernière valeur de x toutes les m unités de temps, que x ait été modifiée ou non. L'avantage de cette approche sur la première est que les pertes de messages de mise à jour dues au réseau ou aux pannes de site sont écartées.

Conditions sur le moment. Ces conditions, introduites par [WQ87, WQ90], sont un cas particulier des conditions de périodicité. Elles spécifient qu'une copie x' de x doit être mise à jour à un moment donné avec la dernière valeur de x, par exemple tous les jours à 8h.

Conditions sur la version. Ces conditions spécifient le nombre de modifications pouvant avoir lieu sur la copie x avant que la copie x' soit mise à jour.

Conditions numériques. Si la valeur d'un objet est numérique, ces conditions permettent de borner la déviation entre les valeurs des différentes copies. Il est possible de considérer des différences absolues, relatives ou exprimées en pourcentage. Pour vérifier ces conditions il faut connaître la valeur des deux copies.

Conditions sur les objets. Ces conditions portent sur la structure des objets. Trois types de conditions sont définies : x' doit être mise à jour avec la dernière valeur de x lorsque (a) au moins i sous objets de la copie x ont été modifiés, (b) au moins q pourcent des sous objets de x ont été modifiés ou (c) le sous objet a de x a été modifié, depuis la dernière mise à jour de x'. Ces conditions sont particulièrement adaptées aux systèmes à objets, mais aussi à d'autres modèles, par exemple les bases de données relationnelles. En effet ces relations

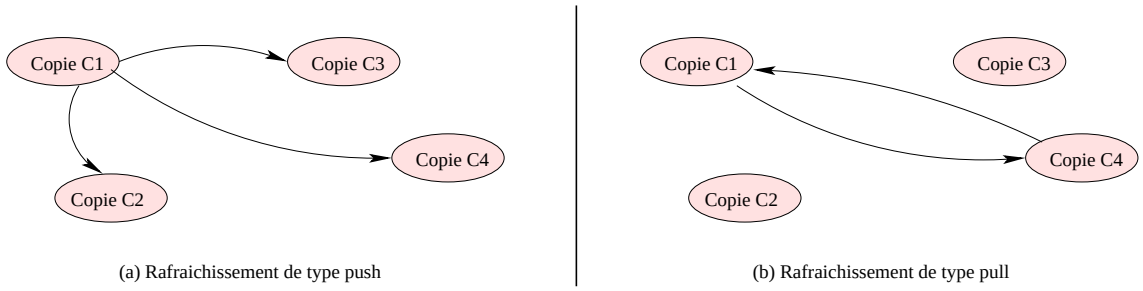


FIG. 2.3 – Rafraîchissement de type push et pull

objets sous objets existent aussi entre les relations et les attributs, entre les relations et les n-uplets ou encore entre les n-uplets et les attributs. Par exemple, pour une copie x' qui est une relation en lecture seule servant pour faire des calculs statistiques, il est raisonnable de ne mettre à jour x' que si plus de q% des n-uplets de x ont changé.

Conditions événements. Finalement, le déclenchement des mises à jour des copies peuvent être dirigées par des événements. Cette classe de conditions est la plus générale. Un modèle d'événements assez riche permettrait d'exprimer les conditions précédentes.

2.1.4 Initiative de la mise à jour

Les copies possédant l'information permettant de faire une mise à jour sont appelées copies sources, alors que les copies sur lesquelles doivent être propagées les modifications sont appelées copies cibles. Deux approches pour le rafraîchissement des copies sont possibles. Soit la copie source est l'initiatrice des propagations (figure 2.3 (a)), soit les copies cibles demandent les mises à jour à une (aux) copie(s) source(s) (figure 2.3 (b)).

Définition 2.5 : *Rafraîchissement de type push*
Propagation des mises à jour à l'initiative de la copie source vers les copies cibles.

Définition 2.6 : *Rafraîchissement de type pull*
Propagation des mises à jour à l'initiative de la copie cible.

Avec la première approche, la copie source diffuse à toutes les copies cibles les mises à jour. Dans certains cas, cela peut poser des problèmes de passage à l'échelle. Des messages inutiles de synchronisation peuvent être envoyés à des copies qui ne seront pas consultées. L'approche de type pull permet de réduire la charge réseau en ne propageant que les dernières

modifications ou en les regroupant. Par contre, les copies cibles ne savent pas si une mise à jour est nécessaire. De plus, si elles interrogent trop souvent la copie source cette approche peut s'avérer inintéressante.

2.1.5 Nature des mises à jour

La nature des mises à jour désigne le type d'information envoyé aux différentes copies lors de la synchronisation. On distingue deux approches : les protocoles à diffusion des écritures et les protocoles à invalidation [EK89].

Définition 2.7 : *Protocole à diffusion des écritures*

Un protocole à diffusion des écritures envoie les modifications faites sur une copie aux autres copies.

Définition 2.8 : *Protocole à invalidation*

Suite à la modification d'une copie, un protocole à invalidation envoie aux autres copies une notification les informant qu'elles sont invalides et ne doivent plus être accédées. Avant de pouvoir de nouveau être utilisée, les copies invalidées doivent être resynchronisées.

Pour les protocoles à diffusion des écritures, deux cas sont envisageables : soit il y a transfert des états (ou d'un delta), soit la procédure exécutée sur la source peut être propagée vers le site cible pour y être exécutée (duplication des opérations). Dans certains cas, la deuxième approche évite le transfert d'importants volumes de données.

Les protocoles à invalidation permettent de limiter la taille des messages échangés entre les différentes copies, alors que les protocoles à diffusion des écritures augmentent le trafic et les risques d'engorgement sur le médium de communication. De plus, le travail que doit fournir chaque copie est plus important, celles-ci devant installer les mises à jour. Cependant, celles-ci sont moins en retard que dans le cas des protocoles à invalidation.

2.1.6 Topographie de la synchronisation

Bien souvent, la copie devant propager une mise à jour le fait par diffusion à toutes les copies (figure 2.4 (a)). D'autres protocoles propagent les mises à jour de copie en copie (figure 2.4 (b)) ou vers un ensemble de copies où chacune d'elles les propagent à leur tour vers un autre ensemble (figure 2.4 (c)). Certains protocoles construisent des chemins particuliers entre les copies que doivent suivre les mises à jour (figure 2.4 (d)). On peut ainsi imaginer différentes topographies de propagation des mises à jour entre les copies.

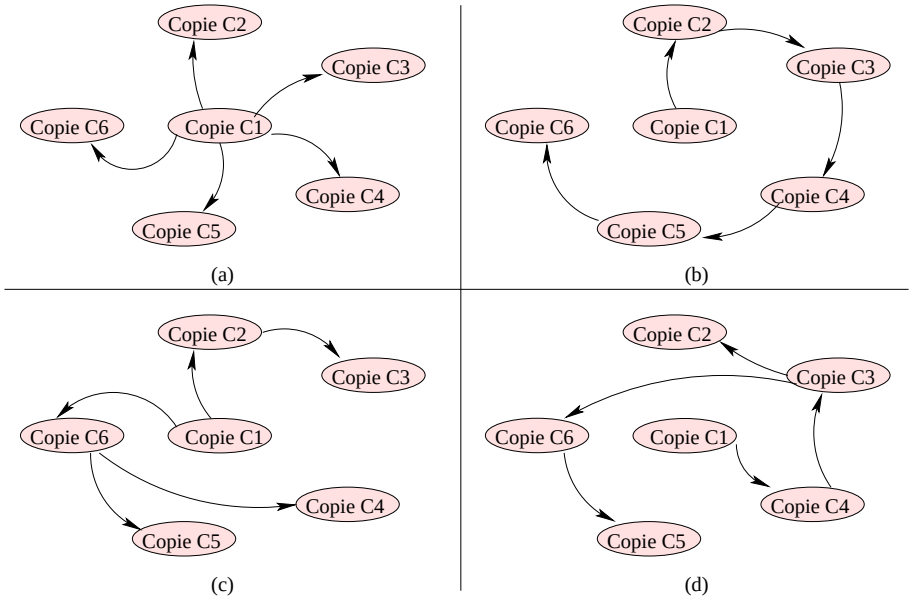


FIG. 2.4 – Différentes topographies de propagation des mises à jour

Ces différents protocoles se justifient quand des copies (ou des groupes de copies) doivent être mises à jour avant d'autres ou qu'il est avantageux de s'appuyer sur la topographie du réseau.

2.1.7 Capture des mises à jour

Le mécanisme utilisé pour détecter et sélectionner les changements sur une copie afin de les propager aux autres copies est appelé la capture. Il peut s'implanter de diverses façons. Une manière de faire consiste simplement à consulter la copie afin de connaître son dernier état.

Une deuxième façon de faire consiste à enregistrer les modifications sur un support particulier : un journal (*log sniffing*) ou une copie ombre (*shadow*). Dans le premier cas on consigne dans un journal les requêtes modifiant une copie ou la nouvelle valeur de la copie. L'inconvénient est qu'il faut modifier la gestion du journal pour qu'il tienne compte de la duplication. Une copie ombre est une "copie de la copie". Elle permet de garder une trace de la valeur de la copie avant les modifications. Cette approche est surtout utilisée pour faire de la tolérance aux fautes (pour pouvoir revenir en arrière si un problème survient), mais elle peut aussi être utilisée pour détecter et sélectionner les dernières modifications.

Une troisième approche consiste à déclencher un mécanisme particulier : un trigger (*trigger-based*) ou une API (*API-based*). Avec la première technique, la modification d'une donnée

dupliquée déclenche un trigger. C'est un mécanisme général et extensible. Avec la deuxième technique, un appel explicite à une API particulière permet d'informer de l'exécution d'une modification.

2.1.8 Gestion des conflits

Dans certains protocoles deux copies peuvent être modifiées de manière concurrente. Lorsque le protocole désire synchroniser les copies (pas forcément immédiatement), il se trouve face à un conflit. Dans ce cas de figure, il doit être à même de détecter le conflit et de réconcilier les différentes copies afin de ne pas perdre de modification ou de compromettre sa sémantique.

Définition 2.9 *: Détection d'un conflit*

Lors du fonctionnement normal du protocole de duplication, la détection des conflits est l'action de détecter à posteriori des accès conflictuels sur différentes copies

Définition 2.10 *: Réconciliation d'un conflit*

Suite à la détection d'un conflit, la réconciliation est l'action de supprimer un conflit afin de rendre sa sémantique au protocole de duplication.

On distingue deux approches pour la détection et la réconciliation : l'approche syntaxique et l'approche sémantique [DGMS85].

2.1.9 Interactions avec le protocole de communication

Il existe divers protocoles de communication offrant différentes garanties (ordre, fiabilité, etc.) dont les protocoles de duplication peuvent tirer parti. Si le protocole de communication ne fournit pas une garantie suffisante, le protocole de duplication doit en tenir compte.

Le passage de message point à point. Le passage de message repose sur deux primitives : envoyer (*send*) et recevoir (*receive*), sur lesquelles il est possible d'avoir certaines garanties de fiabilité ou d'ordre. La fiabilité est un critère décrivant la façon dont le système réagit en cas de défaillance d'un canal de communication ou d'un processeur (tampons pleins, perte de messages, ...). Dans le cas non fiable, le système ne fournit aucune garantie sur le bon acheminement du message. Dans le cas fiable, le système garantit la livraison du message (s'il y a perte de message, il signal une erreur). Il existe plusieurs types d'*ordre de livraison* des messages : sans ordre, ordre FIFO (First In, First Out) et l'ordre causal. Sans ordre, un message m1 émis avant un message m2 par un processus P1 peut être reçu par un processus

P2 après le message m2. L'*ordre FIFO* conserve l'ordre d'émission depuis un même émetteur. Cependant, il n'y a aucune garantie sur l'ordre des messages transitant par un troisième processus. L'*ordre causal* permet d'ordonner la livraison des messages en fonction d'une relation de causalité. Informellement, cela signifie que si un évènement E' est causé ou influencé par un évènement E, alors tout le système doit observer l'évènement E avant l'évènement E'.

Passage de messages de un vers plusieurs (communications de groupe). Un processus peut également envoyer un message à un groupe constitué de n destinataires (les membres). Une primitive réalisant l'envoi d'un message à un groupe est appelée multicast. On peut définir de manière informelle une diffusion fiable comme suit : les destinataires corrects reçoivent le même ensemble de messages (*propriété d'accord*), cet ensemble de messages contient tous les messages provenant d'émetteurs corrects (*propriété de validité*) et enfin il n'existe pas dans cet ensemble des messages non émis (*propriété d'intégrité*). Des variantes *uniformes* des propriétés d'accord et d'intégrité définissent le comportement des processus quand certains messages sont délivrés à des processus fautifs [HT93]. La fiabilité étant difficilement réalisable dans le cas de la communication de groupe, beaucoup de systèmes fournissent des primitives de communication non fiables. L'envoi de messages vers un groupe de processus ajoute une dimension supplémentaire à l'ordre de livraison des messages : l'ordre de livraison des messages sur un membre du groupe par rapport à l'ordre de livraison sur chaque membre. L'*ordre total* est un ordre où tous les destinataires reçoivent les messages dans le même ordre. C'est une dimension supplémentaire, car un ordre total ne garantit pas forcément l'ordre FIFO ou l'ordre causal. Ainsi, il existe des diffusions FIFO totales et des diffusions causales totales [HT93]. Une diffusion fiable respectant un ordre total est appelée une diffusion atomique. La sémantique de *livraison* détermine quand la diffusion d'un message est considérée comme réussie pour l'émetteur. Il en existe principalement trois : quand k destinataires l'ont reçue (*k-delivery*), quand une majorité des membres du groupe l'ont reçue (*quorums*) ou quand tous les membres non fautifs ont reçu le message (dans le cas contraire, aucun d'entre eux ne doit l'avoir reçu[2]).

Appel de procédures à distance. Dans un système réparti asynchrone il n'est pas possible de différencier une machine lente d'une défaillance. Si l'appelant réémet pensant à une défaillance, alors que la machine est seulement lente, la procédure sera exécuté deux fois. [Nel81] envisage trois sémantiques pour les RPC : au moins une fois, au plus une fois ou exactement une fois. Tout comme il existe des communications de groupe, il existe des appels de procédures à distance qui provoquent n exécutions sur différentes machines. L'appelant est soit bloqué jusqu'à ce que les n réponses lui parviennent (MultiRPC [SS90]), soit il a la possibilité d'exécuter du code entre chaque réponse (PARPC [MBBP89]).

L'invocation de méthodes à distance (Remote Method Invocation, RMI) est un appel de procédures à distance dans un contexte objet. Les mêmes caractéristiques, remarques, avantages et inconvénients que pour les appels de procédures à distance s'appliquent aux invocations de méthodes à distance.

[2]C'est donc une diffusion fiable

2.1.10 Gestion de la concurrence

Un objet dupliqué peut être accédé simultanément par l'intermédiaire de différentes copies. Suivant le protocole de duplication, différentes situations sont possibles. Un objet dupliqué peut être accédé par :

- Plusieurs lecteurs ou un écrivain à un moment donné.
- Plusieurs lecteurs ou plusieurs écrivains en même temps sur des copies différentes.
- Plusieurs lecteurs et un écrivain en même temps sur une copie différente.
- Plusieurs lecteurs et plusieurs écrivains en même temps sur des copies différentes.

2.1.11 Gestion des fautes

Certains protocoles de duplication sont à même de supporter les fautes survenant sur les copies et d'autres non. Quatre types de fautes sur les copies peuvent survenir :

- les fautes par arrêt : la copie s'arrête prématurément de façon définitive.
- les partitions réseau : les copies constituant l'objet dupliqué se retrouvent dans deux sous groupes ne pouvant plus communiquer entre eux.
- les fautes par valeur : une valeur n'appartient plus au domaine des valeurs attendues.
- les fautes byzantines : la copie a un comportement imprévisible.

Généralement la tolérance aux fautes dans un protocole de duplication comporte deux aspects : la détection des fautes et la récupération. Bien souvent, pour les fautes par arrêt ou les partitions réseau, la détection se fait quand une copie ne répond plus. Pour les fautes par valeur ou byzantine, elle se fait par des algorithmes de vote. Dans ce cas, il est nécessaire de dupliquer les traitements. Une fois la faute détectée, la copie incriminée est supprimée du groupe des copies géré par le protocole jusqu'à ce qu'elle retrouve un comportement normal. Des traitements peuvent alors être nécessaires afin de retrouver une configuration normale pour le protocole (élection d'un nouveau maître, ré-exécution de traitements sauvegardés et non exécuté sur toutes les copies, etc.). Avant de réintégrer une copie ayant à nouveau un comportement normal au groupe, il est nécessaire qu'elle rattrape son retard sur les autres copies.

Rappelons que la duplication permet également de rendre tolérant aux fautes un système. Dans ce cas, elle est un mécanisme pour assurer la tolérance aux fautes.

2.1.12 Notion de copie

Suivant les protocoles, la notion de copie peut être différente. Certains protocoles de duplication ne savent pas ce qu'est une copie. Ils s'en remettent à l'application pour créer et détruire les copies. De plus, un objet peut référencer d'autres objets. Suivant les cas, il est possible qu'il soit nécessaire de dupliquer ces références lors de la création d'une copie.

Différents modèles de données sont également à considérer : orienté objet, relationnel, fichier, autre ou ouvert. Le terme "ouvert" signifie que le protocole supporte différents modèles. Il est également nécessaire de décider combien de copies dans le système sont nécessaires, où les mettre, à quels moments en créer de nouvelles. Cette allocation des copies peut être dynamique ou statique.

2.1.13 Transparence à la duplication

Le protocole de duplication peut offrir plus ou moins de transparence à l'application. Il y a transparence à la duplication si l'application n'a pas conscience du fait que les objets sont dupliqués. L'application manipule les objets dupliqués comme des objets classiques ; on parle d'objets logiques. S'il n'y a pas transparence à la duplication, l'application manipule explicitement chaque copie. On parle alors d'objets physiques.

Définition 2.11 *: Objet logique*

Un objet logique est une abstraction représentant l'ensemble des copies d'un même objet dupliqué. Cette abstraction permet de référencer l'ensemble des copies sous une seule désignation.

2.1.14 Grille de comparaison des protocoles de duplication

Le tableau 2.5 résume les différentes caractéristiques des protocoles de duplication présentées dans cette section.

Dans les sections suivantes, nous présentons les protocoles de duplication utilisés dans les SCG, les SGBDR et les MPR. La grille proposée nous permet de comparer les protocoles de chacun de ces domaines. De plus, nous montrons que chaque domaine introduit ses propres spécificités. Nous n'abordons pas si ces protocoles peuvent créer des copies, ce qu'ils font des références, le modèle de données et si la création des copies est dynamique ou statique. Ces points ne nous intéressent pas directement dans la suite de ce document.

2.2 Duplication à l'aide de systèmes de communication de groupe

Cette section présente comment mettre en œuvre la duplication à l'aide d'un système de communication de groupe. Un système de communication de groupe propose des primitives de communication permettant de diffuser des messages vers des groupes avec différentes garanties de fiabilité et d'ordre. Les protocoles de duplication tirent parti des propriétés offertes par ces systèmes en regroupant les copies d'un même objet dans un groupe.

Nombre copie pour E ou L	Une		0<n<toutes				Toutes	
Droits de mise à jour	maitre–esclave				n'importe quelle copie			
	Un		plusieurs					
	dynamique	statique	dynamique	statique				
Moment de la synchronisation	Immédiate	Différé						
		selon délai	selon périodicité	selon moment	selon version	selon numérique	selon objets	selon événements
Initiative de la mise à jour	Push				Pull			
Nature des mises à jour	Diffusion				Invalidation			
	Valeur		Opération					
Topologie	Etoile		Copie en Copie		Groupe en Groupe		Chemin particulier	
Détection des mises à jour	Utilisation d'un support			Déclenchement d'un mecanisme			Lecture de la copie	
	Journal	Copie ombre		Trigger		API		
Conflits	Oui				Non			
couche com.	Oui				Non			
Concurrence	n L ou 1 E		n L ou n E		n L et 1 E		n L et n E	
Fautes	Oui				Non			
Copie	Création des copies		Références		Modèle de données		Allocation des copies	
Transparence	Oui				Non			

FIG. 2.5 – Caractéristiques des protocoles de duplication

Nous commençons (section 2.2.1) par décrire les quatre principaux protocoles de duplication utilisés dans ce contexte. Ensuite (section 2.2.2), nous présentons quelques systèmes, notamment certains visant à relâcher la cohérence entre les copies. En conclusion (section 2.2.3), nous mettons en valeur pour chaque protocole présenté ses caractéristiques telles que données en section 2.1.

2.2.1 Quatre protocoles majeurs

Les quatre principaux protocoles de duplication utilisés sur des systèmes de communication de groupe sont la duplication passive (section 2.2.1.1), la duplication active (section 2.2.1.2), la duplication semi-active (section 2.2.1.3) et la duplication coordinateur cohorte (section 2.2.1.4). Ils garantissent une cohérence forte entre les différentes copies d'un objet dupliqué. Bien souvent, ils servent à assurer la tolérance aux fautes.

2.2.1.1 Duplication passive

Dans un protocole de duplication passive (*passive replication* ou *primary/backups replication*) [PV91, BMST93] une seule copie, appelée copie primaire (*primary copy*), reçoit la requête d'un client et l'exécute. Les autres copies, nommées copies secondaires (*secondary copies*) ou copies de sauvegardes (*backups*), ne sont là que pour prendre le relais en cas de défaillance de la copie primaire. Afin d'assurer la cohérence des copies secondaires, la copie primaire leur diffuse régulièrement son nouvel état (un point de reprise). Cette méthode assure que toutes les copies ont la même valeur même si les traitements sont non déterministes car il y a diffusion de l'état de la copie primaire vers les copies secondaires et non exécution des requêtes[3].

La diffusion de l'état doit respecter un certain ordre. Dans le cas d'un mode de communication synchrone, le protocole de communication doit assurer que le canal entre le client et la copie primaire est fiable et que les messages sont délivrés atomiquement. Dans un mode de communication asynchrone, les messages doivent au moins respecter l'ordre causal.

La duplication passive supporte les fautes par arrêt silencieux et les coupures réseaux, mais pas les fautes byzantines et les fautes par valeurs. La copie primaire étant la seule à exécuter les requêtes, il ne peut y avoir de vote sur la valeur des réponses. Quand une copie secondaire défaille aucun traitement particulier (à part la détection et le fait de l'enlever du groupe) n'est nécessaire. Son seul effet est de diminuer le taux de duplication du composant. Par contre, quand une copie primaire défaille, une élection a lieu pour désigner une nouvelle copie primaire parmi toutes les copies secondaires. Si une copie secondaire manque une mise à jour, cette copie est périmée, elle ne peut devenir une copie primaire avant qu'elle ne récupère le dernier état. Si la copie primaire défaille pendant une invocation d'un client, alors celui-ci n'obtient aucune réponse à sa requête et il doit la ré-émettre en l'adressant à la nouvelle copie primaire. Si la défaillance de la copie primaire est détectée avant la réception du point de reprise par les copies secondaires, tout le traitement effectué par celle-ci est perdu. Par contre, si la défaillance de la copie primaire est détectée après, la nouvelle copie primaire construit la réponse à partir du point de reprise et envoie la réponse au client.

La figure 2.6 donne un exemple de protocole de duplication passive assurant une cohérence forte entre les copies. Il existe trois copies de S : S_1, S_2 et S_3. Un client C envoie la requête Q uniquement à la copie primaire S_1. Celle-ci traite la requête, construit un point de reprise (*checkpoint*) et l'envoie, à l'aide d'un multicast fiable assurant l'ordre FIFO, aux copies secondaires S_2 et S_3. Le point de reprise contient à la fois la réponse R et le nouvel état de la copie primaire. Ensuite, la copie primaire envoie la réponse R à C.

Variantes : La construction d'un point de reprise est faite systématiquement après le traitement d'une requête, afin d'assurer que l'état de la copie primaire soit déjà sauvegardé sur les copies secondaires lorsque la réponse est envoyée au client. La copie primaire ne peut répondre tant que la dernière, et donc la plus lente, des copies secondaires n'a pas enregistré le nouvel état. Afin d'optimiser cette approche, certains protocoles tiennent compte de la

[3]Un traitement est déterministe si pour les mêmes données en entrée, il donne toujours le même résultat.

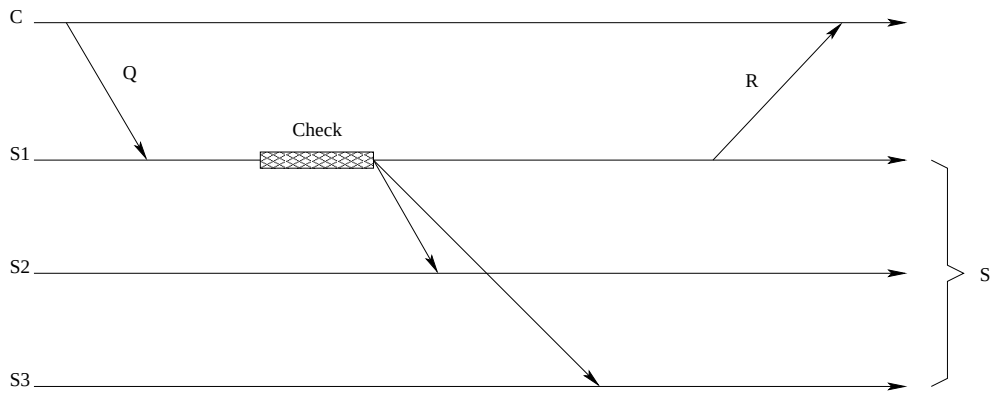


FIG. 2.6 – Exemple de duplication passive

sémantique des requêtes en ne construisant pas de point de reprise si celles-ci ne modifient pas l'état de l'objet. D'autres variantes construisent les points de reprise périodiquement (toutes les n requêtes). Si cette approche est moins coûteuse, elle est aussi moins sûre, car les copies secondaires sont toujours en retard de plusieurs requêtes (n-1) sur la copie primaire.

La duplication passive ne permet pas généralement d'améliorer les performances. La copie primaire est un goulot d'étranglement, puisque toutes les requêtes lui parviennent. Les copies secondaires ne sont que des sauvegardes prêtes à prendre le relais en cas de défaillance. Ainsi, certains protocoles proposent de fournir plusieurs copies primaires pour un même objet : chaque requête peut avoir une copie primaire différente ou chaque requête manipulant des données différentes de l'objet utilise une copie primaire différente. Les données manipulées par ces différentes copies primaires doivent être indépendantes les unes des autres ou des mécanismes d'exclusion mutuelles doivent exister.

2.2.1.2 Duplication active

Dans un protocole de duplication active (*active replication* ou *state machine approach*) [Sch90, PV91] toutes les copies jouent le même rôle. Elles reçoivent toutes la même séquence totalement ordonnée de requêtes de la part des clients, les exécutent de manière déterministe et renvoient la même séquence totalement ordonnée de réponses. La condition pour que toutes les copies reçoivent et exécutent toutes les requêtes dans le même ordre peut être décomposée en deux conditions distinctes :

- Toutes les copies non fautives reçoivent les mêmes requêtes [4] (C1) et
- Toutes les copies non fautives exécutent les requêtes dans le même ordre relatif [5] (C2).

[4] Propriété d'accord
[5] Propriété d'ordre.

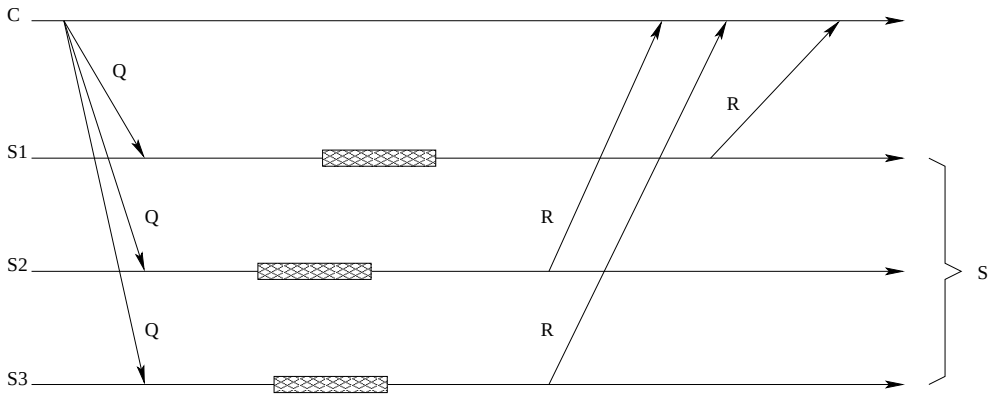


FIG. 2.7 – Exemple de duplication active

L'intérêt majeur de ce protocole est qu'il n'y a pas de point de reprise coûteux. Cependant, l'exécution des requêtes doit être déterministe, ou alors des mécanismes supplémentaires doivent être utilisés. De plus, ce protocole nécessite des mécanismes de collecte des réponses des différentes copies.

Une façon de garantir les conditions C1 et C2 est d'utiliser un mécanisme de diffusion atomique. Mais ce type de protocole est coûteux. Cependant, une diffusion fiable peut garantir la condition C1, alors que la condition C2 peut être fournie par des mécanismes de transaction, laissant plus de souplesse pour relâcher les conditions. Si la communication est asynchrone entre les clients et les copies, un protocole garantissant l'ordre causal doit être ajouté.

La duplication active supporte plus de types de fautes que la duplication passive. Dans le cas des fautes par valeur et des fautes byzantines, des mécanismes de vote parmi les différents résultats sont nécessaires. Quand une copie défaille, aucun mécanisme n'est à mettre en œuvre. La tolérance aux fautes est réalisée par masquage d'erreur. La défaillance d'une copie est masquée par le comportement des copies non défaillantes. Il est nécessaire de disposer de mécanismes de reprise après panne pour qu'une copie puisse rattraper son retard après une panne [BJRA85].

La figure 2.7 donne un exemple de replication active assurant une cohérence forte entre les copies. Les S_i sont des copies du composant dupliqué S. Lorsque C invoque S, il envoie une requête Q à tous les S_i à l'aide d'un multicast fiable assurant l'ordre total. Chaque S_i traite la requête et envoie une réponse R à C.

Variantes : Le client peut attendre la première réponse, et donc continuer son exécution en se synchronisant sur la plus rapide des copies afin d'améliorer les performances. De plus, en prenant en compte la sémantique des requêtes, on peut relâcher la condition C1. Par exemple, les requêtes en lecture seule ne sont pas diffusées à toutes les copies.

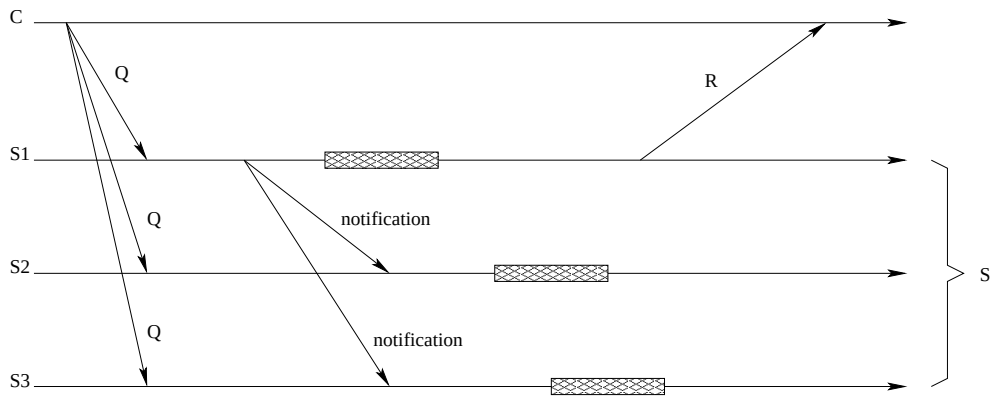


FIG. 2.8 – Exemple de duplication semi-active

2.2.1.3 Duplication semi-active

Un protocole de duplication semi-active (*semi-active replication* ou *leader/followers replication*) [PV91] est un protocole de duplication active dans le sens où chaque copie exécute la requête. Par contre, toutes les sources d'indéterminisme sont résolues par le choix d'une copie primaire qui diffuse aux autres copies ses choix. De plus, la copie primaire est la seule à renvoyer les résultats aux clients. La copie primaire est appelée leader (*leader*) et les copies secondaires sont appelées suiveurs (*followers*). Le leader traite une requête dès qu'il la reçoit. Par contre, un suiveur doit attendre une notification du leader pour pouvoir traiter une requête. Lorsqu'il y a des sources d'indéterminisme, le leader envoie ses choix aux suiveurs. Ainsi, contrairement à un protocole de duplication active, les protocoles d'accord sont évités.

La tolérance aux fautes est réalisée par détection et compensation d'erreur, comme dans le cas de la duplication passive. Cependant, comme toutes les copies reçoivent la requête, le client n'a pas besoin de ré-émettre sa requête lorsque le leader défaille. Le nouveau leader envoie automatiquement la réponse au client, qui risque de recevoir plusieurs fois la même réponse. Comme dans le cas de la duplication passive, deux situations peuvent se présenter suivant que la défaillance du leader est détectée par les suiveurs avant ou après la réception de la notification. Si la défaillance du leader est détectée avant la réception de la notification, alors le nouveau leader envoie une notification concernant la première requête présente dans sa file d'entrée et la traite normalement. Cette requête peut aussi bien être la requête du client qu'une autre requête reçue précédemment pour laquelle le nouveau leader n'a pas reçu de notification. Si la défaillance du leader est détectée après la réception de la notification, le nouveau leader traite la requête correspondante sans envoyer de notification et envoie la réponse au client. Celui-ci a pu déjà recevoir cette réponse du leader défaillant, si celui-ci a défailli après avoir envoyé la réponse.

La figure 2.8 illustre ce mode de fonctionnement. Le client C envoie la requête Q à tous les copies S_i. Le leader S_1 envoie une notification aux suiveurs et commence le traitement de

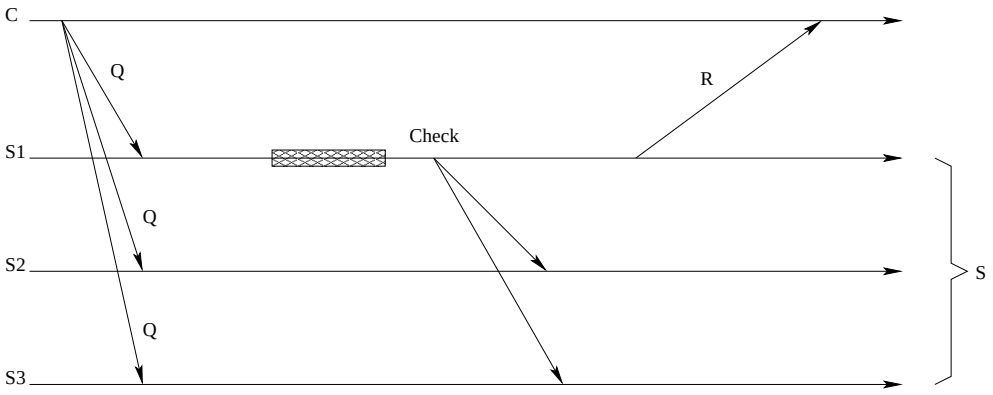


FIG. 2.9 – Principe de la duplication coordinateur/cohortes

Q. Les suiveurs S_2 et S_3 ne commencent à traiter Q qu'après avoir reçu la notification du leader. Dès le traitement terminé, S_1 envoie la réponse R au client C.

2.2.1.4 Duplication coordinateur/cohortes

Comme le protocole de duplication semi-active, le protocole coordinateur/cohortes (*coordinator cohort replication*) [Bir85] est un protocole hybride entre la duplication active et la duplication passive. Ce protocole distingue une copie primaire appelée coordinateur (*coordinator*) et des copies secondaires appelées cohortes (*cohort*). Toutes les copies reçoivent la requête, mais le coordinateur est le seul à la traiter. Le coordinateur envoie des points de reprise systématiques aux cohortes. La duplication coordinateur/cohortes est une duplication passive où les requêtes sont envoyées à toutes les copies afin de ne pas en perdre. Contrairement à la duplication semi-active, seul le coordinateur exécute les requêtes.

La tolérance aux fautes est réalisée par détection et compensation d'erreur. Comme pour le protocole de duplication active et le protocole de duplication semi-active, le client n'a pas besoin de ré-émettre sa requête lorsque le coordinateur défaille. Dans ce cas, le nouveau coordinateur envoie automatiquement la réponse au client.

La figure 2.9 illustre ce mode de fonctionnement. Le client C envoie la requête Q à toutes les copies S_i. Le coordinateur S_1 traite la requête et procède exactement comme dans le cas de la duplication passive.

2.2.2 Présentation de différents systèmes

Cette section présente les particularités de quelques travaux : Isis (section 2.2.2.1) pour la répartition de charge, Psync (section 2.2.2.2) pour la gestion des opérations commutatives,

Grapevine (section 2.2.2.3), Lazy replication (section 2.2.2.4), pour le relâchement de la cohérence entre les copies et CODA pour (section 2.2.2.5) l'approche utilisé dans les systèmes de fichiers.

2.2.2.1 Isis

Le système Isis [BJRA85, Bir93, BR994, RB94] fournit une infrastructure et une boîte à outils pour construire des systèmes répartis tolérant aux fautes. Il fournit une abstraction, les groupes de processus, permettant d'implanter des protocoles de duplication. Il est possible de mettre en œuvre, avec plus ou moins de difficulté la duplication active, passive, semi-active et coordinateur cohortes. Un groupe contient les copies d'un objet. Ces groupes peuvent être créés et détruits dynamiquement et les membres peuvent entrer et sortir à n'importe quel moment. Les messages sont envoyés à ces groupes, par des *multicast* causales ou atomiques, et Isis garantit que tous les membres vont les recevoir. Il propose également différente collectes de résultat : zéro, un, un quorum ou toutes les réponses sont collectées.

Isis propose une variante du protocole de duplication coordinateur-cohortes supportant le partage de charge. A chaque requête, un nouveau coordinateur est choisi permettant ainsi aux requêtes de s'exécuter de manière concurrente (différentes copies les exécutent). Les requêtes concurrentes ne doivent pas modifier la même partie de la copie ou alors des mécanismes de gestion de la concurrence sont nécessaires. Pour chaque requête l'ensemble des copies est trié à l'aide d'un algorithme déterministe, pouvant prendre en compte la provenance des requêtes et des informations sur la charge. Grâce à cet algorithme, le même ensemble trié de copies est obtenu, et cela sans utiliser de protocoles d'accord. La copie de plus haut rang est désignée comme coordinateur pour cette requête. De plus, toutes les copies ont connaissance de toutes les autres copies non fautives. Un suspecteur de fautes surveille tous les membres du groupe, notifiant de la défaillance probable d'une copie par l'envoi d'un message GBCAST (diffusion respectant un ordre total sur tous les groupes) au groupe. Grâce à cela, toutes les copies observent de la même façon toutes les défaillances et réintégrations des autres copies. Avec cette propriété et le déterminisme de l'algorithme de tri, toutes les copies sont d'accord sur le processus coordinateur pour une même requête. Si le coordinateur défaille, la copie suivante dans la liste est désignée comme nouveau coordinateur (aucun message n'est envoyé). Une fois que le coordinateur a fini d'exécuter la requête, il diffuse à l'aide d'un CBCAST (diffusion respectant l'ordre causal) un point de reprise et le résultat de la requête à toutes les cohortes.

2.2.2.2 Psync

Psync [Pet87] est un système de communication fournissant une diffusion fiable. De plus, il permet de tirer parti des messages concurrents (envoyés en même temps relativement au temps logique). Dans [MPS89], Psync est utilisé pour construire un système à objets tolérant aux fautes en s'appuyant sur un protocole de duplication active. Pour améliorer les performances du système, les auteurs définissent des groupes d'opérations commutatives. Les opérations

de ces groupes peuvent s'exécuter dans un ordre quelconque sur les différentes copies. Un algorithme de tri déterministe ordonne totalement toutes les opérations non commutatives entre elles, mais permet aux opérations commutatives de s'exécuter dans un ordre différent sur les différentes copies du système. Par exemple, pour les quatre opérations arithmétiques de base, il n'est pas nécessaire de respecter l'ordre entre l'addition et la soustraction ainsi qu'entre la multiplication et la division. A chaque copie, est associé un gestionnaire qui reçoit les requêtes des clients. Le gestionnaire de la copie primaire, celui qui reçoit la requête, diffuse la requête à toutes les copies, y compris lui-même, en utilisant Psync. De plus, il est chargé de répondre au client.

2.2.2.3 Grapevine

Le système Grapevine [BLMS82] met en œuvre une duplication de serveurs. Chaque serveur fournit des services de courrier, de nommage (de personnes, de machines et de services), d'authentification (de personnes et machines) et de localisation (de services).

Il fut un des premiers systèmes à mettre en œuvre la duplication où le client s'adresse aux serveurs suivant un protocole "lire n'importe quelle copie/écrire n'importe quelle copie" (*read-any/write-any*). Un client peut s'adresser à n'importe quelle copie d'un serveur pour une requête de lecture ou d'écriture et en changer pour une raison ou une autre (défaillance, performance, etc). La cohérence entre les copies est extrêmement faible, occasionnant certains troubles pour les utilisateurs. Par exemple, un mot de passe peut être changé sur une copie d'un serveur, mais sur une autre copie l'ancien mot de passe est toujours utilisé.

2.2.2.4 Lazy replication

Le protocole Lazy Replication [LLS90, LLSG92] propose une duplication active où la cohérence peut être affaiblie afin de fournir, en plus d'une disponibilité accrue, un temps de réponse meilleur. Le système se compose d'un ensemble de serveurs dupliqués, de clients et de canaux de communication FIFO. Le nombre de copies et leur localisation sont fixes.

Un client adresse ses requêtes par un protocole du type "lire n'importe quelle copie/écrite n'importe quelle copie". S'il considère que la réponse tarde à venir, il peut changer de copie, ou même adresser sa requête à plusieurs copies en parallèle. Le système garantit une sémantique "au plus une fois" sur les requêtes. A chaque requête soumise à un serveur, un identificateur unique est retourné au client. Lorsque ce dernier soumet une requête, il envoie au serveur, en plus de sa requête, une liste d'identificateurs. Cette liste lui permet de préciser au serveur quelles opérations doivent avoir eu lieu sur celui-ci avant qu'il exécute la nouvelle requête soumise. Ainsi, le client peut contrôler l'ordre des messages (*client-order*).

Des échanges paresseux de messages (*gossip message* [6]) transportent les requêtes pour mettre à jour l'état des copies sur les différents serveurs. Ces mises à jour entre les différentes

[6] Ragot.

copies sont faites par un protocole de duplication active et grâce à un ordre total des messages (*server-order*). Ces échanges sont dits paresseux car il n'y a pas d'atomicité entre la mise à jour demandée par le client sur un serveur et la mise à jour de tous les serveurs. Ce type de protocole fait partie du groupe des protocoles d'anti-entropie [7] ou épidémique (*anti-entropy*). Occasionnellement une copie échange des messages *gossip* avec une autre copie pour s'informer mutuellement des messages de mise à jour reçus par chacune d'elles. Si une copie se rend compte que l'autre a de nouveaux messages, elle les lui demande. Une fois que toutes les copies se sont échangées leurs mises à jour toutes les copies sont cohérentes. Cette approche a été premièrement proposée par [OD83]. On trouve également des approches utilisant des estampilles temporelles [Gol92, GL93]. Tous ces protocoles maintiennent des journaux sur les messages échangés sur chacune des copies permettant aux copies de retarder la livraison des messages pour mettre en œuvre certains ordres (causal, total).

Dans le protocole Lazy Replication, il est également possible d'ordonnancer les messages des clients et des serveurs (*global-order*). Pour cela, un mécanisme de validation à trois phases est utilisé (*three phase commit*).

Ce protocole présente un inconvénient. En effet, il est possible qu'une copie du serveur défaille après avoir répondu au client, mais avant d'avoir envoyé un message de synchronisation. Si ce client émet de nouvelles opérations sur un autre serveur, il se peut que celles-ci soient bloquées, car attendant des opérations qui sont perdues (dû à l'absence de synchronisation). Ce problème ne peut se résoudre qu'en envoyant les messages de synchronisation plus souvent. Il est aussi possible d'introduire des accusés de réception que la copie du serveur doit attendre avant de répondre au client, mais cela augmente le temps de réponse.

2.2.2.5 CODA

CODA [SKK+90] est un système de fichiers tolérant aux fautes et supportant les opérations déconnectées de la part d'hôtes mobiles. La tolérance aux fautes est assurée par la duplication des serveurs. Un client désirant un fichier s'adresse à un serveur disponible disposant d'une copie. Ce serveur est appelé "serveur préféré". De plus, chaque serveur possédant une copie est contacté pour vérifier que le serveur préféré possède une copie à jour. Si ce n'est pas le cas, un autre serveur est désigné serveur préféré. Une copie est mise en cache sur le client, puis le serveur préféré est mis à jour quand le fichier est fermé. A chaque copie sur les clients sont associées un vecteur de versions permettant de détecter les conflits en écriture. Certains conflits peuvent être automatiquement réconciliés, d'autres demandent l'aide de l'utilisateur. Pour faciliter les opérations en mode déconnecté, l'utilisateur peut indiquer une liste de fichiers ou répertoires prioritaires devant être mis dans son cache. A la reconnexion, les serveurs détenant une copie sont mis à jour s'il n'y a pas de conflit. En cas de conflit, les copies des clients sont temporairement stockées sur un serveur et l'utilisateur peut manuellement résoudre les conflits.

[7] Entropie n. f. : En thermodynamique, fonction définissant l'état de désordre d'un système, croissante lorsque celui-ci évolue vers un autre état de désordre accru.

2.2.3 Conclusion

La figure 2.10 situe les différents protocoles décrits dans cette section selon les caractéristiques présentées dans la section 2.1.

<table>
<tr><td>Nombre copie pour E ou L</td><td colspan="3">Une
DA1R DPCP DSA
DCC I G C LR</td><td colspan="2">0<n<toutes</td><td colspan="2">Toutes
DA DP P</td></tr>
<tr><td rowspan="2">Droits de mise à jour</td><td colspan="2">Un maitre</td><td colspan="2">Plusieurs maitres</td><td colspan="2" rowspan="2">N'importe quelle copie
DA P G C LR</td></tr>
<tr><td>Dynamique
DPMs I</td><td>Statique
DP DSA
DCC</td><td>Dynamique</td><td>Statique</td></tr>
<tr><td rowspan="2">Moment de la synchronisation</td><td colspan="2">Immédiate
DA DP I
DSA DCC P</td><td colspan="6">Différée G C LR</td></tr>
<tr><td>Délai</td><td>Périodicité</td><td>Moment</td><td>Version
DPCP</td><td>Numérique</td><td>Objets</td><td>Evénements</td></tr>
<tr><td>Initiative de la mise à jour</td><td colspan="4">Push
DA DP DSA DCC I P</td><td colspan="4">Pull
LR C G</td></tr>
<tr><td>Nature des mises à jour</td><td colspan="2">Valeur DP DSA
G I C</td><td colspan="3">Opération DA DSA DCC
LR P</td><td colspan="3">Invalidation</td></tr>
<tr><td>Topologie</td><td>Etoile
DA DP DSA
DCC I P</td><td colspan="2">Copie en Copie
G LR C</td><td colspan="2">Groupe en Groupe</td><td colspan="3">Chemin particulier</td></tr>
<tr><td>Détection des mises à jour</td><td>Journal</td><td>Copie ombre</td><td>Trigger</td><td colspan="2">API</td><td colspan="3">Lecture de la copie</td></tr>
<tr><td>Conflits</td><td colspan="4">Oui
G C LR</td><td colspan="4">Non
DA DP DSA DCC I P</td></tr>
<tr><td>couche com.</td><td colspan="4">Oui
DA DP DSA DCC I P</td><td colspan="4">Non
G C LR</td></tr>
<tr><td>Concurrence</td><td colspan="2">n L ou 1 E DA DP
DPMs I DSA DCC</td><td colspan="2">n L ou n E</td><td colspan="2">n L et 1 E</td><td colspan="2">n L et n E
G C LR</td></tr>
<tr><td>Fautes</td><td colspan="4">Oui DA DP DSA DCC I P</td><td colspan="4">Non DPCP G LR C</td></tr>
</table>

DA : duplication active DA1R : duplication active avec synchronisation du client sur la plus rapide des copies
DP : duplication passive DSA : duplication semi active DCC : duplication coordinateur cohortes
DPMs : duplication passive avec différents maitres DPCP : duplication passive avec point de reprise périodique
I : Isis G : Grapevine P : Psync LR : Lazy Replication C : Coda

FIG. 2.10 – Comparaison des protocoles de duplication utilisant les SGC

En résumé, lors d'une requête externe, dans les protocoles de duplication passive et active, toutes les copies sont mises à jour avant de répondre au client afin d'assurer la tolérance aux fautes. Les autres protocoles accèdent uniquement à une copie avant de répondre, améliorant ainsi le temps de réponse. Les protocoles de duplication passive, semi-active et coordinateur/cohortes possèdent un maître décidant des mises à jour à faire sur les autres copies.

Dans les protocoles de duplication semi-active et coordinateur-cohortes toutes les copies reçoivent les requêtes externes, cependant le maître décide des mises à jour des autres copies. La variante de duplication passive avec maître dynamique diffère uniquement de la duplication passive dans le fait que le maître est dynamique et qu'il est nécessaire de n'autoriser qu'un écrivain en même temps afin de gérer la concurrence. Les protocoles de duplication passive, active, semi-active et coordinateur-cohortes propagent les mises à jour de manière immédiate. Dans les protocoles utilisées dans Grapevine, Lazy Replication et Coda la propagation est différée. Elle l'est également dans la variante de duplication passive avec point de reprise périodique qui s'appuie sur le nombre de mises à jour ayant était faites sur la copie maîtresse pour déclencher la mise à jour. Tous les protocoles adoptent une approche de type push, sauf Coda, Grapevine et Lazy Replication où l'approche est de type pull. Les protocoles de duplication passive Isis, Grapevine et Coda envoient des valeurs comme message de mise à jour alors que pour la duplication active, coordinateur cohorte Psync et Lazy Replication les opérations sont envoyées. La duplication semi-active envoie des valeurs ou des opérations selon qu'il y ait ou non indéterminisme dans les traitements. Tous les protocoles diffusent les mises à jour, sauf Grapevine, Coda et Lazy Replication qui les propagent de copie en copie. Des conflits peuvent apparaître avec Grapevine, Coda et Lazy Replication, nécessitant donc des mécanismes de résolution. Comme nous l'avons dit, tous s'appuient sur les propriétés des primitives de communication pour la propagation des mises à jour. Exception à la règle, Grapevine, Coda et Lazy Replication ne s'appuient pas directement sur ces propriétés. Tous ont pour objectif la tolérance aux fautes en utilisant des mécanismes particuliers, sauf Grapevine, Coda et Lazy Replication. La variante de duplication passive avec point de reprise périodique supporte les fautes, mais de manière restreinte (des mises à jour peuvent être perdues).

Certains protocoles diffèrent les uns des autres uniquement selon un certain point. Par exemple, les protocoles de duplication passive et sa variante avec point de reprise périodique sont différents dans le moment de la synchronisation. De plus pour certains protocoles, duplication passive, active, semi-active et coordinateur-cohortes nous avons présenté dans les sections précédentes une façon de faire. Mais il est tout à fait possible, par exemple d'implanter la duplication passive sans s'appuyer sur la couche de communication, en utilisant les transactions. On peut ainsi imaginer d'autres variantes des protocoles présentés, par exemple un protocole de duplication passive où le maître renvoie sa réponse au client et envoie le point de reprise aux autres copies simultanément, afin d'augmenter les performances.

2.3 Duplication dans les systèmes de gestion de bases de données réparties

Cette section est consacrée aux travaux sur la duplication dans les SGBDR. Nous commençons (section 2.3.1) par introduire leur principale spécificité, le contexte transactionnel, puis nous présentons différents travaux de recherche (section 2.3.2). En conclusion (section 2.3.3), nous reprenons la grille présentée en section 2.1 afin de comparer les protocoles présentés.

2.3.1 Contexte transactionel

Cette section rappel brièvement la notion de transaction (section 2.3.1.1), puis présente l'impact des transactions sur les protocoles de duplication (section 2.3.1.2). En fin de section (section 2.3.1.3), nous présentons la notion de cohérence dans les bases de données dupliquées.

2.3.1.1 Transaction

Dans la section précédente (section 2.2) une requête est une opération s'adressant à un objet. Dans le contexte des SGBDR, une transaction (une requête) est un ensemble d'opérations exécutées séquentiellement pouvant s'adresser à différents objets. Une transaction appliquée à une base cohérente restitue une base cohérente. Elle s'éxécute comme si elle était une unité de travail atomique : soit elle arrive à complétion (*commit*), soit elle n'a pas d'effet du tout (*abort*). En général, une transaction vérifie les propriétés ACID : A pour atomicité (toutes les opérations d'une transaction doivent être traitées comme une entité élémentaire), C pour cohérence (si elle est effectuée seule, une transaction transforme la base de données d'un état cohérent en un autre état cohérent), I pour isolation (une transaction en cours ne peut révéler ses résultats intermédiaires à d'autres transactions avant sa validation), D pour durabilité (les modifications validées (*commit*) sont toujours conservées même après une défaillance).

Une manière simple de mettre en œuvre le modèle transactionnel (sauf l'atomicité) consiste à exécuter chaque transaction l'une après l'autre tout en utilisant un mécanisme de recouvrement. Évidemment pour des raisons d'efficacité, on cherche à exécuter plusieurs transactions en parallèle. Il se pose alors la question du maintien de la cohérence. La solution la plus répandue est la sérialisabilité. Informellement, la sérialisabilité exprime le fait que l'exécution d'un ensemble de transactions doit être équivalente à une exécution séquentielle du même ensemble de transactions pour être cohérente. La sérialisabilité est considérée comme relativement peu efficace ; de nombreuses propositions de relâchement de la sérialisabilité sont faites pour améliorer les performances dans les bases de données (section 2.3.2).

2.3.1.2 Stratégies de propagation des mises à jour

Contrairement aux systèmes à communication de groupe (section 2.2), dans les SGBDR on ne raisonne plus au niveau des copies pour la propagation des mises à jour, mais au niveau des transactions. Deux stratégies existent : la propagation impatiente et la propagation paresseuse.

Définition 2.12 *: Propagation impatiente (eager replication)*

Lors d'une propagation impatiente des mises à jour, celles-ci sont appliquées sur toutes les copies dans une seule transaction répartie entre les différentes copies.

Cette approche permet de détecter les conflits avant que la transaction ne valide, assurant

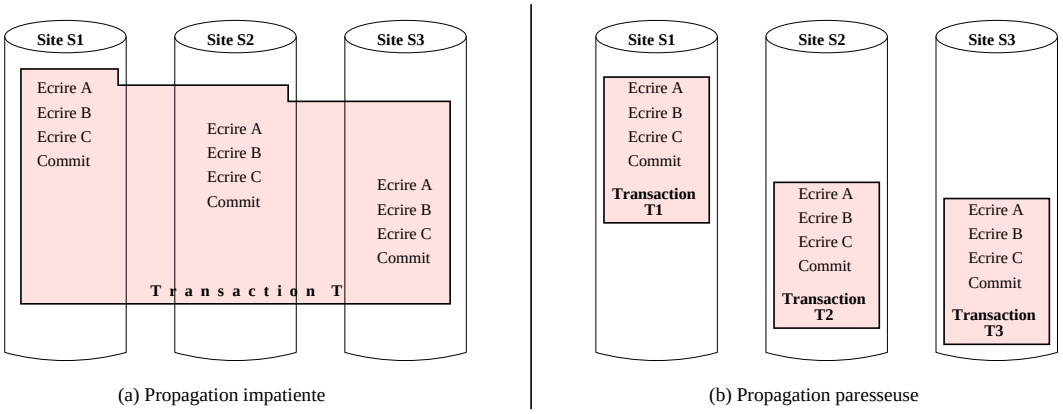


FIG. 2.11 – Stratégies de propagation dans les SGBDR

une cohérence forte entre les copies, mais un temps de réponse élevé. La figure 2.11 (a) donne un exemple de propagation impatiente des mises à jour. Les mises à jour des différentes copies des objets A, B et C des sites S1, S2 et S3 sont faites dans une même transaction.

Définition 2.13 : *Propagation paresseuse (lazy replication)*

Lors d'une propagation paresseuse des mises à jour, une seule copie est mise à jour par la transaction originelle. Les autres copies sont mises à jour de manière asynchrone par une transaction et cela pour chaque site possédant une copie.

Par rapport à la propagation impatiente, la propagation paresseuse est récompensée par une diminution du nombre de messages réseau et par une amélioration de la disponibilité des données. La figure 2.11 (b) donne un exemple de propagation paresseuse. Une première transaction met à jour les objets A, B et C du site S1. Ensuite, deux autres transactions se chargent de mettre à jour les sites S2 et S3. Cependant, avec l'approche paresseuse avec mises à jour par n'importe quelle copie, différentes copies d'un même objet peuvent être accédées simultanément par deux transactions en écriture. Des protocoles de réconciliation de conflit sont donc nécessaires afin que les copies convergent vers le même état.

Si l'on ne s'intéresse qu'à l'aspect duplication, suivant les caractéristiques présentées dans la section 2.1, la stratégie de propagation utilisée dans les SGBDR concerne à la fois : le nombre de copies concernées par une opération de lecture ou d'écriture, le moment de la synchronisation et la gestion de la concurrence.

2.3.1.3 Cohérences

Dans les SGBDR avec duplication, le critère de correction théorique le plus couramment utilisé est la sérialisabilité sur une copie (*one-copy serializability*) [BVG87]. Intuitivement,

elle exige que l'exécution des transactions sur les copies des objets soit équivalente à une exécution en série de ces mêmes transactions sur des objets logiques : tout se passe comme si les transactions s'éxécutent en série sur des objets non-dupliqués.

Définition 2.14 *: Sérialisabilité sur une copie*

Une exécution d'un ensemble de transactions sur une base de données dupliquée est sérialisable sur une copie si elle est équivalente à une exécution en série de ces transactions sur une base de données non dupliquée.

Ce critère de cohérence ne concerne pas uniquement la duplication, car il s'agit d'un ordre sur des transactions pouvant modifier différents objets dupliqués. Pour mettre en œuvre la sérialisabilité sur une copie, il faut s'assurer que toutes les transactions contenant des écritures sont totalement ordonnées. Un tel protocole peut se décomposer selon deux points :

- Assurer la sérialisabilité pour toutes les transactions contenant des écritures en supposant que les objets sont des objets logiques et
- Assurer que les mises à jour des différentes copies se font dans le même ordre total.

C'est ce que vise à faire les protocoles impatients. Les protocoles paresseux n'assurent pas la sérialisabilité des transactions en considérant des objets logiques. Ils considèrent chaque copie indépendamment des autres. Cependant, nous verrons dans la section 2.3.2.2 qu'il est quand même possible d'assurer la sérialisabilité sur une copie avec ces protocoles.

Selon la stratégie de propagation, la cohérence entre les copies d'un même objet n'est pas la même. Dans le cas impatient, une transaction contenant une écriture entraîne une mise à jour des copies de manière synchrone et suivant un ordre total. Dans le cas paresseux, cette écriture est asynchrone mais cependant la synchronisation des différentes copies doit permettre de construire un ordre total dans les mises à jour des copies. Ainsi, dans les SGBDR, il est défini deux sortes de cohérence entre les copies : la cohérence forte et la cohérence faible.

Définition 2.15 *: Cohérence forte*

Toute interrogation d'une copie quelconque reflète le résultat de toutes les modifications antérieures.

Définition 2.16 *: Cohérence faible*

Une interrogation ne reflète pas forcément toutes les modifications antérieures, mais avec la garantie que celles-ci soient toutes répercutées au bout d'un temps fini.

Il y a cohérence forte avec la stratégie de propagation impatiente et dans certaines formes de propagation paresseuse. Lorsqu'il y a cohérence faible, une ou plusieurs copies peuvent

être temporairement incohérentes (avoir une valeur ancienne) avec la garantie qu'au bout d'un temps fini elles deviendront cohérentes. C'est le niveau de cohérence que l'on rencontre généralement avec les protocoles paresseux.

On trouve également d'autres critères de correction plus permissifs comme la quasi-sérialisabilité [DE89], la M-sérialisabilité [RKC93], l'ϵ-sérialisabilité [PL91, RP95], la sérialisabilité causale [TK97], etc.

2.3.2 Quatre approches pour les protocoles de duplication

Initialement les protocoles de duplication proposés dans les SGBDR étaient des protocoles impatients [BVG87]. Malheureusement, ils coordonnent chaque opération de manière individuelle, en utilisant des mécanismes tel que le verrouillage réparti et la validation à deux phases (*2-phase commit*), engendrant des problèmes de passage à l'échelle. Quand le nombre de sites intervenant dans le système augmente, les temps de réponse des transactions, la probabilité d'avoir des conflits et des interbloquages (*deadlock*) augmentent de manière très importante. [GHOS96] conclut que la duplication impatiente ne peut pas être utilisée en pratique et propose d'utiliser les approches paresseuses. Cependant la duplication paresseuse pose de nouveaux problèmes : comment maintenir la cohérence entre les différentes copies et celle du système [CRR96]. Ainsi, parallèlement à ces travaux, de nombreuses recherches s'efforcent de proposer des protocoles impatients en utilisant les mécanismes de communication de groupe [KA00b].

Cette section est organisée de la manière suivante : nous commençons par présenter quelques protocoles impatients (section 2.3.2.1), puis des protocoles paresseux maître-esclaves (section 2.3.2.2), d'autres avec mise à jour sur n'importe quel site (section 2.3.2.3) et enfin quelques protocoles impatients utilisant les primitives de communication de groupe (section 2.3.2.4).

2.3.2.1 Protocoles impatients

Les premiers protocoles proposés implantent une approche impatiente primaire/secondaires[8] [AD76, Sto79]. Cependant, leur intérêt paraît limité par rapport à l'approche où les mises à jour peuvent se faire par l'intermédiaire de n'importe quelle copie. Dans cette section, nous nous intéressons donc à ces derniers. Ils se répartissent en trois catégories : ceux à synchronisation totale, ceux à synchronisation des copies disponibles et ceux à base de quorum. Tous ces protocoles garantissent la sérialisabilité sur une copie.

Synchronisation totale. Les protocoles à synchronisation totale sont les plus simples mais surtout les plus coûteux. Toutes les copies doivent être mises à jour de manière synchrone et,

[8] Dans le domaine des SGBDR dupliqué, on parle de primaire/secondaires plutôt que de maître/esclaves

bien souvent, l'atomicité des transactions est garantie par le protocole de validation à deux phases [Gra79]. Ainsi, dans le protocole ROWA (Read One Write All) [BG84] une lecture se fait sur n'importe quelle copie, mais une écriture doit être exécutée sur toutes les copies de manière synchrone et atomique. Ces protocoles ne s'avèrent d'aucune utilité du point de vue de la tolérance aux fautes : une seule défaillance d'un site peut bloquer tout le système.

Synchronisation des copies disponibles. Pour remédier aux limites des protocoles à synchronisation totale, dans les protocoles à synchronisation des copies disponibles, seules les copies disponibles sont mises à jour de manière synchrone. L'atomicité des transactions est garantie par le protocole de validation à deux phases. Une opération d'écriture peut être exécutée même si certaines copies ne sont pas disponibles, celles-ci seront mises à jour par des mécanismes asynchrones. Dans l'algorithme ROWAA (Read Only Write All Available) [CS86, GSC+83], une lecture peut se faire sur une copie arbitraire. Cependant, étant donné qu'une copie défaillante qui se remet à fonctionner ne reflète plus l'état courant de l'objet dupliqué, les transactions doivent connaître les copies non à jour et tous les sites doivent être d'accord sur les sites disponibles. Le protocole DOAC (Directory-Oriented Available Copies) [BVG87] utilise le concept de répertoire d'une donnée. C'est un ensemble de références sur les copies d'une donnée. Un répertoire peut être dupliqué. Lors d'une opération de lecture, le protocole localise une copie du répertoire de la donnée afin de trouver une copie disponible. Lors d'une opération d'écriture, une fois une copie du répertoire de la donnée localisée, le protocole envoie l'opération à chaque copie référencée. Si certaines sont indisponibles l'opération est annulée, les copies indisponibles sont supprimées de tous les répertoires de la donnée, puis l'opération est recommencée. Cette dernière étape garantit que les copies disponibles sont toujours à jour.

Quorum de copies. Pour ces protocoles, les mises à jour se font de manière synchrone sur un sous-ensemble des copies, qui forment un quorum en écriture. Les copies ne faisant pas partie de ce quorum sont mises à jour de manière asynchrone. Plus précisément, si V est le nombre total de copies, chaque opération de lecture ou d'écriture doit obtenir un *quorum de lecture Vl* ou un *quorum d'écriture Ve*, telles que : (1) $Vl + Ve > V$ et (2) $Ve > V/2$. La première condition évite les conflits lecture-écriture simultanées alors que la deuxième exclut les écritures simultanées [OV98]. De plus, à chaque copie est associé un numéro de version représentant le nombre de transactions validées qui ont modifié cette copie. Ce numéro de version permet de connaître les copies les plus à jour. Ainsi, une opération de lecture d'une transaction sur un objet, s'exécute sur une majorité de copies (Vl) et la valeur retournée est celle de la copie ayant le plus grand numéro de version. Une opération d'écriture d'une transaction sur un objet est envoyée à toutes les copies. Chaque copie doit exécuter cette opération et renvoyer un acquittement, accompagné de son numéro de version. L'écriture est abandonnée si une majorité de copies (Ve) et ayant le dernier numéro de version ne peut pas être accédé. De nombreux efforts sont fournis afin d'optimiser la taille des quorums (Quorum Consensus [Gif79, KA88], Tree Quorum Protocol [AA90b, AA90a], Missing Writes [Eag81, ES83], Virtual Partition [ASC85, AT86]). Le protocole à partition virtuelle, permet qu'une lecture se fasse sur une seule copie.

2.3.2.2 Protocoles paresseux maître-esclaves

Les protocoles paresseux entraînent moins d'interbloquage que les protocoles impatients, les transactions étant beaucoup plus courtes. Cependant la sérialisabilité sur une copie devient plus délicate à assurer. Dans cette section, nous nous focalisons sur les protocoles maître-esclaves, avec des protocoles proposant un affaiblissement de la cohérence au niveau des copies, et d'autres garantissant la sérialisabilité sur une copie. En fin de section, nous présentons des protocoles combinant les approches impatiente et paresseuse.

Affaiblissement de la cohérence. Certaines applications peuvent fonctionner (sans compromettre leur sémantique) avec des copies non à jour [BGM90]. Ainsi, de nombreux protocoles sont proposés où les différentes copies d'un même objet ne sont pas mises à jour de manière atomique et synchrone [PL91, KB91, GN95]. Ces protocoles déclenchent le processus de synchronisation selon des conditions, comme celles présentées dans la section 2.1.3.

Le protocole "As soon as possible" [BVG87], exécute les opérations d'écriture sur la copie primaire, puis les écritures validées sont collectées et envoyées aux autres copies comme des transactions indépendantes le plus tôt possible. Dans le protocole Quasi Copies [ABMA88], un site central contient les copies primaires. A chaque copie secondaire sont associées des conditions de cohérence définissant les déviations possibles avec la copie primaire. Ces conditions peuvent être associées au temps, à la version ou à la valeur. Le site central, quand il est opérationnel, doit s'assurer que chaque site distant reçoit un message toutes les s secondes. Si un site distant ne reçoit pas de message après s secondes, il en déduit que le site central est en panne. La propagation des mises à jour par le site primaire peut être faite tout de suite, à n'importe quel moment avant la violation des conditions de cohérence, au moment où une condition de cohérence va être violée ou retardée sur le site primaire de telle façon qu'aucune condition de cohérence n'est violée. Le protocole "Differential File" [SL76] utilise un fichier afin d'enregistrer les changements effectués sur la copie primaire et pour mettre à jour les copies. La mise à jour se fait quand une copie est accédée, sur la demande de l'utilisateur ou bien périodiquement. D'autres protocoles sont plus orientés tolérance aux fautes. Les opérations d'écriture sont toujours faites sur la copie primaire, mais une des copies secondaires, désignée comme sauvegarde, est mise à jour de manière synchrone. Elle est responsable de la reprise sur défaillance de la copie primaire. Toutes les autres copies secondaires sont mises à jour de manière asynchrone.

Solutions garantissant la sérialisabilité sur une copie. Suite aux travaux relâchant la cohérence, de nombreux auteurs proposent des protocoles paresseux garantissant la sérialisabilité sur une copie. [CRR96] montre qu'il est possible de propager les mises à jour de manière paresseuse tout en garantissant la sérialisabilité sur une copie si et seulement si le graphe non orienté obtenu à partir du graphe des copies[9] ne contient pas de cycle. De plus, à partir d'un

[9] Un graphe des copies est un graphe dans lequel chaque noeud représente un site S_i. Il y a un arc orienté entre S_i et S_j si S_i est le site primaire d'une donnée dupliquée x et S_j un site secondaire pour x

ensemble de données dupliquées, il donne un algorithme proposant, si c'est possible, les sites sur lesquels doivent se trouver les copies primaires pour assurer la sérialisabilité sur une copie.

D'autres travaux étendent la classe des graphes de copies pour lesquels il est possible de garantir la sérialisabilité sur une copie. Les protocoles DAG(WT) et DAG(T) [BKR+99] permettent d'obtenir la sérialisabilité sur une copie dans le cas de graphes des copies orientés acycliques, en contrôlant l'ordre dans lequel sont appliquées les mises à jour sur les copies secondaires. Le protocole DAG(WT), à partir du graphe des copies, construit une forêt[10] satisfaisant la propriété suivante : si le site S_i est un fils du site S_j dans le graphe des copies, alors S_i est un descendant de S_j dans la forêt. Ainsi, une transaction de mise à jour qui valide sur le site S_i est envoyée à ces fils dans l'arbre. Les transactions sont exécutées et renvoyées aux fils dans l'ordre reçu. Les sites recevant des transactions ne les concernant pas se contentent de les renvoyer à leurs fils. DAG(WT) engendre donc des messages et des traitements inutiles. Le protocole DAG(T) utilise des estampilles pour propager les mises à jour directement le long des arcs du graphe des copies, évitant ainsi des traitements inutiles. Ces estampilles sont affectées par les sites primaires aux transactions envoyées aux sites secondaires. Ensuite, sur chaque site, les transactions sont éxécutées dans l'ordre des estampilles afin d'assurer la sérialisabilité. Intuitivement, l'estampille d'une transaction arrivant sur un site secondaire contient l'information permettant de connaître les transactions devant être exécutées avant. [PMS99] propose aussi un protocole étendant la classe des graphes de copies pour lequel il est possible de garantir la sérialisabilité.

Duplication paresseuse impatiente. Des protocoles hybrides impatient paresseux permettent d'assurer la sérialisabilité sur une copie même si le graphe des copies contient des cycles. "The backEdge protocol" [ABKW97, BKR+99] adopte une stratégie paresseuse dans les parties du graphe des copies ne contenant pas de cycle et une stratégie impatiente pour les parties cycliques.

2.3.2.3 Protocoles paresseux avec mise à jour sur n'importe quel site

Ces protocoles se destinent particulièrement bien aux environnements mobiles, les déconnexions fréquentes obligeant à maintenir des copies sur les hôtes mobiles sans possibilité de synchronisation. Cependant, il est très difficile de maintenir la sérialisabilité sur une copie. Les mises à jour pouvant se faire par l'intermédiaire de n'importe quel site, deux sites peuvent mettre à jour de manière concurrente le même objet logique par l'intermédiaire de deux de ses copies. Des mécanismes de détection des conflits et de réconciliation sont alors nécessaires afin d'assurer la sérialisabilité sur une copie et la convergence des différentes copies vers le même état.

Généralement, des estampilles sont utilisées pour détecter et réconcilier les copies, chaque copie portant l'estampille de sa plus récente mise à jour. Chaque propagation de mise à jour

[10] Une forêt est une collection d'arbres où un arbre est un graphe orienté dans lequel chaque noeud a exactement un parent, sauf la racine.

contient la nouvelle valeur de l'objet ainsi que la précédente valeur de l'estampille de l'objet. Un site recevant une demande de mise à jour teste si l'estampille de la copie locale est égale à la valeur de l'estampille propagée. Si c'est le cas la mise à jour est sûre, l'estampille de la copie locale et sa valeur sont actualisées. Sinon la mise à jour est dite dangereuse. Dans ce cas, le site rejette la transaction de mise à jour et la soumet pour une réconciliation.

Le protocole de démarcation [BGM91] propose une approche extrême. Des contraintes locales sont formulées sur chaque copie, afin d'assurer que celles-ci pourront être fusionnées par la suite. Par exemple, dans une application de vente de billets d'avion, on décompose un avion de 100 places en deux copies sur lesquelles il est permi de vendre 50 places. Ce protocole présume que les opérations changeant la valeur d'une donnée sont commutatives.

2.3.2.4 Protocoles impatients utilisant les primitives de communication de groupe

Une nouvelle voie de recherche s'attache à améliorer les performances des protocoles impatients en s'appuyant sur les primitives de communication de groupe (*multicast*) [AAAS97, PGS97, HAA99, KA98, KA00b]. Les supports à la communication de groupe ayant nettement avancés, il est possible d'en tirer profit pour optimiser le processus de synchronisation.

[KA00b][KA00a] proposent des protocoles où une transaction est d'abord exécutée localement, puis la propagation des mises à jour est effectuée au moment de sa validation. Les mises à jour sont envoyées à toutes les copies en utilisant une diffusion de groupe avec ordre total. Ceci garantit que tous les sites reçoivent les modifications dans le même ordre. Lors de la réception, chaque site vérifie les conflits lecture/écriture. Les transactions sans conflits sont validées. Les écritures conflictuelles sont exécutées dans l'ordre d'arrivée produisant ainsi un ordonnancement en série. Un protocole de validation à deux phases explicite n'est plus nécessaire. Les interblocages peuvent également être évités si toutes les opérations d'une transaction sont envoyées dans un même message. Puisque les transactions sont "ordonnées" selon l'ordre d'arrivée des messages (le même sur tous les sites), il suffit d'octroyer les verrous aux transactions dans cet ordre. Notons qu'avec cet approche, il est possible d'effectuer des mises à jour par l'intermédiaire de n'importe quelle copie. [GHOS96] a démontré qu'avec des protocoles impatients classiques, et dans certaines configurations, la probabilité d'interblocages est directement proportionnelle à n^3, où n est le nombre de copies. Cette nouvelle approche permet donc de passer cet obstacle.

2.3.3 Conclusion

La plupart du temps, dans le contexte des SGBDR l'objectif est d'obtenir la sérialisabilité sur une copie. Ce critère de cohérence ne concerne pas uniquement la cohérence entre les copies, mais également la cohérence entre l'ensemble des objets du système.

La figure 2.12 compare les différents protocoles décrits dans cette section selon les caractéristiques présentées dans la section 2.1.

Nombre copie pour E ou L	**Une** DP DI		**0<n<toutes** ROWAA DOAC Q		**Toutes** DI			
Droits de mise à jour	**Un maitre**		**Plusieurs maitres**		**N'importe quelle copie** DI DPUA			
	Dynamique	**Statique** DI DPME S	**Dynamique**	**Statique**				
Moment de la synchronisation	**Immédiate** DI	**Différée** DP						
		Délai QC ASAP	**Périodicité**	**Moment**	**Version** QC	**Numérique** QC	**Objets**	**Evénements**
Initiative de la mise à jour	**Push** DI DP				**Pull** DP			
Nature des mises à jour	**Valeur** DI DP		**Opération** DI DP		**Invalidation**			
Topologie	**Etoile** DI DP		**Copie en Copie** DPUA		**Groupe en Groupe** DPUA		**Chemin particulier** DAG(T) DAG(WT)	
Détection des mises à jour	**Journal** DF S I DB2		**Copie ombre**		**Trigger** O In		**API** V	**Lecture de la copie**
Conflits	**Oui** DPUA				**Non** DI DPME			
Couche com.	**Oui** DIPC				**Non** DI DP			
Concurrence	**n L ou 1 E** DI DIPC		**n L ou n E**		**n L et 1 E** DPME		**n L et n E** DPUA	
Fautes	**Oui** ROWAA DOAC Q				**Non** ROWA DPME DPUA			

DI : duplication impatiente DP : duplication paresseuse DPUA : duplication paresseuse à copies identiques
DIPC : duplication impatiente + primitives communication de groupe DPME : duplication paresseuse maitre esclaves
ROWA : protocole read one write all ROWAA : protocole read one write all available QC : protocole quasi copies
DOAC : protocole directoty oriented available copies ASAP : protocole as soon as possible Q : quorum
DAG(T) : protocole DAG(T) DAG(WT) : protocole DAG(WT) DF : protocole differential file

FIG. 2.12 – Classification des protocoles de duplication utilisés dans les SGBDR

En résumé, les stratégies paresseuses et impatiente diffèrent d'abord sur le nombre de copies consultées lors d'une requête externe de lecture ou d'écriture : l'approche paresseuse consulte une copie et l'approche impatiente toutes les copies lors d'une écriture et une copie lors d'une lecture. Les deux types de stratégies peuvent adopter les approches maître/esclaves et "n'importe quelle copie". Dans une stratégie impatiente la propagation des mises à jour est immédiate, alors que dans une stratégie paresseuse elle peut être différée. Les deux approches sont de type push, mais l'approche paresseuse peut aussi être de type pull. Elles peuvent toutes les deux propager des valeurs ou des opérations. Avec les stratégies impatientes et

paresseuses maître-esclaves, les mises à jours sont généralement diffusées à toutes les copies. Avec l'approche paresseuse avec mise à jour par l'intermédiaire de n'importe quelle copie, les mises à jour peuvent se faire de copie en copie ou de groupe en groupe. Avec les stratégies impatientes et paresseuses maître-esclaves, il ne peut y avoir de conflit contrairement à l'approche paresseuse avec mise à jour sur n'importe quelle copie. L'approche impatiente ne supporte qu'un écrivain à un instant donné alors que l'approche maître-esclaves supporte plusieurs lecteurs et un écrivain. L'approche paresseuse avec mise à jour sur n'importe quelle copie supporte plusieurs lecteurs et plusieurs écrivains en même temps.

En ce qui concerne les implantations de la stratégie impatiente, ROWA ne supporte pas les fautes. ROWAA et Directory Oriented Available Copies (DOAC) ne consultent qu'un sous ensemble des copies lors de l'écriture. Les protocoles à base de quorum ne consultent qu'un sous-ensemble des copies lors des écritures et des lectures. ROWA ne supporte pas les fautes contrairement à ROWAA, DOAC et les quorums. Pour les protocoles paresseux maître-esclaves, "As soon as possible", "Quasi Copies" et "Differentiated File" se basent sur des conditions sur le délai, la version ou numérique pour déclencher la synchronisation. Les protocoles DAG(T) et DAG(WT) propagent les mises à jour suivant des chemins particuliers.

La figure 2.12 présente également les caractéristiques en ce qui concerne le suivi des mises à jour de quelques systèmes commerciaux : Oracle version 8 [BS97] (O), Sybase version 12 [Syb00] (S), Informix [Inf98] (I), Ingres [ZAL96] (In) et Versant [Ver99b, Ver99a] (V). En ce qui concerne les autres points, tous ces systèmes proposent des protocoles impatients et paresseux avec différentes variantes selon le produit.

2.4 Duplication dans les mémoires partagées réparties

Cette section est consacrée aux travaux sur la duplication dans les mémoires partagées réparties. Nous commençons par introduire une notion fondamentale dans ce contexte, les modèles de cohérence (section 2.4.1). Ensuite, nous présentons les modèles de cohérence sans synchronisation (section 2.4.2), puis ceux avec synchronisation (section 2.4.3). En conclusion (section 2.4.4), nous reprenons notre grille (section 2.1) afin de comparer chaque protocole.

2.4.1 La notion de modèle de cohérence

Une mémoire partagée construite sur un système réparti est une mémoire partagée répartie (*Distributed Shared Memory DSM*). Un système implantant une telle mémoire est un ensemble de sites (ou de processeurs) doté chacun d'un gestionnaire de mémoire et communiquant entre eux par passage de messages. Les processus au niveau applicatif communiquent par accès (lecture et écritures) à la mémoire partagée. Une mémoire partagée libère donc le programmeur d'application de l'architecture sous jacente puisqu'il n'a plus qu'à considérer le paradigme de programmation par variables partagées.

Les mémoires partagées réparties introduisent la notion de modèle de cohérence. Un modèle

de cohérence [AH90] est un contrat entre l'application et la mémoire partagée qui spécifie formellement les accès à cette mémoire et leur perception par le programmeur. Un protocole de cohérence implante un modèle de cohérence particulier et un modèle de cohérence peut être implanté par différents protocoles. Dans un système avec duplication, un protocole de cohérence sert à imposer un certain degré de synchronisation entre les différentes copies, et impose un certain ordre sur ces mises à jour.

Le modèle de cohérence le plus naturel est le modèle atomique garantissant un comportement de mémoire centralisée où toutes les opérations d'accès sont effectuées de façon exclusive. Dans un contexte avec duplication, un seul gestionnaire de mémoire est autorisé à écrire sur un objet à un instant donné. Ce modèle facile à utiliser se révèle très peu efficace. Ainsi, de nombreux autres modèles sont proposés afin de tirer profit du parallélisme et de la répartition. Cependant, il est important de comprendre que la sémantique centralisée est remise en cause. Deux écritures concurrentes sur un même objet logique peuvent donner deux valeurs distinctes sur deux copies. La convergence des copies n'est pas obligatoire.

Comme pour les systèmes à communication de groupe, et contrairement aux SGBDR, la propagation des mises à jour se fait au niveau des copies. Les modèles de cohérence existant se répartissent en deux catégories : les modèles sans synchronisation et ceux avec synchronisation. Les modèles de cohérence sans synchronisation utilisent les seules primitives de lecture et d'écriture. Dans la littérature, ils sont également appelés modèles de cohérence fort, car ne faisant pas intervenir le programmeur d'application pour assurer la cohérence. Les modèles avec synchronisation utilisent, en plus des primitives de lecture et d'écriture, des primitives manipulant des variables de synchronisation. Avec les modèles de cohérence sans synchronisation, les opérations pour synchroniser les processus se font par des mécanismes indépendants. Or, en intégrant ces synchronisations avec les modèles de cohérence, il est possible de relâcher des contraintes, et donc d'obtenir un protocole de cohérence moins coûteux. Par exemple, il est fréquent de protéger des variables partagées par une exclusion mutuelle entre les processus. Le processus possédant la section critique est donc le seul à lire et à modifier ces variables. Il est donc judicieux d'alléger la cohérence, puisque ces variables ne sont plus partagées dans ce cas. Les modèles de cohérence avec synchronisation sont aussi appelés modèles de cohérence faibles car faisant intervenir le programmeur dans la gestion de la cohérence.

En résumé, un modèle de cohérence permet d'abstraire la répartition, la concurrence, la tolérance aux fautes et la duplication. Tous ces aspects sont à la charge du protocole de cohérence. Dans les deux sections suivantes, nous présentons quelques modèles avec synchronisation (section 2.4.2) et sans synchronisation (section 2.4.3). Pour chaque modèle, nous donnons sa définition, ainsi que des protocoles l'implantant. Pour les protocoles présentés, nous mettons en valeur ce qui concerne la duplication dans la section 2.4.4.

2.4.2 Modèles de cohérence sans synchronisation

Cette section présente les modèles de cohérence atomique (section 2.4.2.1), séquentielle (section 2.4.2.2), causale (section 2.4.2.3), PRAM (section 2.4.2.4), objet (section 2.4.2.5), et

à la longue (section 2.4.2.6), ainsi que divers protocoles les implantant.

2.4.2.1 Cohérence atomique

Le modèle de cohérence atomique garantit un comportement similaire à celui de la mémoire centralisée [CF78]. Le modèle de cohérence atomique apparaît dans les systèmes dont l'exigence première est un modèle de programmation facile d'utilisation. Cette facilité d'utilisation est obtenue au détriment des performances.

Définition 2.17 : *Modèle de cohérence atomique*

Un modèle de cohérence atomique garantit qu'une opération de lecture sur une donnée retourne la valeur affectée par la dernière opération d'écriture sur cette donnée.

Protocoles : Ivy [LH89] propose un protocole de cohérence implantant ce modèle. C'est un protocole à invalidation des écritures. Chaque donnée (une page dans ce protocole) est possédée par un processus, à savoir le dernier processus qui a écrit dans cette donnée. Quand un processus veut lire une page dont il n'a pas de copie, il envoie une requête au gestionnaire de la page qui fait suivre cette requête au propriétaire courant. Quand le propriétaire reçoit une telle requête, il envoie une copie de la page au processus demandeur et invalide son droit d'accès associé à la page. Quand un processus veut écrire une page, il envoie, via le gestionnaire de la page correspondante, une requête au propriétaire courant. Lors de la réception d'une telle requête, le propriétaire invalide d'abord toutes les copies qu'il a précédemment disséminées sauf la sienne, et envoie ensuite sa copie au processus demandeur. Le processus demandeur est alors le nouveau propriétaire de la page, et aucun autre processus n'a de copie de la page. Ces mécanismes garantissent l'atomicité entre n'importe quelle paire d'opérations de lecture et d'écriture, et n'importe quelle paire d'opérations d'écriture. Par contre, les paires d'opérations lecture/lecture peuvent être concurrentes.

La principale difficulté est de trouver des mécanismes pour gérer de manière efficace le propriétaire d'une page. Il existe des implantations proposant des gestionnaires centralisés ou répartis, avec propriétaires fixes ou dynamiques.

La cohérence atomique peut aussi être mise en oeuvre en utilisant un protocole à base de quorum comme présenté dans la section 2.3.2.1.

2.4.2.2 Cohérence séquentielle

Ce modèle de cohérence a été pour la première fois proposé par [Lam79] pour définir un critère de correction pour des systèmes multi-processeurs à mémoire partagée. Un ordre séquentiel est obtenu en entrelaçant toutes les opérations de tous les processus et en conservant l'ordre des opérations effectuées par chaque processus. Ce modèle garantit que tous les processus perçoivent les opérations dans le même ordre. Cependant, il ne garantit pas qu'un

processus effectuant une opération de lecture obtiendra la dernière valeur affectée par une opération d'écriture émise par un autre processus. La différence fondamentale entre la cohérence séquentielle et la cohérence atomique réside dans la signification du mot "dernière". Dans le cas de la cohérence séquentielle, "dernière" se réfère au temps logique alors que pour la cohérence atomique, le mot "dernière" se réfère au temps physique.

Définition 2.18 *: Modèle de cohérence séquentielle*

Un modèle de cohérence séquentielle garantit que le résultat de n'importe quelle exécution est le même que si les opérations de tous les processeurs étaient exécutées dans un certain ordre séquentiel, et les opérations de chaque processeur apparaissent dans cette séquence dans l'ordre spécifié par son programme.

Protocoles : Un protocole pour ce modèle de cohérence doit essentiellement garantir l'ordre total des opérations. Cela peut être fait grâce à un protocole de diffusion atomique (par exemple [AGM93] et [AW94]). Dans [MRZ94], un processus P_i effectuant une opération d'écriture envoie un message d'écriture à un gestionnaire central et attend une réponse. Le gestionnaire central ordonne totalement toutes les opérations d'écriture. Après avoir reçu un message d'écriture venant d'un processus P_i, le gestionnaire central lui répond en lui communiquant la liste des variables pour lesquelles il possède des valeurs obsolètes et par conséquent pour lesquelles les futures lectures ne sont plus légales. Deux versions de ce protocole sont décrites. Dans la première, les variables, dont les futures lectures par P_i sont illégales, sont invalidées dans sa mémoire locale. Dans la seconde, le gestionnaire informe le processus P_i des valeurs courantes associées avec ces variables.

Dans le contexte des systèmes répartis, des protocoles mettant en oeuvre la cohérence séquentielle sont proposés. Cependant, généralement, ces protocoles utilisent les mécanismes de vote ou de quorum et par conséquent, implantent en fait le modèle de cohérence atomique.

2.4.2.3 Cohérence causale

Bien que conceptuellement simple et facile d'utilisation, un système assurant la cohérence atomique ou séquentielle impose d'importantes restrictions sur les accès nuisant à l'efficacité. Ainsi des modèles relâchant la cohérence tout en fournissant un modèle de programmation raisonnable (c'est à dire facile d'utilisation) pour le programmeur sont proposés. Le modèle de cohérence causale, basé sur la causalité potentielle décrite dans [Lam78], en est un. La cohérence causale [HA90, ABHN91] n'est que le relâchement de la cohérence séquentielle en fonction de la causalité entre les opérations (aucune contrainte sur les opérations d'écriture concurrentes). Ainsi, avec un protocole de cohérence causal, tous les processus perçoivent les opérations d'écriture causalement liées dans le même ordre, et il n'y a aucune contrainte sur les opérations concurrentes (c'est à dire sans lien de causalité). Il s'agit donc d'un ordre partiel sur les opérations d'écriture. Il y a donc moins de synchronisation que dans le modèle de cohérence séquentielle, mais les copies d'un objet peuvent diverger les unes des autres.

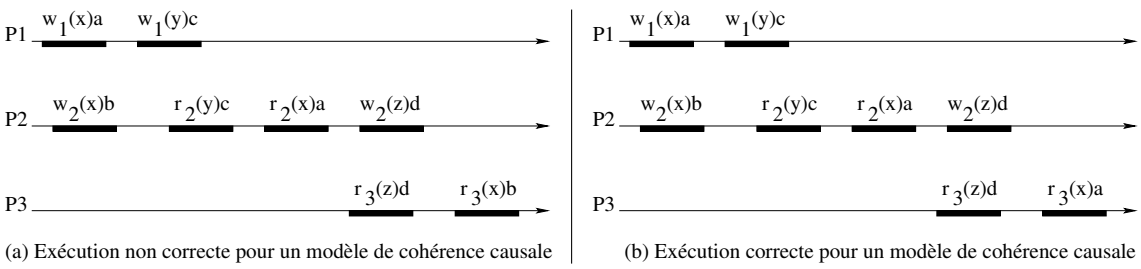


FIG. 2.13 – Relation modèle de cohérence/duplication

Définition 2.19 *: Modèle de cohérence causale*

Un modèle de cohérence est causal s'il garantit les deux conditions suivantes :
- Les opérations d'écriture potentiellement causalement liées doivent être perçues par tous les processus dans le même ordre ;
- Les opérations d'écriture non causalement liées peuvent être perçues dans des ordres différents, sur des processus différents.

La figure 2.13 montre un exemple d'exécution ne satisfaisant pas le modèle de cohérence causale et un autre le satisfaisant. Cet exemple montre bien qu'un modèle de cohérence considère plus que la duplication. Dans la figure (a) le processus P_3 ne peut lire x = b, car suivant la relation de causalité s'il a lu z=d il doit ensuite lire x=a (figure (b)).

Protocoles : La mise en œuvre d'un modèle de cohérence causale s'avère compliquée car il faut garder trace de la relation de causalité entre les différents objets. Il peut être mis en œuvre à l'aide d'horloges vectorielles.

2.4.2.4 Cohérence PRAM

Le modèle de cohérence PRAM [LS88] (*Pipelined Randomize Access Memory*) relâche le modèle de cohérence causale. Certaines transitivités dues à la relation de causalité ne sont pas considérées. La cohérence PRAM n'influe que sur la relation entre deux processus alors que la cohérence causale influe sur la relation entre plusieurs processus.

Définition 2.20 *: Modèle de cohérence PRAM*

Un modèle de cohérence est PRAM s'il garantit que les opérations d'écriture effectuées par un même processus sont perçues par tous les processus dans l'ordre où ces opérations ont été effectuées. Les opérations d'écriture effectuées par différents processus peuvent être perçues par chaque processus dans un ordre différent.

Protocoles : Un protocole de cohérence PRAM est facilement implantable sous la forme

de files d'opérations d'écriture, mais c'est un modèle difficile à utiliser.

2.4.2.5 Cohérence objet

Informellement, la cohérence objet (connue également sous le nom de cohérence de cache) assure la cohérence séquentielle pour chaque objet logique : c'est à dire un accès séquentiel pour chaque objet logique.

Définition 2.21 *: Modèle de cohérence objet*
Un modèle de cohérence est objet si les opérations d'écriture sur un même objet sont totalement ordonnées.

Protocoles : Un protocole implantant ce modèle doit garantir un ordre global sur les écritures pour chaque objet logique. Aucun ordre n'est imposé entre les écritures sur différents objets.

2.4.2.6 Cohérence "à la longue"

Ce modèle de cohérence est le plus faible, aucun ordre entre les opérations n'est imposé.

Définition 2.22 *: Modèle de cohérence "à la longue"*
Un modèle de cohérence est "à la longue" s'il garantit que les opérations d'écriture sont propagées tôt ou tard.

Protocoles : Un protocole implantant ce modèle doit tout simplement propager les mise à jour entre les copies quand il le désire, il n'y a aucune contrainte entre les mises à jour de différents objets.

2.4.3 Modèles de cohérence avec synchronisation

Cette section présente les modèles de cohérence faible (section 2.4.3.1), au relâchement (section 2.4.3.2) et à l'entrée (section 2.4.3.3), ainsi que divers protocoles les implantant.

2.4.3.1 Cohérence faible

Le modèle de cohérence faible [DSB86] (*weak consistency*) fait intervenir des variables de synchronisation que le programmeur doit accéder de manière explicite pour gérer la cohérence. Quand un processus effectue une opération sur une variable de synchronisation, il a la garantie que, d'une part, toutes les modifications qu'il a effectuées sont reçues par tous les autres processus et, d'autre part, qu'il a reçu toutes les modifications faites par tous les autres

processus. Si un processus modifie plusieurs fois de suite la même variable entre deux accès à une variable de synchronisation, seule la dernière valeur est propagée aux autres processus. Ce modèle permet à un protocole de cohérence faible, de regrouper et de propager dans un seul message toutes les modifications qu'un processus a effectuées. Néanmoins, les accès aux variables de synchronisation génèrent des messages supplémentaires.

Définition 2.23 *: Modèle de cohérence faible*

Un modèle de cohérence est faible s'il garantit les trois propriétés suivantes :
- Les opérations sur les variables de synchronisation se font selon un modèle de cohérence séquentielle ;
- L'accès aux variables de synchronisation ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites ;
- Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations sur les variables de synchronisation sont terminées sur tous les sites.

Protocoles : Ce modèle de cohérence s'implante très simplement à l'aide de barrières de synchronisation. De plus, le protocole doit savoir quelles sont les modifications effectuées par un processus entre deux accès à une variable de synchronisation.

2.4.3.2 Cohérence au relâchement

Dans le précédent modèle, on ne distingue pas le commencement d'un accès de sa fin (c'est plus des barrières de synchronisation). [GLM+90] définit le modèle de cohérence au relâchement (*release consistency*). Deux opérations sur les variables de synchronisation existent : acquérir (*acquire*) et relâcher (*release*). Les opérations sur les variables de synchronisation sont régies par une cohérence processeur (celles issues d'un même processus sont observées dans le même ordre par tous les processus). Quand un processus effectue une opération d'acquisition, il observe toutes les modifications effectuées par les autres processus avant les relâchements précédents. Quand un processus effectue une opération de relâchement, les modifications qu'il a effectuées seront observées par tous les processus faisant une acquisition ultérieure.

Définition 2.24 *: Modèle de cohérence au relâchement*

Un modèle de cohérence est au relâchement s'il garantit les trois propriétés suivantes :
- Les opérations de synchronisation (acquire et release) se font selon un modèle de cohérence processeur ;
- Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations d'acquisition (acquire) précédentes sont terminées sur tous les sites ;
- L'opération de relâchement (release) ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites.

Protocoles : Il existe deux protocoles mettant en œuvre ce modèle : le protocole de cohérence au relâchement impatient et le protocole paresseux. Dans le protocole de cohérence au relâchement impatient (*eager release consistency*), toutes les modifications entre les deux opérations de synchronisation sont propagées au moment du relâchement à tous les processus, même ceux qui n'effectueront pas d'acquisition ultérieure. Munin [CBZ91] implante ce protocole.

Dans le protocole de cohérence au relâchement paresseux (*lazy release consistency*), toutes les modifications entre les deux opérations de synchronisation sont propagées au moment d'une acquisition par un autre processus. Le surcoût dû à ce protocole est compensé par le nombre et la taille des messages échangés. [Kel95] implante ce protocole. Une autre variante, TreadMarks [ACD+96], propage les modifications au moment du premier accès après l'acquisition.

2.4.3.3 Cohérence à l'entrée

Le modèle de cohérence à l'entrée [BM91] (*entry consistency*) affaiblit encore celui au relâchement. Il se distingue par l'association d'une variable de synchronisation s pour une ou plusieurs variable partagées Ds.

Définition 2.25 *: Modèle de cohérence à l'entrée*

Un modèle de cohérence est à l'entrée s'il garantit les trois propriétés suivantes :

- Une acquisition (acquire) d'une variable de synchronisation s ne peut se terminer que si toutes les modifications de Ds ont été effectuées ;
- Avant qu'un accès exclusif à une variable de synchronisation soit susceptible de se terminer, plus aucun processus ne doit détenir cette variable de synchronisation dans un mode non exclusif ;
- Après qu'un accès exclusif par un processus p à une variable de synchronisation s soit terminé, tous les accès en mode non exclusif à cette variable s ne pourront se terminer que quand toutes les modifications de Ds par p seront mises à jour.

Aucun processus ne peut obtenir des valeurs de Ds plus vieilles que la dernière modification si tous les accès à Ds sur tous les processus se situent dans une section critique représentée par la variable de synchronisation s. Une variable de synchronisation s ne peut pas être détenue par plus d'un processus. Il s'agit du propriétaire de s qui est le dernier processus ayant fait un accès exclusif à s. En mode non exclusif, plusieurs processus peuvent avoir une copie de la variable de synchronisation. La dernière propriété assure que les processus peuvent faire des accès en mode non-exclusif à une variable de synchronisation sans communiquer avec le propriétaire de s tant qu'un accès à s en mode exclusif n'a pas été effectué.

Protocoles : Une variable de synchronisation s sert à protéger des données partagées Ds (comme le ferait une section critique). Si durant cette section critique, Ds sont modifiées, il

faut acquérir s en mode exclusif. Par contre, si Ds sont consultées, il faut acquérir s en mode non-exclusif. Cela s'apparente donc à un verrou écrivain/lecteurs.

Ce modèle permet des accès concurrents aux différentes sections critiques protégeant des variables indépendantes. Cependant, l'association des variables de synchronisation aux variables partagées est manuelle, ce qui introduit un risque d'erreur non négligeable.

2.4.4 Conclusion

Les protocoles présentés dans cette section sont plus que des protocoles de duplication. Ils ordonnancent des messages échangés entre différents processus et concernant des objets différents. On peut discerner deux types de protocoles de cohérence dans le contexte des MPR. Le premier type ordonnance les opérations de lecture et d'écriture en ne s'appuyant sur aucune information provenant de l'application. Ces opérations portent sur des objets différents. Ces protocoles implantent les modèles de cohérence sans synchronisation (ou fort). Le deuxième type ordonnance les opérations de lecture et d'écriture d'après des informations provenant de l'application. Ces protocoles implantent les modèles de cohérence avec synchronisation. Cependant, on peut discerner les modèles où l'application décide des points de synchronisation (modèle de cohérence faible et au relâchement), des modèles ordonnançant des sections critiques (modèle de cohérence à l'entrée), ces derniers modèles se rapprochant plus des modèles rencontrés dans les SGBDR. On remarque cependant que l'on retrouve les points propres à la duplication (figure 2.14) malgré le type du modèle de cohérence (cohérence au sens MPR) à mettre en œuvre.

2.5 Conclusion

Dans ce chapitre, nous avons dégagé certaines caractéristiques communes aux protocoles de duplication. Ces caractéristiques se retrouvent dans des protocoles aussi différents que ceux ayant pour objectif d'assurer une cohérence forte entre les copies que ceux affaiblissant la cohérence entre les copies. Elles se retrouvent également dans des protocoles utilisés dans des domaines aussi différents que les SCG, les SGBDR ou les MPR. Ces caractéristiques sont : le nombre de copies concernées par une lecture ou une écriture, les droits d'accès, le moment de la synchronisation, l'initiative de la mise à jour, la nature des mises à jour, la topographie de la synchronisation, la capture des mises à jour, la gestion des conflits, les prérequis sur le protocole de communication, la gestion de la concurrence, la gestion de la tolérance aux fautes, la notion de copie et la transparence à la duplication.

Lors de la présentation des protocoles de duplication, nous avons également montré que chacun des trois domaines apporte ces spécificités. Dans le cas des SGBDR, le protocole de duplication participe à la mise en œuvre d'un critère de correction, la sérialisabilité sur une copie (ou un affaiblissement de la sérialisabilité), et dans le cas des MPR à la construction d'un modèle de cohérence spécifique. En résumé, deux niveaux de maintien de la cohérence

Nombre copie pour E ou L	**Une** PC PL		**0<n<toutes** I PS PP PF PRI PRP PE		**Toutes**			
Droits de mise à jour	**Un maitre**		**Plusieurs maitres**		**N'importe quelle copie** PC PP PL PF PRI PRP PE			
	Dynamique I	**Statique** PS	**Dynamique**	**Statique**				
Moment de la synchronisation	**Immédiate** PC PP	**Différée** I PS PC PP PL PE						
		Délai	**Périodicité**	**Moment** PF PRI PRP	**Version**	**Numérique**	**Objets**	**Evénements**
Initiative de la mise à jour	**Push** PC PP PL PF PRI PE			**Pull** I PS PF PRP				
Nature des mises à jour	**Valeur** PF PRI PE I PS PC PP PL		**Opération**		**Invalidation** I PS			
Topologie	**Etoile**	**Copie en Copie** I PS PC PP PL PF PRI PRP PE	**Groupe en Groupe**		**Chemin particulier**			
Détection des mises à jour	**Journal**	**Copie ombre**	**Trigger**	**API**	**Lecture de la copie**			
Conflits	**Oui**			**Non** I PS				
couche com.	**Oui** FR PC PP PRI PRP PE			**Non** I PL PF PRI PRP PE				
Concurrence	**n L ou 1 E** I PS PE	**n L ou n E**		**n L et 1 E**	**n L et n E** PRI PRP PC PP PL PF			
Fautes	**Oui**			**Non**				

I : Ivy FR : Fast Read PS : protocole de cohérence séquentielle PC : protocole de cohérence causale
PP : protocole de cohérence PRAM PL : protocole de cohérence à la longue
PRI : protocole de cohérence au relachemet impatient PRP : protocole de cohérence au relachement paresseux
PF : protocole de cohérence faible PE : protocole de cohérence à l'entrée

FIG. 2.14 – Classification des protocoles de duplication utilisés dans les MVP

se dégagent. Le premier niveau concerne le protocole assurant la cohérence sur les accès aux objets du SGBDR ou de la mémoire partagée répartie. Cette cohérence n'est pas spécifique à la duplication. Le deuxième niveau est le protocole de duplication assurant la cohérence entre les différentes copies d'un même objet.

Il est beaucoup plus facile pour un philosophe d'expliquer un nouveau concept à un autre philosophe qu'à un enfant. Pourquoi ? Parce que l'enfant pose les vraies questions.

Jean-Paul Sartre - Extrait d'une interview dans Le monde - Octobre 1971

Chapitre 3

Supports de duplication adaptables

Sommaire

Le chapitre précédent nous a montré que la littérature abonde en protocoles et techniques de duplication. Selon l'application, le contexte et l'objectif recherché, tel ou tel protocole se trouve être plus ou moins approprié. De plus, une fois le protocole choisi, sa mise en œuvre est une tâche particulièrement difficile pour le programmeur d'application. Ainsi, de nombreux travaux visent à le décharger des considérations propres à la duplication en offrant un support de la duplication. Notre propos dans ce chapitre est de comprendre comment les travaux existants proposent de rendre ce support adaptable et d'être critique par rapport à ceux-ci.

Ce chapitre est organisé de la manière suivante. Nous commençons par délimiter notre cadre de travail (section 3.1), puis nous présentons des travaux offrant une certaine adaptabilité en terme de duplication. Ces travaux sont issus du domaine des mémoires partagées réparties (section 3.2), des systèmes à objets répartis (section 3.3) et du contexte CORBA (section 3.4). Nous terminons ce chapitre par une conclusion (section 3.5).

3.1 De la nécessité de l'adaptabilité pour être adaptable

Dans cette section, nous commençons par définir ce que nous entendons par adaptable, adaptabilité et adaptation (section 3.1.1), puis nous présentons brièvement la façon dont un

support adaptable peut être mis en œuvre (section 3.1.2).

3.1.1 Adaptable, adaptation et adaptabilité

Offrir un support de duplication adaptable semble être une bonne intention vu la difficulté de construire un protocole de duplication. Cependant, la notion d'adaptabilité dans les systèmes informatiques est un problème très vaste que nous nous proposons de délimiter dans cette section.

Adaptable : Afin de mieux comprendre le cadre de notre travail, commençons par définir ce que nous entendons par adaptable :

Définition 3.1 : *Adaptable adj.*
Qui peut être adapté [Lar89].

Définition 3.2 : *Adapter v.*
Rendre (un dispositif, des mesures, etc.) apte à assurer ses fonctions dans des conditions particulières ou nouvelles [Lar89].

D'après ces définitions, un support de la duplication est adaptable si les fonctions internes déjà présentes peuvent être utilisées dans des conditions particulières, mais s'il est également possible de modifier son comportement afin de supporter de nouveaux besoins. Nous considérons qu'un support de la duplication peut être adaptable :

- **A l'application (au code fonctionnel).** Le support de la duplication défini peut être utilisé avec différentes applications.
- **Au contexte non fonctionnel.** Le support peut être utilisé avec différents aspects non fonctionnels (répartition, concurrence, tolérance aux fautes, transactionnel, etc.)
- **Dans tout ou partie des protocoles de duplication.** Il est possible de rajouter/changer/modifier des fonctionnalités du support de la duplication afin de prendre en compte de nouveaux protocoles.

Adaptation : Un autre terme apparaît lorsque l'on parle de support adaptable : l'adaptation.

Définition 3.3 : *Adaptation n. f.*
Action d'adapter [Lar89].

L'adaptation est le processus de modification nécessaire pour permettre son fonctionnement adéquat dans un contexte donné. Adéquat signifie que le support correspond parfaitement à ce que l'on attend de lui dans ce contexte précis. L'adaptation se décompose en trois étapes successives : le déclenchement, la décision et la réalisation [ARC01]. Le rôle de l'étape de déclenchement est de détecter et de notifier d'un changement. Lors de l'étape de décision, les modifications devant être effectuées pour réagir aux changements sont déterminées. L'étape de réalisation concerne tous les moyens mis en œuvre pour appliquer la décision prise lors de l'étape précédente.

Certains travaux parlent de systèmes adaptatifs et mettent en œuvre l'adaptation (adaptatif est un terme du domaine biologique signifiant qui réalise une adaptation [Lar89]).

Adaptabilité : Afin d'obtenir un support de la duplication adaptable, il est nécessaire que celui-ci présente une certaine adaptabilité.

Définition 3.4 *: Adaptabilité n.f.*
Caractère de ce qui est adaptable [Lar89].

Cette notion fait référence aux qualités inhérentes du système. Pour mettre en œuvre l'adaptation du support de la duplication, il est nécessaire que celui-ci soit adaptable. Ce support est adaptable s'il offre l'adaptabilité. Dans ce chapitre et dans notre thèse, nous ne nous intéressons pas à la dimension adaptation mais à l'adaptabilité, c'est à dire comment mettre en œuvre l'adaptabilité afin de pouvoir offrir un support adaptable pouvant ensuite servir à l'adaptation.

3.1.2 Approches pour obtenir l'adaptabilité

Deux grandes approches existent pour mettre en œuvre l'adaptabilité : l'approche par paramétrisation et l'approche modulaire.

Approche par paramétrisation : L'approche par paramétrisation consiste à définir un protocole générique offrant un ensemble de paramètres afin d'être adapté en fonction des besoins. Cependant, il semble impossible de fournir un protocole de duplication générique pouvant être paramètré afin de supporter les différents protocoles de duplication existants et pouvant s'adapter à différents contextes d'exécution. Cette approche nous paraît également difficilement utilisable si l'on désire que le support de la duplication puisse supporter de nouveaux besoins non prévus. De plus augmenter le nombre de paramètres augmente dangereusement la complexité du code. Ainsi pour mettre en œuvre l'adaptabilité une approche modulaire semble plus appropriée.

Approche modulaire : Les supports modulaires, à l'inverse des supports monolithiques, présentent une architecture qui permet leur modification en vue de répondre à des besoins particuliers. Dans un système modulaire, il est possible de modifier ou de remplacer certaines parties du système avec un minimum d'interférences avec le reste du système.

La programmation orientée objets encourage la réutilisation notamment grâce à l'encapsulation et l'héritage. Elle offre ainsi la possibilité d'interchanger des objets ayant la même signature et d'adapter des objets existants à de nouveaux besoins. Néanmoins, les lignes de code gérant les aspects non fonctionnels se trouvent dans les méthodes spécifiques au code fonctionnel. Ce problème est appelé intercalage (*interleaving*). De plus ce code se trouve dans plusieurs méthodes. Cet autre problème est appelé délocalisation.

Ainsi, la séparation de considérations [LBS01] (*separation of concern*) propose davantage que la programmation orienté objets en matière de réutilisation. Elle consiste à découper les applications en modules traitant chacun d'un aspect de l'application.

Une fois le système décomposé sous forme de modules, il est nécessaire de les assembler. La composabilité est la capacité d'assemblage entre les éléments constituants du système. De nombreux travaux sont en cours, approches à base de composants ou d'aspects, mettant l'accent sur cette problématique. Ils s'orientent par exemple, sur la réification des connexions entre modules [MT00] ou définissent des langages de tissage de modules [KLM+97].

Dans ce chapitre, nous nous intéressons à l'approche par paramétrisation et l'approche modulaire. Cependant, nous ne nous penchons pas sur la partie composabilité qui est un problème plus large. Notre objectif est de décomposer la duplication afin d'offrir un support pour les travaux consacrés à la composabilité.

3.2 Adaptabilité dans les mémoires partagées réparties

Cette section présente deux travaux abordant le problème de l'adaptabilité dans le support de la duplication dans le contexte des mémoires partagées réparties. Nous avons choisi Munin (section 3.2.1) pour son approche par paramétrisation et un canevas flexible pour la gestion de la cohérence dans les mémoires partagées réparties orienté objet (FFCM) (section 3.2.2) pour son approche modulaire.

3.2.1 Munin

Munin [CBZ91] offre aux développeurs d'applications un ensemble de protocoles de cohérence mettant en œuvre le modèle de cohérence au relâchement (voir section 2.4.3.2).

Choix possibles pour le protocole de cohérence : Chaque objet est annoté par le programmeur pour spécifier le protocole de cohérence qui doit le gérer. Sept annotations

Annotation	Paramètres							
	I	R	D	FO	M	S	Fl	W
Lecture seule	N	O						N
Migratoire	O	N		N	N		N	O
Ecriture partagée	N	O	O	N	O	N	N	O
Prod. Cons.	N	O	O	N	O	O	N	O
Réduction	N	O	N	O	N		N	O
Résultat	N	O	O	O	O		O	O
Conventionnel	O	O	N	N	N		N	O

FIG. 3.1 – Correspondance annotations/paramètres des protocoles Munin

sont proposées. Les objets **en lecture seule** une fois initialisés ne peuvent plus être mis à jour. Le protocole de cohérence se contente de créer des copies à la demande. Pour les objets **migrateurs**, un unique processus peut accéder à l'objet, et faire une ou plusieurs écritures, avant qu'un autre processus puisse accéder l'objet. Lorsqu'un processus veut écrire sur l'objet, le protocole migre la copie vers le nouveau processus et invalide l'ancienne. Les objets **en écriture partagée** peuvent être écrits simultanément par plusieurs processus, sans que les copies soient synchronisées. Ces objets sont utiles si le programmeur sait que les mises à jour concernent des parties différentes de l'objet. Les objets **producteur consommateur** sont écrits par un processus et lus par un ou plusieurs processus. Les objets **réduction** sont accédés par des opérations du type acquisition d'un verrou sur l'objet, lecture, écriture, puis relâchement du verrou. Les objets **résultat** sont accédés suivant deux phases : lors d'une première phase, ils sont modifiés en parallèle par plusieurs processus, puis lors de la seconde un unique processus les accède de manière exclusive. Les objets **conventionnels** sont dupliqués à la demande et un écrivain est le seul possesseur de l'objet garantissant ainsi la cohérence de l'objet.

Paramètres des protocoles de cohérence : Pour mettre en œuvre les protocoles décrits par les annotations, huit paramètres sont proposés :

- **Invalidation ou mise à jour (I)** : Spécifie si les changements sur un objet doivent être propagés par invalidation ou par mise à jour des copies distantes.
- **Copies permises (R)** : Spécifie si plus d'une copie d'un objet peut exister.
- **Opérations retardées permises (D)** : Spécifie si les mises à jour ou les invalidations peuvent être retardées.
- **Propriétaire fixe (FO)** : Indique de ne pas propager la maîtrise d'un objet. Toute copie peut être accédée en lecture, mais les écritures doivent être envoyées au propriétaire.
- **Plusieurs écrivains permis (M)** : Spécifie si l'objet peut être modifié de manière concurrente.
- **Accès partagé fixe (S)** : Indique si un objet doit toujours être accédé par les mêmes processus durant toute l'exécution du programme.

- **Envoie des modifications au propriétaire (FI)** : Indique qu'un processus doit propager ses modifications en les envoyant uniquement au propriétaire de l'objet (ensuite il invalide sa copie).
- **Ecriture permise (W)** : Spécifie que l'objet partagé peut être modifié.

Aux différentes annotations présentées correspondent différentes valeurs pour les paramètres comme décrit dans la figure 3.1 (O=oui, N=non).

Adaptabilité : Munin est adaptable par rapport à l'application en séparant le code fonctionnel du code non fonctionnel. L'aspect duplication n'est pas adaptable au contexte non fonctionnel : le protocole de cohérence est plus qu'un protocole de duplication. Il assure à la fois la cohérence entre les objets du système (le modèle de cohérence au relâchement) et entre les copies d'un même objet, en gérant à la fois la concurrence, la mise en cohérence des copies et la création/destruction/migration des copies. Il n'y a pas d'isolation de l'aspect duplication. Finalement, l'adaptabilité à l'intérieur du protocole de cohérence est obtenue par paramétrisation engendrant un nombre limité de protocoles supportés.

3.2.2 Un canevas flexible pour la gestion de la cohérence dans les MPR

[WNT98] propose un canevas flexible pour la gestion de la cohérence dans les mémoires réparties orientées objets. Afin d'obtenir l'adaptabilité, les auteurs s'attachent d'une part à séparer modèle de cohérence et protocole de cohérence et d'autre part à avoir une approche modulaire. Ainsi, les deux principaux composants de leur canevas sont le modèle de cohérence et le protocole de cohérence. Ces composants sont assistés par un composant pour la communication et un pour le contrôle de concurrence (figures 3.2 (a) et (b)).

Le composant modèle de cohérence (*consistency model*) : Il est constitué des composants protocole de cohérence et gestion de la concurrence. De plus, les auteurs lui donnent quatre tâches :

- **L'ordre des accès.** Cette propriété définit l'ordre dans lequel sont vus les accès par les différentes parties. Cet ordre peut être vu selon deux points de vue : l'ordre dans lequel le nœud origine voit ses propres opérations, ou l'ordre dans lequel les autres nœuds voient les opérations. Cette propriété est le cœur du modèle de cohérence.
- **La concurrence.** La concurrence sur les accès définit si les nœuds peuvent de manière concurrente accéder aux données. Généralement les opérations de lecture peuvent être concurrentes et les opérations d'écriture le peuvent sous certaines conditions. Quatre combinaisons d'accès lecture/écriture sont possibles : EREW (lecture exclusive, écriture exclusive), CREW (lectures concurrentes, écriture exclusive), ERCW (lecture exclusive, écritures concurrentes) et CRCW (lectures concurrentes, écritures concurrentes).
- **La portée.** La portée détermine l'ensemble des données devant rester cohérentes.

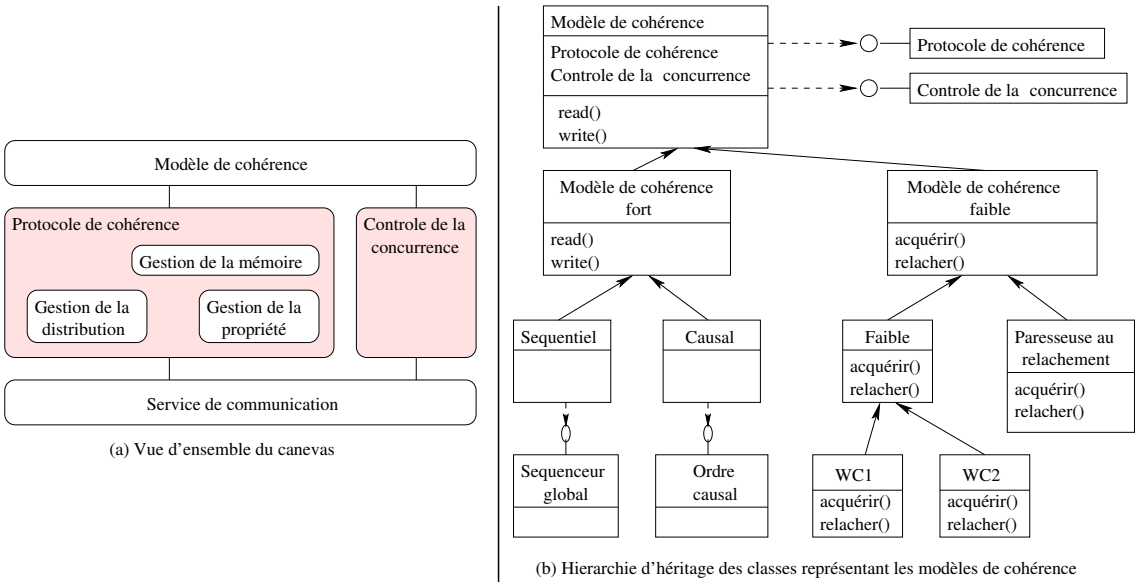


FIG. 3.2 – Vue d'ensemble et hiérarchie de classes du canevas FFCM

– **L'atomicité.** Cette propriété définit si la propagation des mises à jour est faite à chaque accès ou lorsque plusieurs mises à jour locales sont faites.

Le composant modèle de cohérence offre quatre opérations afin de prendre en compte les deux familles de modèles de cohérence (fort et faible). Les modèles forts ordonnent chaque opération d'accès et les faibles un ensemble d'opérations. Ainsi pour les modèles forts, seules les opérations `lire()` et `écrire()` sont utilisées. Elles sont redéfinies afin de prendre en compte le contrôle de concurrence. Dans les modèles faibles, les opérations `acquerir()` et `relacher()` sont introduites afin de permettre à l'application d'annoter le début et la fin des ensembles d'opérations d'accès.

Le composant protocole de cohérence (*coherency protocol*) : Un protocole de cohérence s'occupe de la mise en cohérence des copies. Deux types d'événements sont gérés par le protocole : les événements d'accès et les événements de synchronisation. Un événement d'accès est une opération de lecture ou d'écriture d'une copie locale. Un événement de synchronisation est une requête pour synchroniser toutes les copies d'une donnée. Un protocole de cohérence se compose d'un :

– **Composant de gestion de la mémoire.** Ce composant contrôle l'accès aux copies locales des données partagées. Ces accès se répartissent en : lecture ou écriture provenant des application locales et incorporation de mise à jour.

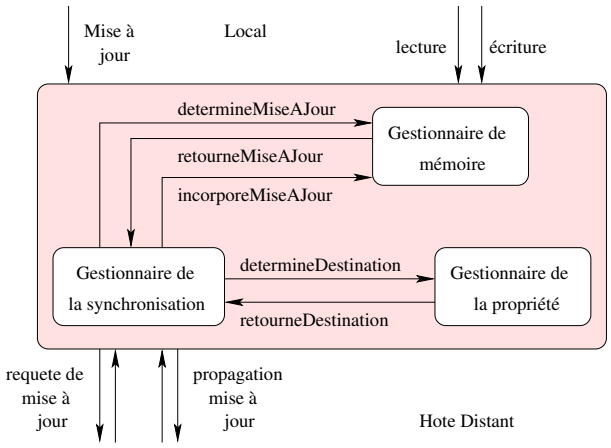


FIG. 3.3 – Vue d'ensemble du protocole de cohérence dans FFCM

- **Composant de gestion de la propriété.** Ce composant garde des informations sur le propriétaire courant de la donnée partagée. Différentes algorithmes sont proposés dans la littérature : gestion statique et centralisée, basée sur des répertoires, etc.
- **Composant de gestion de la synchronisation.** Ce composant génère les événements de synchronisation : propagation des mises à jour (approche push) ou requêtes de mise à jour (approche pull).

Le modèle de cohérence dispose de deux opérations pour diriger le protocole de cohérence (figure 3.3) : `forcerMiseAJour` (propagation des mises à jour) et `assurerCohérence` (demande des mises à jour). L'opération `forcerMiseAJour` fonctionne de la manière suivante : (a) elle détermine ce qui a besoin d'être mise a jour, (b) puis demande au gestionnaire de la propriété la destination des mises à jour et enfin (c) la mise à jour est envoyée vers les destinataires par le gestionnaire de synchronisation. L'opération `assurerCoherence` fonctionne de la manière suivante : (a) elle demande au gestionnaire de la propriété la source où se trouve les mises à jour, (b) puis demande une mise à jour en utilisant l'opération `requestUpdate` du composant de synchronisation et (c) une fois la mise à jour retournée elle est incorporée dans la copie locale par le gestionnaire de mémoire.

Adaptabilité : FFCM est adaptable par rapport à l'application en séparant le code fonctionnel du code non fonctionnel. Ils s'attachent également à isoler l'aspect duplication des autres aspects non fonctionnels. La séparation et la définition des interactions entre protocole de cohérence (gérant la mise en cohérence des copies) et modèle de cohérence (gérant l'ordre d'accès entre tous les objets) rendent l'aspect duplication adaptable au contexte non fonctionnel[1]. Le protocole de cohérence ne traite que ce qui est propre à la duplication, et un

[1]Selon nous, il y a une légère confusion sur le terme modèle. En effet un modèle n'est qu'une spécification plus ou moins formelle du comportement de la mémoire et non une implantation.

même protocole peut être adapté pour mettre en œuvre divers modèles de cohérence (causale, séquentielle, au relâchement, etc.). Cependant, FFCM reste limité au contexte des mémoires partagées réparties. L'adaptabilité à l'intérieur du protocole est obtenue par modularité. Les trois composants proposés (gestionnaire de la mémoire, de la propriété et de la synchronisation) peuvent être implantés de diverses façons selon les besoins. Néanmoins, le nombre de composants nous paraît assez restreint.

3.3 Adaptabilité dans les systèmes à objets répartis

Les systèmes à objets répartis se sont attachés à offrir des supports adaptables de la duplication. L'approche choisie est la séparation du code applicatif du code gérant la duplication. L'idée est d'avoir deux niveaux de programmation : le niveau fonctionnel concernant la partie purement applicative et le niveau non fonctionnel concernant les aspects système (répartition, duplication, persistance, tolérance aux fautes, etc.). Cette approche permet d'obtenir la modularité et la réutilisabilité et ainsi d'obtenir l'adaptabilité.

Dans cette section, nous présentons trois travaux : GARF (section 3.3.1) pour présenter les concepts généraux de programmation à deux niveaux, Core (section 3.3.2) pour son adaptabilité à l'application et au contexte non-fonctionnel et Globe (section 3.3.3) pour son adaptabilité d'une partie du support des protocoles.

3.3.1 GARF

GARF [GGM95] est une plateforme orientée objet facilitant le développement d'applications réparties tolérantes aux fautes. Cette dernière est mise en œuvre par duplication assurant une cohérence forte entre les copies (duplication active et passive).

Garf est représentatif de ce qui se fait dans les travaux sur la séparation des considérations pour mettre en œuvre la duplication passive et active (FRIENDS [FP98], MAUD [AS94], RepliXa [KG96]). Chaque objet réparti est séparé en deux objets indépendants, un **objet applicatif** et un **objet de comportement**. L'objet de comportement intercepte toutes les invocations reçues ou envoyées à son objet applicatif associé de manière transparente. Il est constitué de deux objets : un **encapsulateur** et un **mailer**. Un encapsulateur est présent sur chaque site où se trouve une copie de l'objet applicatif qu'il gère. Une copie du mailer est présente sur tous les sites pouvant contenir un objet désirant invoquer son objet applicatif.

Création d'objet : L'association objet de comportement/objet applicatif est faite à la création de l'objet applicatif. La méthode `new` est réifiée par GARF en la méthode `garfNew` créant l'objet applicatif, un encapsulateur et un mailer, les liant entre eux et renvoyant un mailer. Le choix (dans une librairie) de l'encapsulateur et du mailer détermine le comportement de toutes les instances de l'objet applicatif.

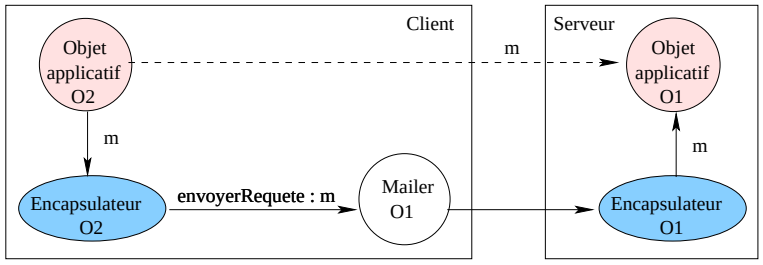


FIG. 3.4 – Invocation d'une méthode dans Garf

Invocation : Quand un objet réparti invoque un autre objet, seul son encapsulateur participe. Par contre s'il est invoqué son encapsulateur et son mailer participent. Une invocation d'une méthode m sur l'objet O_1 par un objet O_2 (figure 3.4) est transformée par GARF en une invocation sur l'encapsulateur de O_2. Ce dernier reçoit les arguments suivants : la méthode réifiée m et un clone du mailer de l'objet O_1. L'encapsulateur de O_2 transmet l'invocation réifiée au mailer de O_1 en appelant la méthode `sendRequest`. Le mailer appelle alors `inRequest` sur l'encapsulateur de O_1, pour faire suivre la méthode m sur le site de O_1. L'encapsulateur de O_1 invoque la méthode m sur l'objet applicatif O_1. Le résultat revient à O_2 en prenant le chemin inverse. Les mailers sont implantés en utilisant Isis (chapitre 2.2.2.1).

Duplication active dans GARF : La duplication active est mise en œuvre de la manière suivante[2] : la gestion du groupe de copies est fournie par l'encapsulateur (instance de `ActiveReplica`) et la communication entre les membres du groupe par le mailer (instance de `Abcast`). Le mailer `AbCast` s'appuie sur le multicast fiable fourni par Isis.

A la création de l'objet dupliqué, la méthode `garfNew` commence par créer un encapsulateur sur chaque site devant contenir une copie. Ensuite elle demande à chaque encapsulateur de construire une instance de l'objet applicatif. Un mailer est ensuite créé avec le groupe d'encapsulateur fourni en paramètre. Ce mailer est finalement retourné à l'appelant de `garfNew`.

Quand l'encapsulateur d'un objet s'adressant à l'objet dupliqué invoque le mailer, ce dernier diffuse la méthode invoquée au groupe, c'est à dire à chaque encapsulateur. Ceux-ci invoquent la copie locale et retourne le résultat au mailer. Le mailer regroupe toutes les réponses des différents encapsulateurs, et renvoie la première réponse à l'encapsulateur appelant.

Adaptabilité : Ce travail offre l'adaptabilité par rapport à l'application en permettant deux niveaux de programmation. Néanmoins, comme de nombreux travaux sur la séparation des considérations, la séparation des aspects non fonctionnels est limitée ou inexistante. L'ensemble du code gérant la duplication et la concurrence se trouve dans les encapsulateurs

[2]Le même principe est utilisé pour mettre en œuvre la duplication passive

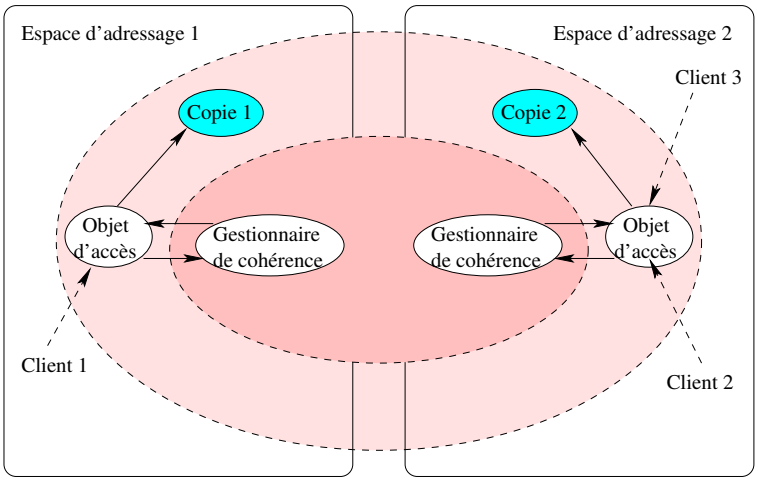


FIG. 3.5 – Architecture Core

et l'ordonnancement des accès sur les différentes copies et sur les différents objets se trouve pris en charge par le protocole de communication. Ainsi, aucune adaptabilité au contexte non fonctionnel de l'aspect duplication n'est proposée. L'adaptabilité dans tout ou partie des protocoles n'est pas abordée.

3.3.2 Core

Core [BCM95, BC98] est une architecture et un environnement d'exécution pour objets dupliqués adaptables. Les auteurs séparent ce qu'ils appellent la **logique spécifique** au type de l'objet dupliqué et la **logique générique**. La logique spécifique concerne l'information relative au type de l'objet pouvant être exploitée pour augmenter les performances . La logique générique concerne la gestion de la cohérence.

Exploitation de l'information spécifique au type de l'objet : Afin d'augmenter les performances, Core contrôle la concurrence à un grain fin en exploitant la sémantique de l'objet. Pour cela, une intention caractérise une activité (un ensemble d'accès limité dans le temps) que le client compte faire sur un objet dupliqué. Elle identifie un client, un domaine d'activité (les parties logiques de l'objet qui sont concernées par les intentions d'accès) et un ensemble de types d'action que le client veut faire sur le domaine spécifié. Une intention généralise un verrou. Le gestionnaire de cohérence peut utiliser l'information donnée par les intentions afin d'augmenter les performances en relâchant l'ordonnancement des accès en écritures sur les différentes copies.

Gestion de la cohérence : Un objet dupliqué est constitué de trois types d'entités (figure 3.5) : les copies, les objets d'accès et le gestionnaire de cohérence réparti. Les deux premiers composants sont spécifiques au type de l'objet alors que le gestionnaire de cohérence est générique. Il peut donc être réutilisé.

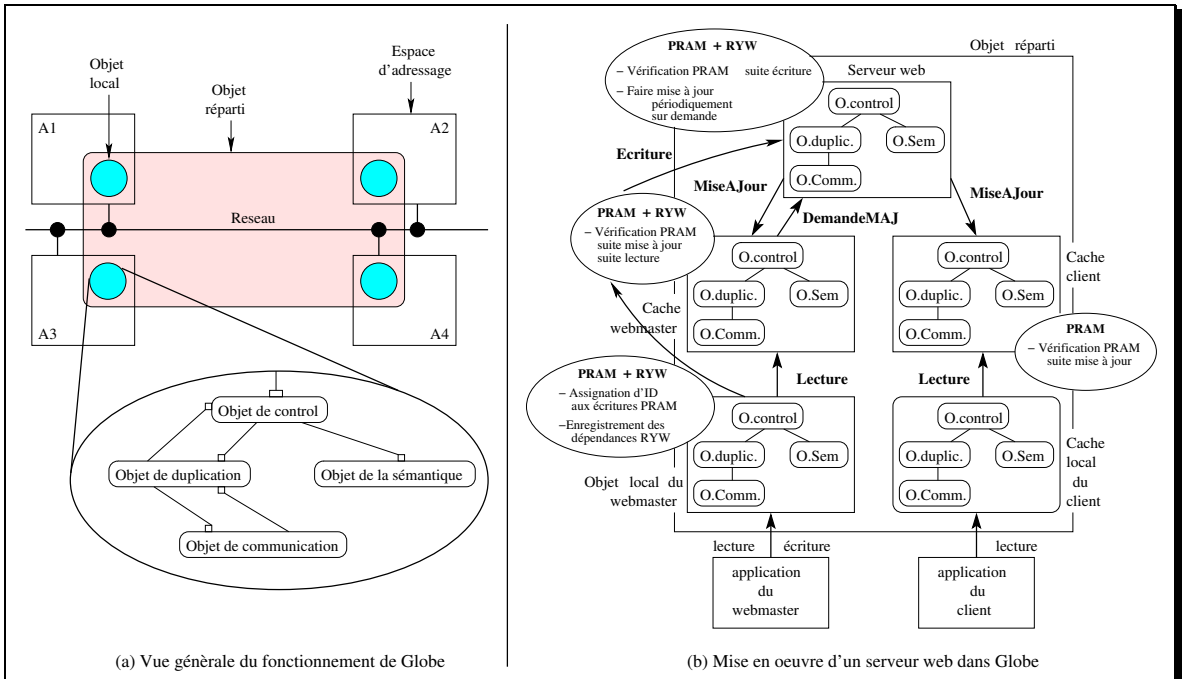
Une **copie** encapsule les données des objets dupliquées. Elle ne gère pas la concurrence ou la duplication. Elle fourni une interface pour manipuler l'état dupliqué.

Un **objet d'accès** encapsule la copie locale. Il assure le protocole d'accès suivant : il commence par acquérir un verrou auprès du gestionnaire de cohérence, puis invoque la copie locale notifiant les modifications au gestionnaire de cohérence et finalement relâche le verrou. Chaque objet d'accès a deux interfaces : une interface manipulée par les objets locaux pour accéder à l'objet dupliqué et une interface utilisée par le gestionnaire de cohérence pour rapporter les modifications distantes. L'interface client est divisée en deux parties : une interface reprenant celle de l'objet dupliqué et une interface de contrôle de concurrence. Cette dernière permet à un objet s'adressant à l'objet dupliqué de protéger une séquence d'invocations sous une seule intention.

Le **gestionnaire de cohérence réparti** implante le protocole de cohérence. Il prend les verrous et propage les mises à jour. Il détient une référence vers l'objet d'accès et vers l'objet dupliqué associé. Il présente l'interface suivante :

- L'opération `beginActivity` permet de déclarer les actions prévues par une nouvelle activité et demande la permission de démarrer cette activité. Un appel à cette méthode est bloquant jusqu'à ce que l'activité entière soit permise. Le gestionnaire de cohérence détermine le groupe de gestionnaires de cohérence devant participer à la décision d'autoriser l'activité. Deux types d'activité existent. Une activité faible n'est pas en conflit avec une autre et l'ordre sur les différentes copies n'est pas important. Une activité forte peut être en conflit avec n'importe quelle activité forte ou faible, les différents gestionnaires de cohérence doivent donc vérifier leurs requêtes locales afin de se mettre d'accord sur l'ordre.
- L'opération `updates` prévient le gestionnaire de cohérence que la copie locale a été modifiée. Un premier argument identifie l'activité qui a fait les modifications, le second décrit ces modifications. Le gestionnaire de cohérence propage les mises à jour à tous les objets d'accès. Le protocole de communication est dépendant du contrat de cohérence.
- `endActivity` notifie de la terminaison d'une activité.

Adaptabilité : Comme pour GARF, les deux niveaux de programmation offrent l'adaptabilité par rapport à l'application en séparant le code fonctionnel du code non fonctionnel. De plus, Core prend en compte les spécificités de l'application afin d'adapter le protocole à l'application en relâchant le contrôle de concurrence. Core offre une certaine adaptabilité par rapport au contexte non fonctionnel. Il est possible de l'utiliser dans un contexte transactionnel ou de système à objets répartis. Cependant, cette adaptabilité n'est pas celle de l'aspect duplication. Core est le noyau minimum d'un gestionnaire de gestion de la cohérence des



(a) Vue génèrale du fonctionnement de Globe

(b) Mise en oeuvre d'un serveur web dans Globe

FIG. 3.6 – Concepts de Globe

objets du système qui prend en compte l'aspect duplication. L'adaptabilité dans le protocole n'est pas abordée.

3.3.3 Globe

Globe [vSHT97, vSHT99] est une architecture pour la construction d'applications objets à large échelle. C'est un intergiciel sur des réseaux et des systèmes d'exploitation existants. Il traite de domaines comme la communication, le nommage, la duplication, la migration, la persistance, la tolérance aux fautes et la sécurité.

Principes : Les objets partagés et répartis intègrent leur politique de duplication et de migration. Ces objets sont constitués de plusieurs objets locaux. Un objet local est un objet résidant dans un seul espace d'adressage et pouvant communiquer avec d'autres objets locaux (se trouvant dans d'autres espaces d'adressage). Chaque objet local formant une partie d'un objet partagé et réparti est composé au minimum des quatre objets locaux suivants (figure 3.6 (a)) :

– L'**objet sémantique** est l'objet applicatif. C'est le seul objet que le développeur doit programmer en fournissant les différentes interfaces et une implantation dans un langage.

Les autres objets peuvent être tirés de librairies ou générés à partir de spécifications.
- L'**objet de communication** est responsable des communications entre les différents objets locaux. Généralement c'est un objet fourni par la machine. Il peut implanter des communications point à point, des diffusions ou les deux.
- L'**objet de duplication** doit assurer le modèle de cohérence sur l'état global de l'objet partagé et réparti qui est constitué d'états de plusieurs objets locaux.
- L'**objet de contrôle** prend en charge les requêtes (emballage, déballage, etc.), et contrôle les interactions entre l'objet sémantique et l'objet de duplication.

Gestion de la duplication : [KKvST97, KKvST98] décrit une implantation pour maintenir la cohérence de documents Web dupliqués en utilisant l'architecture Globe. Le document Web est vu comme l'objet partagé réparti et la collection de pages, images, applets, etc. forme l'état de cet objet. Les fichiers constituant l'état de l'objet peuvent être stockés dans trois sortes d'espace de stockage :

- Les espaces de stockage **permanents** sont persistant. Ils maintiennent une certaine cohérence entre les objets Web. Un serveur Web est un exemple d'espace de stockage permanent.
- Les espaces de stockage **initiés par les objets** offrent une cohérence plus faible que celle offerte par les espaces de stockage permanent. Un exemple de ce type d'espace de stockage est un serveur miroir.
- Les espaces de stockage **initiés par les clients** dépendent des processus clients qui lisent et écrivent l'état de l'objet. Ils sont comparables à des caches. Un exemple de ce type d'espace de stockage est un proxy Web.

Différents niveaux de cohérence entre ces trois types d'espace de stockage peuvent être utilisés. Les auteurs font une distinction entre la cohérence offerte par un objet dupliqué (modèle de cohérence des objets) et la cohérence demandé par un client (modèle de cohérence du client). Le **modèle de cohérence des objets** peut être séquentiel, PRAM, causal ou à la longue. Il représente la vue du développeur. Après avoir décidé du modèle en fonction du support de stockage, le programmeur choisi le protocole. Les **modèles de cohérence client** expriment les préférences d'un seul client. Ils reprennent les garanties de sessions définies dans Bayou [TDP+94]. Une session est une séquence d'opérations de lecture et d'écriture effectuées durant l'exécution d'une application. Des garanties sont proposées aux applications sur les résultats de leurs opérations.

- "Lire ses écritures" (Read your writes) garantie à une application qu'une opération d'écriture d'une session sera visible par toutes les opérations de lecture ultérieures dans la même session. Elle ne garantie pas qu'une opération de lecture retournera forcément la valeur de sa dernière écriture sur la même donnée, puisqu'une opération d'écriture d'une autre session aura très bien pu changer cette valeur.
- "Lectures monotones" (Monotonic reads) garantie à un utilisateur d'observer le serveur d'une façon continue, ie sans régression dans les mises à jour.
- "Les écritures suivent les lectures" (Writes follow reads) s'apparente à un ordre causal.

Paramètre	*Valeurs*
Propagation des mise à jour	- mise à jour - invalidation
Espace de stockage	- permanent - permanent et initié par les objets - tous
Ensemble d'écrivain	- un seul - plusieurs
Initiative du transfert	- pull - push
Instant du transfert	- impatiente - paresseuse
Type d'accès	- partiel - complet
Type du transfert	- notification - partiel - complet

FIG. 3.7 – Paramètres Globe pour les protocoles de cohérence

– "Ecritures monotones" (Monotonic writes) garantie à un client que les opérations d'écriture seront exécutées sur toutes les copies dans leur ordre d'émission. C'est un modèle de cohérence PRAM mais basé sur un unique client.

Paramètres des protocoles : De plus pour chaque protocole de cohérence un ensemble de paramètres est proposé afin de préciser quand, comment et par qui la cohérence est gérée. Ces paramètres sont fixés par le programmeur des objets une fois le modèle de cohérence des objets choisi. Le tableau 3.7 reprend l'ensemble de ces paramètres. Deux autres paramètres pour les modèles des objets et des clients existent : attendre ou demander une mise à jour.

Adaptabilité : On retrouve dans ces travaux la séparation code fonctionnel code non fonctionnel. Les modèles de cohérence associés aux objets dupliqués dans Globe (séquentiel, causal, PRAM, etc.) sont des garanties offertes aux utilisateurs sur la perception qu'ils ont sur les différentes copies d'un objet et non sur l'ensemble des objets (voir section 2.4). Ainsi, Globe n'offre pas l'adaptabilité au contexte non fonctionnel. L'adaptabilité dans le support des protocoles, obtenue par paramétrisation, en limite grandement la portée.

3.4 Adaptabilité dans le contexte CORBA

Il n'existe pas de spécification CORBA [OMG99] (*Common Object Request Brocker Architecture*) sur la duplication. Cependant, l'OMG a spécifié la tolérance aux fautes (approche par service et par intégration) [OMG01]. Celle-ci se fait par duplication (avec cohérence forte

entre les copies), détection des fautes et par récupération. Les protocoles de duplication mis en œuvre s'appuient sur des systèmes de communication de groupe. Différentes approches existent [FGS00] : par intégration, par interception et par service.

La toute première approche pour associer les groupes de communication à l'environnement CORBA fut par **intégration** : un système de communication de groupe existant est intégré dans l'ORB. Le principal défaut de cette approche est qu'il est nécessaire de modifier l'implantation de l'ORB (*Object Request Broker*). En effet l'ORB distingue une référence à un objet d'une référence à un groupe. Lorsqu'un client s'adresse à un groupe, la requête est transmise à un système de communication de groupe, puis côté serveur, l'invocation est de nouveau transformée en un appel standard sur chaque membre du groupe. Orbix+Isis [II94], basé sur Orbix [ION95] et sur Isis [BR94], Electra [Maf95] utilisant Isis ou Horus [RBG+95] pour la communication de groupe ou Cobra [BPR97] adoptent cette approche. Cette approche va à l'encontre de la spécification CORBA précisant qu'une référence doit correspondre à un seul objet et qu'elle doit désigner le même objet à chaque utilisation. De plus, cette approche est dépendante de l'ORB utilisé. Néanmoins, cette approche est la plus efficace.

Dans l'approche par **interception**, les messages issus de l'ORB, par exemple les requêtes IIOP[3] (*Internet Inter-Object Protocol*), sont interceptés et mappés sur une boîte à outils de communication de groupe. Cette approche est indépendante de la mise en œuvre des clients et de l'ORB. Elle peut être utilisée avec n'importe quelle implantation d'ORB respectant IIOP. Eternal (section 3.4.1) met en œuvre cette approche.

Avec la dernière approche, l'approche par **service**, la communication de groupe est fournie comme un service CORBA [OMG97] (au dessus de l'ORB). Le service présente une interface que le client doit utiliser de façon explicite. Il n'y a pas de transparence à moins d'utiliser, par exemple, les mandataires intelligents [FGS97] (*smart proxy*), mais au détriment de la portabilité. Avec l'approche par service, le service est libre d'utiliser un système de communication spécifique ou les primitives de communication normalisées par CORBA. Il est également possible de normaliser l'interface d'un service de communication de groupe ou de duplication d'objets non dépendant d'une mise en œuvre particulière. Cette solution, préconisée par CORBA pour d'autres aspects non fonctionnels et facilitant la modularité et la réutilisabilité, est la moins efficace : la requête passe du client à l'ORB local, puis au service, de nouveau sur l'ORB local pour être propagée à un objet distant faisant suivre la requête au serveur par l'intermédiaire de l'ORB distant. OGS (section 3.4.2) adopte cette approche.

3.4.1 Eternal

Le système Eternal [PNMS97, MMSN98, PNMS02] permet de mettre en œuvre la tolérance aux fautes dans un environnement CORBA. Les objets peuvent être dupliqués de manière passive ou active en s'appuyant sur le système de communication de groupe Totem [MMSA+96] qui fournit des primitives de diffusion (*multicast*) avec ordre total fiable.

[3] Les requêtes IIOP sont une standardisation des messages échangés entre les ORB pour en assurer l'interopérabilité.

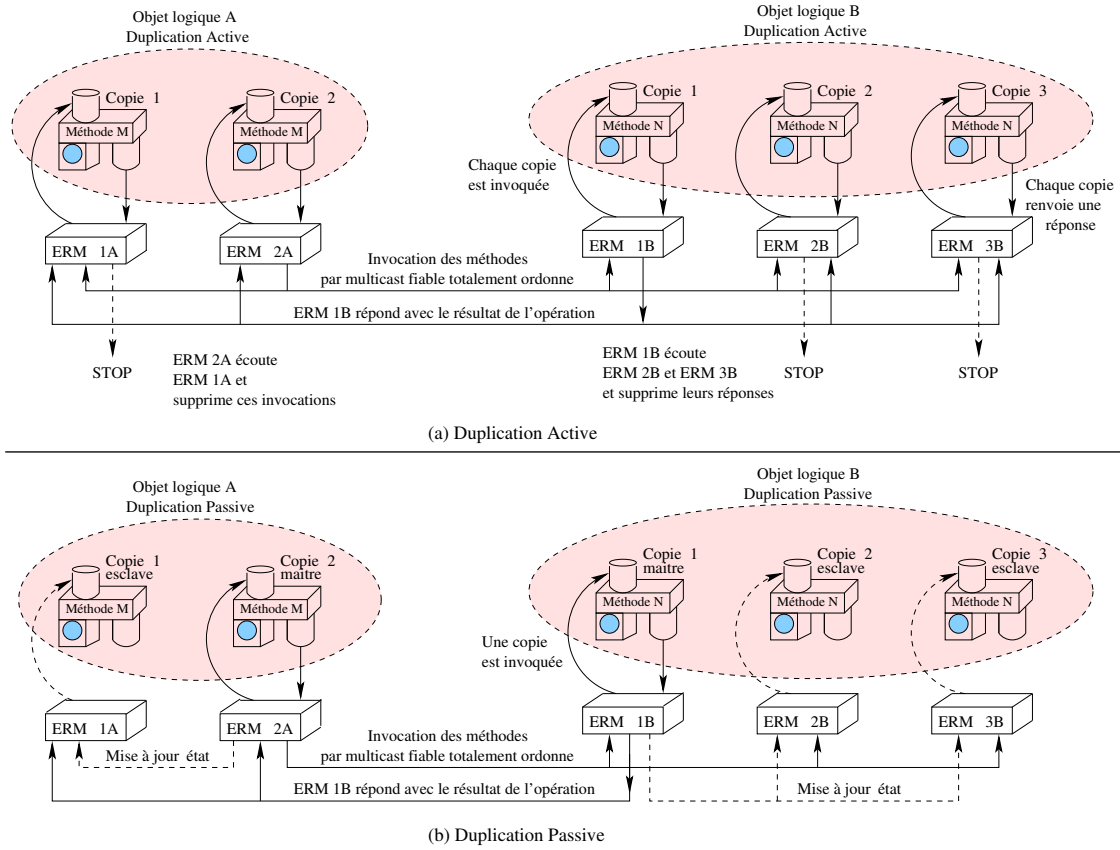


FIG. 3.8 – Duplication active/passive dans Eternal

Principe : Un talon client, généré à partir de la spécification IDL (*Interface Definition Language*), permet de passer les appels à l'ORB. L'Intercepteur Eternal capture les appels IIOP (*Internet Inter-ORB Protocol*) qui sont adressés et les passent au **Gestionnaire de Duplication Eternal** (*Eternal Replication Manager* **ERM**) qui les diffusent par l'intermédiaire de Totem. Sur l'objet serveur un talon serveur, également construit à partir des spécifications IDL, invoque l'opération sur l'objet serveur et retourne le résultat de l'opération à l'objet client. L'ERM gère le cycle de vie des copies : il crée les copies des objets et les répartit à travers le système afin d'obtenir le degré de duplication désiré. De plus, il envoie les opérations aux copies, maintient leur cohérence et détecte et résoud les fautes.

La notion de groupe d'objets : Un groupe d'objet est une abstraction d'une collection d'objets répartis. Cette abstraction permet à un objet d'invoquer les services d'un autre objet de manière transparente. L'objet qui invoque une opération sur un autre objet ne connaît pas la localisation, le degré de duplication ou le type de duplication utilisé. Totem permet

d'assurer un ordre total fiable à l'intérieur de chaque groupe et entre différents groupes. Le groupe est géré par l'ERM.

Duplication active : La figure 3.8 (a) montre comment est mis en œuvre la duplication active. L'ERM 1A (l'ERM associé à la copie 1 du groupe d'objet A) et l'ERM 2A communique leurs invocations aux ERM 1B, 2B et 3B. Les mécanismes de diffusion de Totem assurent que toutes les copies d'un objet reçoivent les mêmes messages dans le même ordre et qu'elles effectuent les mêmes opérations dans le même ordre. Afin d'éviter la duplication de message, Eternal fournit des mécanismes basés sur des identificateurs de message et d'opération qui détectent et suppriment les invocations et les réponses dupliquées.

Duplication passive : Lorsqu'il y a duplication passive (figure 3.8 (b)), l'ERM 2A envoie ses invocations aux ERM 1B 2B et3B. Seul l'ERM 1B invoque l'opération sur sa copie de l'objet B. Les deux autres ERM de l'objet B gardent le message pour l'utiliser en cas de défaillance de la copie primaire 1B. Une fois que la copie primaire a fini l'exécution de l'opération, l'ERM 1B transfère l'état de la copie aux copies secondaires 2B et 3B.

Adaptabilité : On retrouve la séparation code fonctionnel/code non fonctionnel, mais aucune adaptabilité au contexte non fonctionnel de l'aspect duplication. Il n'y a pas séparation et définition des interactions entre l'aspect duplication et les autres aspects. Il n'y a non plus aucune adaptabilité à l'intérieur des protocoles. Le nombre de protocoles supportés est très limité.

3.4.2 OGS

OGS (Object Group Service) [FGS98, FGS00] fournit la tolérance aux fautes et une haute disponibilité par l'intermédiaire de la duplication d'objets dans un environnement CORBA basé sur la communication de groupe. L'environnement OGS définit une architecture et un ensemble d'interfaces pour les groupes d'objets. L'architecture proposée est décomposée en plusieurs services CORBA :

- Le **service de transmission de messages** fournit une communication non bloquante fiable par diffusion et de un vers un.
- Le **service de surveillance** contrôle les objets et fournit des mécanismes de détection des fautes.
- Le **service de consensus** permet de résoudre le problème du consensus réparti. Ce service est utilisé pour implanter la diffusion de groupe et la construction des groupes.
- Le **service de groupe** fournit la diffusion de groupe et la construction des groupes.
  - La diffusion de groupe permet de diffuser des invocations à tous les membres du groupe avec différentes garanties de fiabilité et d'ordonnancement.

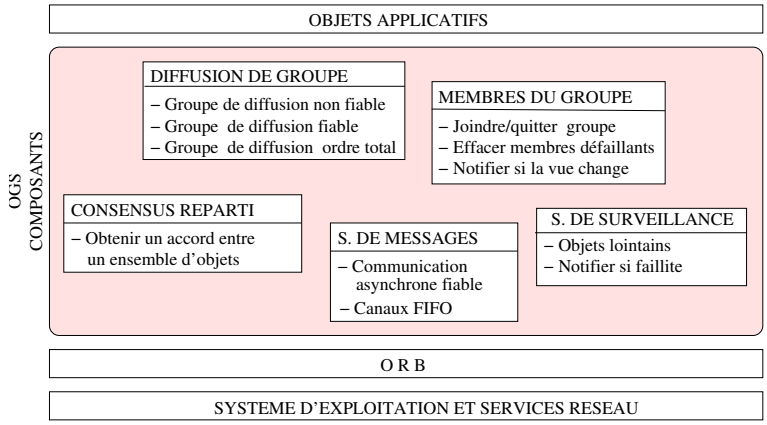


FIG. 3.9 – Composants OGS

– La construction des groupes maintient une liste à jour des membres corrects du groupe et fournit le support pour joindre et quitter les groupes, etc.

Travaux similaires. Newtop [MSEL99] est une boîte à outils de groupe de communication exploité sous forme de service pour fournir la tolérance aux fautes dans un environnement CORBA. L'idée est très proche de celle d'OGS. Il propose en plus de gérer les partitions réseaux en continuant à délivrer de manière totalement ordonnée les messages dans chaque sous-groupe (contrairement à OGS qui ne le fait que dans le sous groupe contenant la majorité des participants). Cependant, aucune procédure de réconciliation n'est prévue lorsque la communication est rétablie entre les sous-groupes.

IRL (*Interoperable Replication Logic*) [MMVB00] adopte également une approche par service. Il fournit la tolérance aux fautes par duplication active et passive. Contrairement aux autres, il ne s'appuie pas sur un système de communication de groupe. Un client s'adressant à un serveur dupliqué le fait par l'intermediaire d'un proxy (`SmartProxy`) contenant une référence vers un objet `ObjectGroup`. Cela permet d'offrir une transparence à la duplication pour le client. Un objet `ObjectGroup` ordonnance de manière séquentiel les requêtes adressées au groupe qu'il gère venant des différents clients. Il diffuse également les messages aux copies des serveurs en utilisant une communication point à point fiable. De plus, il met en œuvre le protocole de duplication.

Adaptabilité : Les mêmes remarques que pour l'approche par interception sont valables pour l'approche par service. Ce manque d'adaptabilité de l'aspect duplication dans le contexte CORBA provient du fait que l'objectif est de mettre en œuvre la tolérance aux fautes. A notre connaissance, il n'existe pas de spécification d'un service de duplication pouvant être utilisé dans différents contextes non fonctionnels (transactionnel, tolérant aux fautes, persistant, etc.)

Système	Adaptable		Protocoles supportés
	au contexte non fonctionnel	dans les protocoles	
Munin	Non	Paramétrisation	Protocoles pour modèle de cohérence au relâchement
FFCM	Oui approche modulaire mais limité aux MVP	Modularité mais limitée à trois composants	Divers protocoles
Garf	Non	Non	Duplication active et passive
Core	Oui mais il s'agit plutôt d'un gestionnaire de cohérence	Non	Divers protocoles
Globe	Non les différents modèles ne sont qu'entre les copies d'un même objet	Paramétrisation	Divers protocoles
Eternal	Non	Non	Duplication active et passive
OGS	Non	Non	Protocoles basés sur les communications de groupe

FIG. 3.10 – Adaptabilité et protocoles fournis par quelques systèmes

et supportant divers protocoles de duplication.

3.5 Conclusion

Nous avons commencé par définir ce que nous entendons par adaptable, adaptabilité et adaptation. L'adaptation est le processus d'adapter. Pour mettre en œuvre l'adaptation du support de la duplication, il est nécessaire que celui-ci soit adaptable. Il peut être adaptable par rapport à l'application, au contexte non fonctionnel et dans tout ou partie des protocoles de duplication. Un support est adaptable s'il offre l'adaptabilité. L'adaptabilité peut être obtenu par paramétrisation ou par modularité.

Notre objectif est d'offrir un support de duplication adaptable. Nous avons donc présenté dans ce chapitre divers supports de la duplication présentant une certaine adaptabilité. La figure 3.10 donne un résumé des différents travaux présentés.

Historiquement, les mémoires partagées réparties se sont premièrement intéressées à offrir l'adaptabilité dans le support de la duplication (approche modulaire). La séparation entre le code fonctionnel (l'application) et le code non fonctionnel est clairement faite grâce à la notion de modèle de cohérence. Cependant, la plupart du temps l'aspect duplication se trouve mélangé parmi les autres aspects non fonctionnels dans le protocole de cohérence. Cela entraîne un support de la duplication non adaptable en fonction du modèle de cohérence que l'on désire mettre en œuvre. Ainsi, FFCM propose d'isoler et de définir les interactions entre la duplication et l'ordonnancement des accès sur l'ensemble des objets. Il devient possible d'adapter le protocole de duplication en fonction du modèle de cohérence. Cependant, FFCM se limite au contexte des MVP. Le protocole de duplication ne peut être adapté à un contexte transactionnel ou tolérant aux fautes. Par ailleur, afin d'obtenir l'adaptabilité à l'intérieur du protocole de duplication, on retrouve l'approche par paramétrisation (Munin) et modu-

laire (FFCM). L'approche par paramétrisation montre vite ses limites alors que l'approche modulaire semble plus apte à prendre en compte la grande variété de protocoles existants.

Par la suite, les systèmes à objets se sont intéressés à offrir également l'adaptabilité pour le support de la duplication. Deux voies ont été suivies : les supports tournés vers la tolérance aux fautes supportant un nombre limité de protocoles de duplication (GARF) et ceux offrant un éventail plus large de protocoles (Core, Globe). Certains se sont attachés à offrir une certaine adaptabilité dans tout ou partie des protocoles de duplication (Globe), mais l'adaptabilité de l'aspect duplication au contexte non fonctionnel n'est pas claire. La principale critique que l'on peut faire est que ce qui est propre à la duplication n'est pas isolé des autres aspects (concurrence, transaction, tolérance aux fautes, modèle de cohérence, etc.) limitant la réutilisabilité, la flexibilité et l'adaptabilité.

A l'heure actuelle de nombreux travaux s'attachent à offrir la tolérance aux fautes dans le contexte CORBA (Eternal, OGS) par l'intermédiaire de la duplication active ou passive, duplication s'appuyant sur les communications de groupe. Du fait que ces travaux ont pour objectif de proposer des services de tolérance aux fautes, la même critique que précédemment peut être faite : il n'y a pas isolation et définition des interactions de la duplication avec les autres aspects non fonctionnels, n'offrant pas ainsi l'adaptabilité au contexte non fonctionnel. De plus, nous n'avons pas trouvé de travaux décomposant les protocoles de duplication en composants permettant d'offrir l'adaptabilité dans tout ou partie des protocoles de duplication afin de prendre en compte de nouveaux besoins.

Dans le contexte des SGBDR, nous n'avons pas trouvé de travaux s'intéressant à offrir l'adaptabilité à l'aspect duplication.

On s'adapte à tout, à l'inconfort, au froid, à la continence, au risque quotidien ; mais non à l'ignorance du sort de ce qu'on aime.
Henry de Montherlant - Fils de personne

Deuxième partie

Eléments de solution

Chapitre 4

Un canevas adaptable de services de duplication

Sommaire

Dans le chapitre précédent (chapitre 3), nous avons vu qu'il existe de nombreux travaux proposant des supports adaptables de la duplication. Cependant, si l'adaptabilité à l'application est largement étudiée, l'adaptabilité au contexte non fonctionnel et l'adaptabilité de tout ou partie des protocoles de duplication le sont beaucoup moins (chapitre 3, section 3.5). Ce manque est une limitation majeure que nous nous proposons d'étudier dans cette thèse.

Ce chapitre introduit notre proposition (présentée dans [DRD02b, DRD02a, DRD03]), nommée RS2.7, un canevas adaptable pour la construction de services de duplication. Il est organisé de la manière suivante : dans une première section (section 4.1), nous situons notre travail dans son contexte et nous motivons notre approche. Nous définissons ensuite quelle est, à notre avis, la partie générique d'un service de duplication (section 4.2), nous introduisons la notion de politique de duplication permettant de comprendre le rôle d'un service de duplication (section 4.3) et nous présentons certaines collaborations avec d'autres aspects (section 4.4). En fin de chapitre nous donnons nos conclusions (section 4.5).

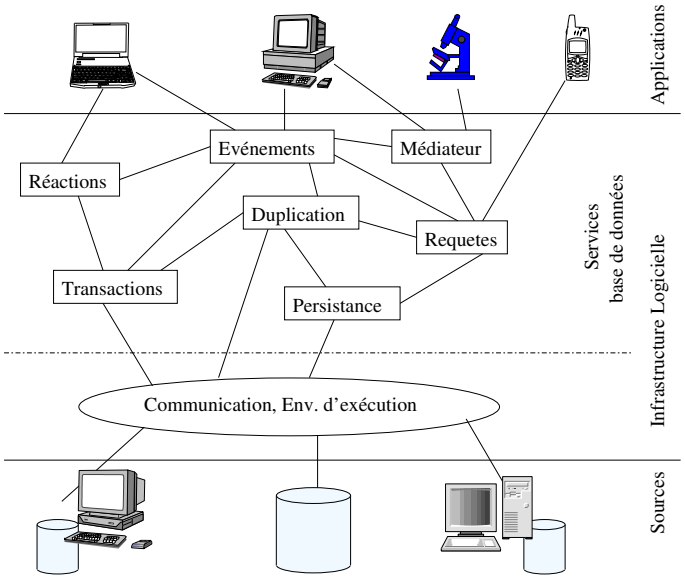


FIG. 4.1 – L'approche NODS

4.1 RS2.7 : Un canevas adaptable de services de duplication

Cette section présente notre contexte de travail (section 4.1.1), puis notre problématique (section 4.1.2) et enfin notre approche (section 4.1.3).

4.1.1 Le projet NODS

Nos travaux s'insèrent dans le contexte du projet NODS [Col00] (*Networked Open Database Services*). Le projet NODS vise à ouvrir les SGBD afin d'en identifier les fonctions et de les mettre en œuvre sous forme de services dans un environnement réparti (voire mobile), hétérogène et potentiellement de grande taille. L'idée générale de ce projet est de ne plus voir un SGBD comme un logiciel fermé dédié à la gestion des données, mais plutôt comme un ensemble de services coopérants, chaque service pouvant être adapté au mieux aux besoins de l'application qui va l'utiliser et au contexte dans lequel il va être utilisé. La figure 4.1 montre ce que pourrait être cette nouvelle vision d'un SGBD. On y distingue deux couches :

- Un support à la communication et à l'exécution des programmes dans un contexte réparti ;
- Un intergiciel offrant des services de transactions, persistance, événements, réactions ou notifications, requêtes, caches, duplication, tolérance aux fautes, médiation, etc.

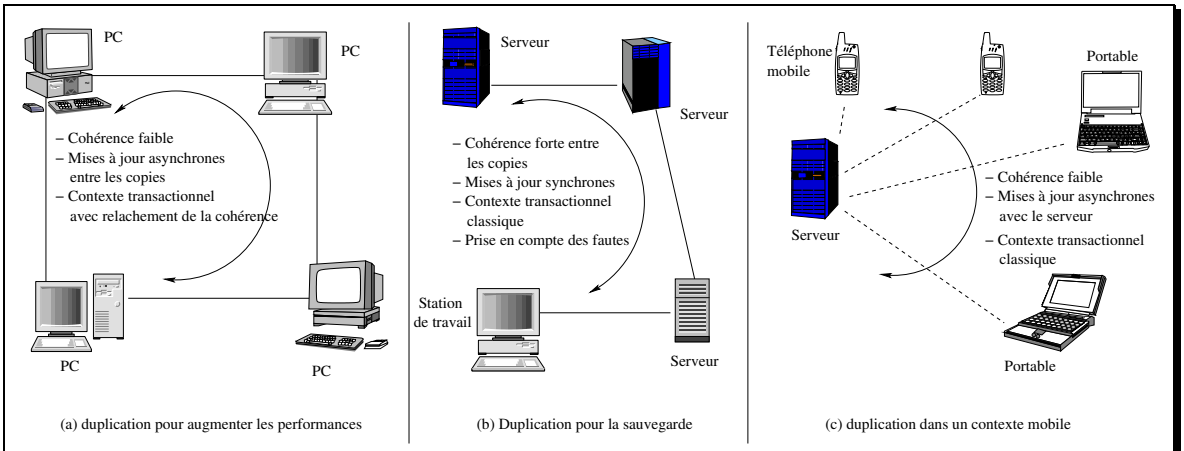


FIG. 4.2 – Choix d'un protocole de duplication approprié

Ouvrir les SGBD pour reconstruire selon les besoins des applications (besoins en terme de persistance, requêtes, duplication, etc.) requiert une connaissance des fonctionnalités internes des services et de la plate-forme sous-jacente. Ainsi, chaque service doit être à même d'exhiber son fonctionnement interne et de proposer une certaine adaptabilité.

4.1.2 Offrir un support adaptable de la duplication

Dans le chapitre 2 nous avons montré qu'il existe de nombreux protocoles de duplication, chacun approprié à un contexte et à un objectif particulier. Un protocole donné n'est pas intrinsèquement meilleur qu'un autre. Un protocole de duplication utilisé dans un contexte de duplication de serveurs pour faire de la tolérance aux fautes peut être très différent d'un protocole utilisé pour augmenter les performances ou d'un protocole utilisé dans un contexte mobile. La nature du contexte non fonctionnel (transactionnel, mémoire partagée répartie, etc.) joue également sur le choix du protocole (figure 4.2).

Si le choix d'un protocole adéquat n'est pas chose aisée, sa mise en œuvre l'est encore moins. De plus, l'évolution actuelle des systèmes informatiques (systèmes à grande échelle, informatique mobile) rend la tâche encore plus difficile, car les connaissances et la configuration du système peuvent évoluer au cours du temps. Afin de faciliter le travail du programmeur d'applications, nous avons vu au chapitre 3 qu'il est intéressant de proposer un support adaptable de la duplication. Ce support adaptable décharge le programmeur des considérations propres aux aspects non fonctionnels. Néanmoins, l'étude des travaux existants nous montre que ceux-ci offrent une adaptabilité limitée. La séparation application/code gérant la duplication est mise en avant par tous, mais (1) l'isolation et la définition des interactions entre le code spécifique à la duplication et les autres aspects non fonctionnels n'est pas clairement réalisée et (2) le code gérant la duplication apparaît bien souvent comme une boîte noire

limitant son adaptabilité. Ainsi, la définition de notre support adaptable de la duplication doit :

1. être indépendante de tout code applicatif.
2. supporter la duplication de n'importe quel type de composants (données, processus, page HTML, serveur de nom, etc.).
3. pouvoir se déployer dans différents contextes de programmation (Corba, EJB, DCOM, etc.).
4. être utilisable dans différents contextes non fonctionnels : transactionnel, mémoires partagées réparties, etc.
5. supporter différents protocoles de duplication. Il doit permettre d'obtenir aussi bien des protocoles assurant une cohérence forte que faible entre les copies.

Les points 1, 2 et 3 sont généralement considérés par un service de duplication. Cependant, il nous semble impossible de fournir un service de duplication unique et générique pouvant être paramétré afin de s'adapter à différents contextes non fonctionnels (point 4) et couvrant l'ensemble des protocoles existants (point 5) (voir chapitre 3).

4.1.3 Un canevas adaptable

Afin de résoudre les problèmes soulevés par les points 4 et 5 présentés ci-dessus, la meilleure solution nous semble être la définition d'un canevas adaptable de services de duplication.

Un canevas définit la structure générale d'une application générique. Les définitions de canevas que nous trouvons le plus souvent sont :

- Un canevas est un modèle réutilisable pour tout ou partie d'un système qui est représenté par un ensemble de classes abstraites et le moyen avec lequel leurs instances interagissent [Joh97].
- Un canevas est une portion d'un système logiciel existant pouvant être raffinée et étendue par des développeurs [Lor95].
- Un canevas est un squelette d'application qui peut être personnalisé par un développeur [Joh97].
- Un canevas est une bibliothèque de classes qui capture des patterns d'interaction entre objets utilisés ensembles [Rog97].
- Un canevas est une collection de patrons de description et d'utilisation de ressources qui représente une abstraction de ces ressources qui peut être projetée dans un modèle de programmation et déclinée en personnalités pour l'adapter aux standards du domaine considéré [Cou02].

Un canevas définit donc un ensemble de composants et les interactions entre ces composants, les classes d'un canevas étant vues sous forme de composants. Un composant facilite la réutilisation de code et il est théoriquement simple à utiliser. Idéalement il suffit de le

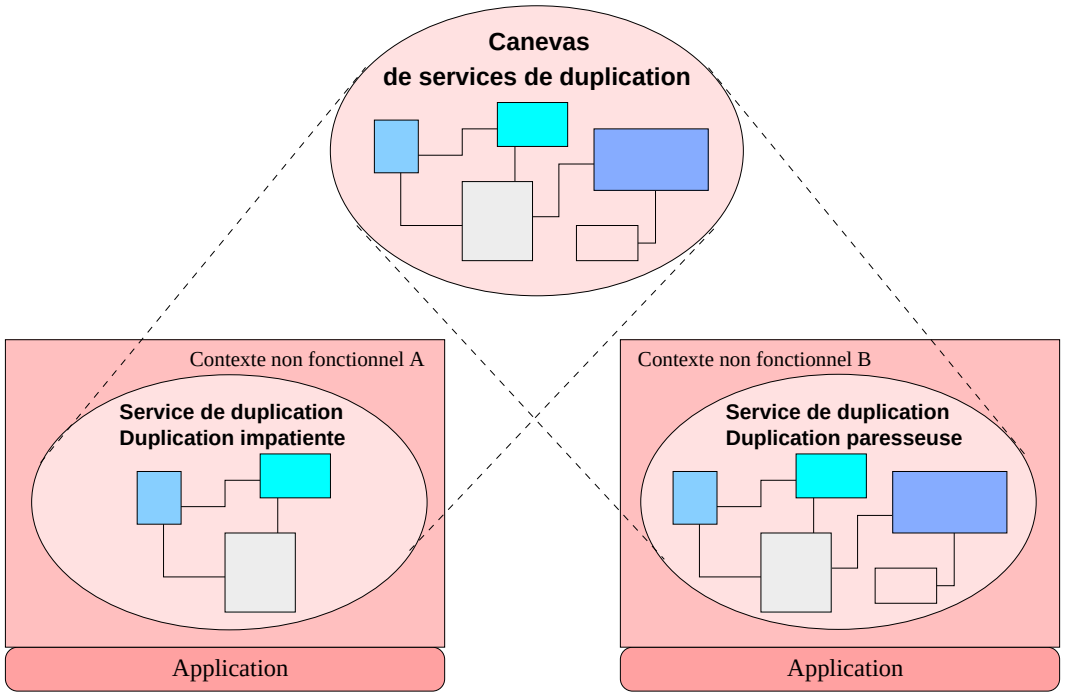


FIG. 4.3 – Développement de services de duplication à partir d'un canevas

connecter avec d'autres composants pour créer une application. En général un composant se caractérise par son rôle et son instance (sa mise en œuvre). Un rôle permet de décrire les propriétés attendues d'un composant. Il est défini par un nom, un ensemble de ports fournis (les fonctionnalités offertes par le composant), un ensemble de ports émis (messages envoyés vers d'autres composants) et un ensemble de rôles de composants avec lesquels il interagit (via les ports émis). Alors qu'il suffit de comprendre la signification des méthodes et de leurs paramètres de chaque composant séparément, la compréhension d'un canevas ne peut se limiter à la compréhension de chaque classe séparément. Il faut tenir compte des relations structurelles et comportementales entre les différents éléments du canevas.

De plus, nous voulons offrir la propriété d'adaptabilité à ce canevas, adaptabilité au contexte non fonctionnel et dans tout ou partie des protocoles. Notre approche consiste donc à définir un canevas adaptable de services de duplication, c'est à dire l'architecture générale d'un ensemble de services de duplication pouvant s'adapter. Ce canevas se nomme RS2.7 (*Replication Services*). Il peut être implanté de diverses façons afin d'obtenir un service de duplication mettant en œuvre un protocole de duplication particulier et approprié à l'application et au contexte non fonctionnel (figure 4.3).

Définition 4.1 : *Canevas adaptable de duplication*

Squelette d'un service de duplication définissant sa structure générale. Il permet d'obtenir des services de duplication indépendants de tout code applicatif, pouvant être utilisés dans différents contextes non fonctionnels (transactionnel, mémoires partagées réparties, etc.) et prennant en compte les contraintes et les protocoles spécifiques à chaque domaine.

La définition de RS2.7 se décompose de la manière suivante : premièrement, il nous faut définir ce qu'est un service de duplication obtenu à partir de RS2.7 (section 4.2), puis comment il est utilisé (section 4.3) et quelles sont les collaborations possibles avec les autres services (section 4.4). Ensuite, nous devons définir les différents types de services pouvant être obtenus à partir de RS2.7 (chapitre 5) afin de clairement définir un premier niveau d'interaction avec les autres aspects non fonctionnels. Une fois le canevas défini, nous pouvons nous intéresser à lui offrir la propriété d'adaptabilité : adaptabilité au contexte non fonctionnel en définissant un deuxième niveau d'interaction avec d'autres aspects non fonctionnels (chapitre 6) et adaptabilité dans tout ou partie des services proposés en factorisant les protocoles de duplication existants (chapitre 7).

4.2 Noyau minimal d'un service de duplication

Même si dans la littérature de nombreux travaux [PNMS02, KKvST98, GGM95, BCM95] proposent un support de duplication, il n'y a pas consensus sur son rôle (voir chapitre 3, section 3.5). Nous insistons sur le fait qu'un tel support doit permettre de faciliter sa réutilisation et l'adaptabilité sans empiéter sur le rôle des autres aspects non fonctionnels. Ainsi, nous définissons le noyau minimal d'un service de duplication comme suit :

Définition 4.2 : *Service de duplication*

Un service de duplication a pour rôle de gérer le cycle de vie des copies d'un objet dupliqué ainsi que leur mise en cohérence lorsque cela est nécessaire.

Un service de duplication prend donc en charge deux tâches :

- la **gestion du cycle de vie** des différentes copies concerne le processus de création et de destruction des copies. Cette gestion ne couvre pas la prise de décision de ces actions.
- la **gestion de la mise en cohérence** concerne uniquement la mise en cohérence des différentes copies d'un objet dupliqué. Elle dépend du "niveau de cohérence" désiré. Cette tâche est implantée par ce que nous nommons un protocole de cohérence locale dans la suite (voir chapitre 5).

Ces tâches peuvent être mises en œuvre de différentes manières selon les besoins de l'ap-

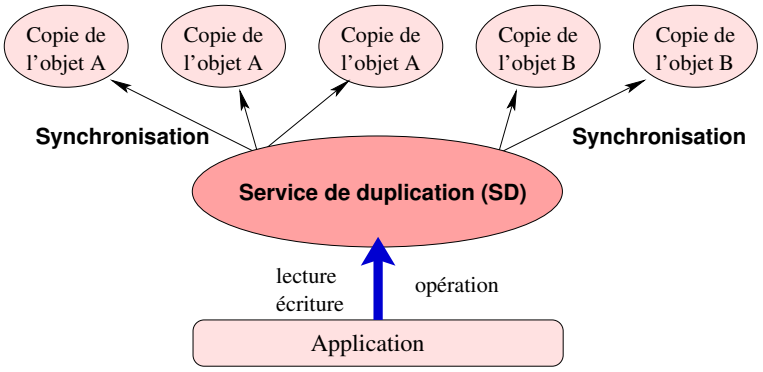


FIG. 4.4 – Interception des accès à un objet dupliqué par un service de duplication

plication, le contexte et/ou l'objectif. Ainsi, on obtient différents services de duplication.

Afin de garantir la cohérence entre les copies, le service de duplication doit capturer les informations concernant les accès en lecture et écriture aux objets dupliqués, et plus généralement toutes les opérations sur ces objets. Dans la figure 4.4, le service de duplication SD, qui est une instance de RS2.7, intercepte les accès en lecture, en écriture et les opérations effectuées sur les objets dupliqués A et B. Il synchronise chacune des copies de l'objet A et de l'objet B suivant le protocole de cohérence locale qu'il met en œuvre. Les interfaces de gestion du cycle de vie et d'accès d'un objet dupliqué sont les suivantes :

Interface du gestionnaire du cycle de vie d'un objet dupliqué :

- `Replica addReplica(DuplicableObject do)` crée une nouvelle copie de l'objet dupliquable `do`.
- `removeReplica(DuplicableObject do, Replica r)` supprime la copie `r` de l'objet dupliquable `do`.

∎

Interface de l'accesseur à un objet dupliqué :

- `Value read(Field f)` lit le champ `f` de l'objet dupliqué.
- `write(Field f, Value v)` écrit la valeur `v` dans le champ `f` de l'objet dupliqué.
- `executeMethod(Name n, Arguments as)` exécute la méthode `n` avec l'ensemble d'arguments `as` sur l'objet dupliqué.

∎

4.3 La notion de politique de duplication

Lorsqu'une application a besoin de la fonctionnalité de duplication, gérer le cycle de vie des copies et leur mise en cohérence n'est pas suffisant. Il est nécessaire de définir d'autres

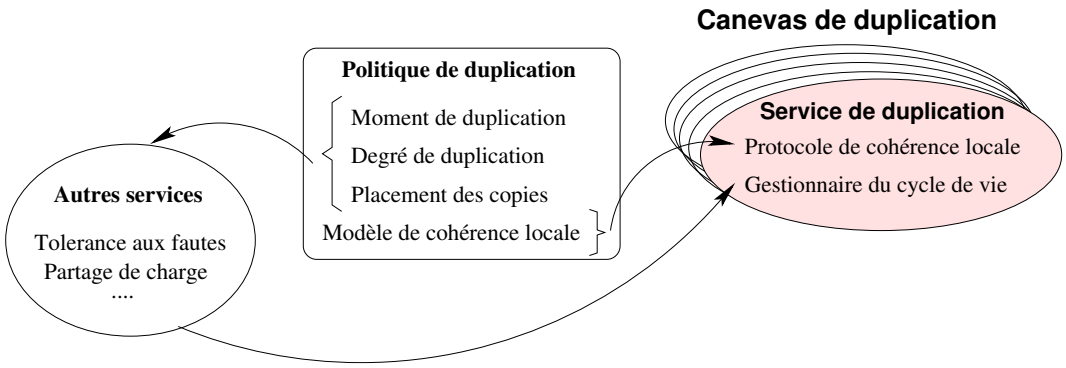


FIG. 4.5 – Politique de duplication / service de duplication

tâches, qui elles sont très dépendantes du contexte et ne peuvent donc pas faire partie du noyau minimal d'un service de duplication. Ainsi, nous définissons la notion de politique de duplication transverse à plusieurs services. Elle comporte quatre points :

1. le **moment** de duplication définissant les instants où il est nécessaire de créer ou détruire des copies,
2. le **degré** de duplication définissant le nombre de copies qu'il doit y avoir dans le système,
3. le **placement** des copies définissant où mettre les copies dans le système et
4. la **cohérence** à assurer entre les copies, mise en œuvre par le protocole de cohérence locale.

Le quatrième point est à la charge d'un service obtenu à partir de RS2.7 comme indiqué dans la section précédente (section 4.2). D'autres services prennent les décisions concernant les trois premiers points. Ils s'appuient sur le gestionnaire du cycle de vie du service de duplication pour créer et/ou détruire des copies selon leurs besoins (figure 4.5). Des services de partage de charge ou de tolérance aux fautes peuvent assurer, par exemple, ces prises de décision. Ainsi, RS2.7 n'offre pas des services transparents. Un service de duplication est dirigé par d'autres services, ou par une couche de transparence, ayant conscience du placement des copies, du degré de duplication et du moment de création/destruction des copies. Dans le cas extrême, toutes ces décisions peuvent être prises par l'application.

Nous reviendrons sur le quatrième point dans le chapitre 5 où nous caractérisons les différents types de cohérence pouvant exister entre les copies.

4.4 Collaboration avec d'autres aspects

Un service de duplication collabore également avec d'autres aspects lors de son fonctionnement, notamment avec le cache et le service de persistance.

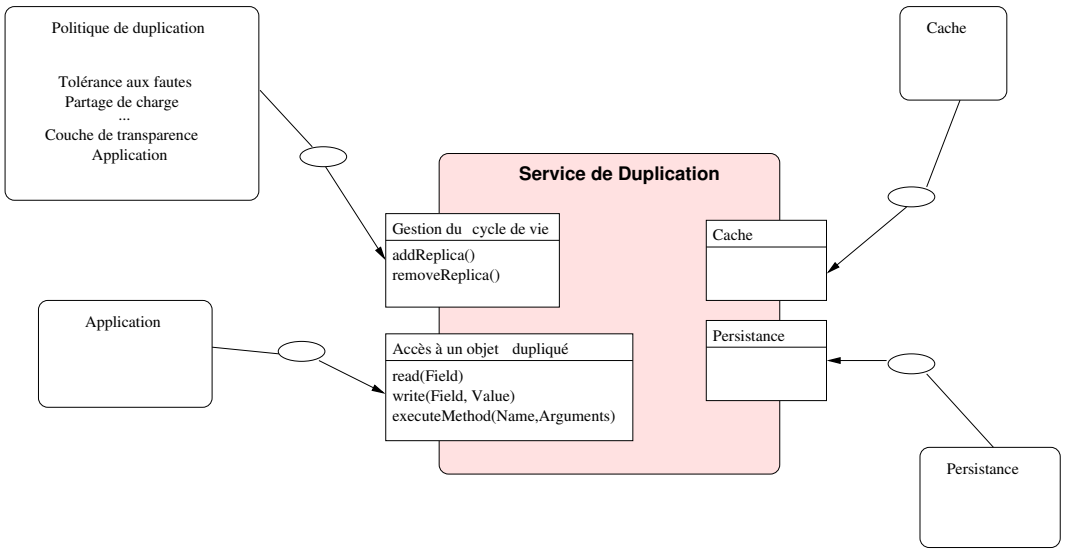


FIG. 4.6 – Intéractions avec le service de duplication

Collaboration avec le cache. Les objets manipulés sont gérés en mémoire par un cache. Un cache est une liste de références d'objets gérée selon diverses politiques de remplacement : LRU (*Last Recently Used*), FIFO (*First In First Out*), LFU (*Last Frequently Use*), etc. Ainsi, des objets peuvent être supprimés ou ajoutés dans la liste gérée par le cache. Un objet supprimé du cache ne peut plus, en principe, être utilisé. Il est donc intéressant qu'il ne soit également plus pris en compte par le service de duplication. Ainsi, un service de duplication doit être informé quand une copie est ajoutée ou retirée du cache.

Collaboration avec le service de persistance. Un service de persistance assure le stockage d'objets sur un support permanent. Toutes les copies n'ayant pas forcément le même degré de fraîcheur, il est utile que le service de duplication collabore avec le service de persistance afin de ne pas déclencher le processus d'écriture sur support permanent de manière inutile. Un service de persistance doit être informé du degré de fraîcheur des copies.

4.5 Conclusion

Dans ce chapitre, nous avons commencé par présenter la raison d'être et l'utilité de proposer un canevas adaptable de services de duplication. Un canevas adaptable permet de proposer une architecture générique permettant d'implanter divers services de duplication correspondant aux besoins des applications, pouvant s'adapter au contexte non fonctionnel et couvrant de nombreux protocoles de cohérence locale.

Ensuite nous avons défini ce que nous pensons être un service de duplication. Afin de faciliter la réutilisabilité et de clairement séparer les rôles de chacun, un service de duplication prend en charge deux tâches : la gestion du cycle de vie des copies et la mise en cohérence des différentes copies lorsque c'est nécessaire. Nous avons vu également que la gestion du cycle de vie est dirigée par d'autres services (partage de charge, tolérance aux fautes, etc.) dans ce que nous appelons une politique de duplication. De plus, un service de duplication collabore également avec d'autres services comme le gestionnaire de cache ou la persistance. Le diagramme de classes de la figure 4.6 illustre les interactions qu'un service de duplication peut avoir avec d'autres aspects non fonctionnels.

Dans le chapitre suivant (chapitre 5), nous définissons les différents types de protocole de cohérence locale qu'un service de duplication, construit en utilisant RS2.7, peut offrir. Cette caractérisation est une modélisation des différents protocoles de cohérence locale existant dans la littérature (chapitre 2). Elle nous permet de dégager un premier niveau d'interaction avec d'autres aspects non fonctionnels.

Le cerveau ne détermine pas la pensée, comme le cadre ne détermine pas le tableau.
Bergson, Henri - L'Energie spirituelle

Chapitre 5

Modélisation des services : les modèles de cohérence locale

Sommaire

Afin de caractériser les différents types de services de duplication qui peuvent être construits à partir de RS2.7, nous introduisons la notion de modèle de cohérence locale [1]. Un modèle de cohérence locale définit comment les utilisateurs d'un service perçoivent les différentes copies d'un même objet. C'est un contrat entre le service et son utilisateur qui spécifie les accès aux copies et leur perception par l'utilisateur. Un modèle de cohérence locale est mis en œuvre par un protocole de cohérence locale, un même modèle pouvant être implanté par différents protocoles.

Définition 5.1 *: Modèle de cohérence locale*
Un modèle de cohérence locale spécifie la vue offerte aux utilisateurs d'un service de duplication sur les différentes copies d'un objet dupliqué.

Cette notion nous permet de proposer une première séparation des considérations entre la duplication et d'autres aspects non fonctionnels.

[1] Nous parlons de cohérence locale car portant sur les copies d'un objet logique.

Notation	*Signification*
O_j	Objet non dupliqué ou son objet logique
$r_i(O_j)v$	Lecture de la valeur v sur l'objet O_j par le processus P_i.
$w_i(O_j)v$	Écriture sur l'objet O_j de la valeur v par le processus P_i.
h_i	Ensemble des opérations survenues sur le processus P_i.
H	Ensemble des opérations survenues dans le système réparti.
$\rightarrow_{h_i}$	Relation d'ordre entre les opérations exécutées par le processus P_i.
$\rightarrow_H$	Relation d'ordre entre les opérations exécutées dans le système.
$\widehat{h_i} = (h_i, \rightarrow_{h_i})$	Histoire locale du processus P_i.
$\widehat{H} = (H, \rightarrow_H)$	Histoire globale du système réparti.

FIG. 5.1 – Notations utilisées pour modéliser une exécution répartie

Ce chapitre est organisé de la manière suivante : dans une première section (section 5.1) nous présentons des définitions préliminaires permettant de définir quatre types de modèle de cohérence locale (section 5.2). Nous montrons ensuite les interactions entre la duplication et d'autres aspects non fonctionnels afin de pouvoir garantir les modèles définis (section 5.3). Nous terminons ce chapitre par une conclusion (section 5.4).

5.1 Définitions préliminaires

Cette section introduit les concepts préliminaires permettant de modéliser une exécution sur un objet dupliqué. Un système réparti est traditionnellement modélisé comme une collection de processus asynchrones communiquant par messages [Lam78, CT96]. Chaque processus produit séquentiellement et de façon atomique des événements de trois types : interne au processus, envoi d'un message et réception d'un message. [Lam78] a le premier introduit ce modèle et la relation de précédence causale (*happened before*). Un événement e précède causalement un événement e' s'il respecte une des trois propriétés suivantes : (i) e et e' se déroulent sur le même processus et e précède e' dans l'ordre du programme, (ii) e est l'envoi d'un message par un processus et e' est la réception de ce message par un autre processus, (iii) il existe un événement e" tel que e précède e" et e" précède e' (transitivité). Informellement cela signifie que si un événement e' est causé ou influencé par un événement e, alors tout le système doit observer l'événement e avant l'événement e'. Une exécution répartie se modélise ainsi par un ordre.

Cette section est organisée de la manière suivante : nous commençons par rappeler certaines définitions sur les ordres (section 5.1.1), puis nous présentons la définition de l'extension linéaire (section 5.1.2), de la modélisation d'une exécution répartie (section 5.1.3), de la légalité des lectures (section 5.1.4) et nous terminons par la modélisation d'une exécution sur un objet dupliqué (section 5.1.5). Les tableaux 5.1 et 5.2 reprennent chacune des notations présentées dans les sections suivantes.

Notation	*Signification*
$O_{j,k}$	Copie k de l'objet logique O_j.
$r_i(O_{j,k})v$	Lecture de la valeur v sur la copie $O_{j,k}$ par le processus P_i.
$w_i(O_{j,k})v$	Écriture sur la copie $O_{j,k}$ de la valeur v par le processus P_i.
$h_{O_{j,k}}$	Ensemble des opérations survenues sur la copie $O_{j,k}$.
H_{O_j}	Ensemble des opérations survenues sur l'objet logique O_j.
$\rightarrow_{h_{O_{j,k}}}$	Relation d'ordre entre les opérations exécutées sur la copie $O_{j,k}$.
$\rightarrow_{H_{O_j}}$	Relation d'ordre entre les opérations exécutées sur l'objet logique O_j.
$\widehat{h}_{O_{j,k}} = (h_{O_{j,k}}, \rightarrow_{h_{O_{j,k}}})$	Histoire locale de la copie $O_{j,k}$.
$\widehat{H}_{O_j} = (H_{O_j}, \rightarrow_{H_{O_j}})$	Histoire globale de l'objet logique $O_{j,k}$.

FIG. 5.2 – Notations utilisées pour modéliser une exécution sur un objet dupliqué

5.1.1 Ordre total, ordre partiel

Un ordre permet de comparer les éléments d'un ensemble. Formellement, un ordre $\widehat{E} = (E, \rightarrow_E)$ est une relation binaire $\rightarrow_E$ sur un ensemble E telle que $\forall x, y, z \in E$ on ait :

- réflexivité : $x \rightarrow_E x$.
- anti-symétrie : $(x \rightarrow_E y$ et $y \rightarrow_E x) ==> y = x$.
- transitivité : $(x \rightarrow_E y$ et $y \rightarrow_E z) ==> x \rightarrow_E z$.

Un ordre est total si deux éléments sont toujours comparables : $\forall x, y \in E$, $x \rightarrow_E y$ ou $y \rightarrow_E x$. Un ordre total $\widehat{E} = (E, \rightarrow_E)$ est un ensemble E et une relation binaire transitive, anti-symétrique et non réflexive $\rightarrow_E$ sur E.

S'il existe deux éléments au moins non comparables, l'ordre est un ordre partiel : $\exists x, y \in E$ tel que $\neg(x \rightarrow_E y)$ et $\neg(y \rightarrow_E x)$. Un ordre partiel $\widehat{E} = (E, \rightarrow_E)$ est un ensemble E et une relation binaire transitive, anti-symétrique et non réflexive $\rightarrow_E$ sur E.

5.1.2 Extension linéaire

Une extension linéaire $\widehat{S} = (S, \rightarrow_S)$ d'un ordre partiel $\widehat{E} = (E, \rightarrow_E)$ est un tri topologique de cet ordre partiel, tel que :

(i) $S = E$,
(ii) $\forall x, y \in E, x \rightarrow_E y \Rightarrow x \rightarrow_S y$
($\widehat{S}$ maintient l'ordre de toutes les opérations ordonnées de $\widehat{E}$) et
(iii) $\rightarrow_S$ définit un ordre total.

5.1.3 Modélisation d'une execution répartie

On note $r_i(O_j)v$ la lecture par le processus P_i sur l'objet O_j ayant pour résultat la valeur v et $w_i(O_j)v$ l'écriture par le processus P_i sur l'objet O_j de la valeur v. Dans la suite, sans perte de généralité pour notre cadre d'étude et pour simplifier le discours, nous faisons l'hypothèse que toutes les valeurs écrites dans un objet sont distinctes. Ainsi, à toute opération de lecture d'un objet correspond une unique opération d'écriture.

Lors de l'exécution d'un programme réparti, chaque processus exécute une séquence d'opérations (lecture ou écriture) sur un ensemble d'objets. L'ensemble des opérations d'un processus P_i est noté h_i. Si un processus P_i effectue l'opération op_1 puis op_2, alors op_1 précède op_2 dans l'ordre du programme de P_i ($op_1 \rightarrow_{h_i} op_2$). L'ensemble des séquences du processus P_i est appelée histoire locale $\widehat{h_i} = (h_i, \rightarrow_{h_i})$ du processus P_i.

Une exécution répartie est représentée par un ordre partiel sur l'ensemble des opérations H exécutées par tous les processus. Cet ordre partiel est appelé histoire globale. Cette histoire $\widehat{H} = (H, \rightarrow_H)$ d'un ensemble de processus $P_1, ..., P_n$ est définie comme suit [Lam78] :

- $H = \bigcup_i h_i$.
- $op_1 \rightarrow_H op_2$ si :
  (i) $\exists P_i : op_1 \rightarrow_{h_i} op_2$ ou
  (ii) $w_i(O_1)x = op_1$ et $r_j(O_1)x = op_2$ ou
  (iii) $\exists op_3 : (op_1 \rightarrow_H op_3)$ et $(op_3 \rightarrow_H op_2)$.

Le point (i) est appelée relation "ordre programme" (*program order*) : si op1 précède op2 sur le processus P_i alors op1 précède op2 dans $\widehat{H}$. Le point (ii), appelée relation "lecture de" (*read-from*), précise que si un processus P_j lit une valeur x écrite par un processus P_i alors l'écriture précède la lecture dans $\widehat{H}$ (on suppose que les valeurs de toutes les écritures sont distinctes). Le point (iii) définit la transitivité. La relation d'ordre $\rightarrow_H$ est également non réflexive. Ce fait n'est pas donné comme condition dans la définition [Lam78] mais apparaît plus tard dans l'article. Deux opérations op_1 et op_2 sont concurrentes si $\neg(op_1 \rightarrow_H op_2)$ et $\neg(op_2 \rightarrow_H op_1)$.

5.1.4 Lecture légale

Une opération de lecture est légale si elle lit la dernière valeur écrite. Plus formellement, une lecture $r(O_1)v$ est légale si $\exists w(O_1)v$ telle que :

- $w(O_1)v \rightarrow_H r(O_1)v$,
- $\nexists w(O_1)v'$ telle que $w(O_1)v \rightarrow_H w(O_1)v' \rightarrow_H r(O_1)v$ (il n'existe pas d'opération d'écriture intermédiaire)

Une histoire $\widehat{H}$ est légale si toutes ses opérations de lecture sont légales.

5.1.5 Modélisation d'une exécution sur un objet dupliqué

Un objet logique O_j se compose d'un ensemble de copies $O_{j,k}$. Chaque copie est associée à un processus distinct. Chaque copie reçoit une succession d'événements d'accès (écriture ou lecture). On note $w_i(O_{j,k})v$ une écriture de la valeur v sur la copie $O_{j,k}$ par le processus P_i et $r_i(O_{j,k})v$ une lecture de la copie $O_{j,k}$ par le processus P_i et ayant pour valeur v.

L'ensemble des opérations sur une copie $O_{j,k}$ est noté $h_{O_{j,k}}$. Si les opérations op_1 et op_2 sont effectuées sur la copie $O_{j,k}$ et op_1 a eu lieu en premier, alors op_1 précède op_2 sur la copie $O_{j,k}$ ($op_1 \rightarrow_{h_{O_{j,k}}} op_2$). Nous appelons l'ensemble des séquences ayant eu lieu sur une copie $O_{j,k}$ l'histoire locale de la copie $O_{j,k}$ et nous la notons $\widehat{h}_{O_{j,k}} = (h_{O_{j,k}}, \rightarrow_{h_{O_{j,k}}})$.

Une exécution sur un objet logique O_j est représentée par un ordre partiel sur l'ensemble des opérations H_{O_j} exécutées par toutes les copies. Nous appelons cet ordre partiel l'histoire globale de l'objet logique O_j. Nous définissons cette histoire $(H_{O_j}, \rightarrow_{H_{O_j}})$ comme suit :

Définition 5.2 : *Histoire globale $\widehat{H}_{O_j} = (H_{O_j}, \rightarrow_{H_{O_j}})$ d'un objet logique O_j*

L'histoire globale $\widehat{H}_{O_j} = (H_{O_j}, \rightarrow_{H_{O_j}})$ de l'objet logique O_j composé d'un ensemble de copies $O_{j,1}$, $O_{j,2}$, ..., $O_{j,k}$ est construite comme suit :

- $H_{O_j} = \bigcup_k h_{O_{j,k}}$.
- $op_1 \rightarrow_{H_{O_j}} op_2$ si :
  (i) $\exists O_{j,k} : op_1 \rightarrow_{h_{O_{j,k}}} op_2$ ou
  (ii) $w_i(O_{j,k})x = op_1$ et $r_{i'}(O_{j,k'})x = op_2$ ou
  (iii) $\exists op_3 : (op_1 \rightarrow_{H_{O_j}} op_3)$ et $(op_3 \rightarrow_{H_{O_j}} op_2)$.
- $\rightarrow_{H_{O_j}}$ est non réflexive.

Le point (i) précise que si op1 précède op2 sur la copie $O_{j,k}$ alors op1 précède op2 dans $\widehat{H}_{O_j}$ et le point (ii) que si un processus $P_{i'}$ lit une valeur x sur la copie $O_{j,k'}$, valeur écrite par le processus P_i sur la copie $O_{j,k}$, alors l'écriture précède la lecture dans $\widehat{H}_{O_j}$. Ce deuxième point définit en fait une synchronisation : une valeur écrite par un processus sur une copie est lue par un autre processus sur une autre copie. Le point (iii) définit la transitivité. La relation d'ordre $\rightarrow_H$ est également non réflexive. Deux opérations op_1 et op_2 sont concurrentes si $\neg(op_1 \rightarrow_{H_{O_j}} op_2)$ et $\neg(op_2 \rightarrow_{H_{O_j}} op_1)$.

5.2 Quatre types de modèle de cohérence locale

Une fois l'exécution sur un objet dupliqué modélisée, nous pouvons définir différents modèles de cohérence locale. Un modèle de cohérence locale est une définition formelle d'une garantie offerte sur l'histoire de l'exécution d'un ensemble d'opérations sur un objet logique,

c'est à dire une garantie sur un ordonnancement des opérations. Nous introduisons cette notion afin de caractériser les différents protocoles de cohérence locale des services de duplication pouvant être obtenus à partir de RS2.7.

Nous distinguons quatre types de modèle de cohérence locale que nous détaillons dans les sections suivantes : les modèles à copie unique (section 5.2.1), les modèles à copies divergentes (section 5.2.2), les modèles à copies convergentes avec lecture sur les copies divergentes (section 5.2.3) et les modèles à copies convergentes avec écriture sur les copies divergentes (section 5.2.4). De plus, il est possible de combiner ces modèles afin d'obtenir des modèles hybrides où des sous-ensembles des copies respectent les garanties de différents modèles (section 5.2.5).

5.2.1 Modèles à copie unique

Pour les modèles de cohérence locale à copie unique, l'utilisateur a l'illusion qu'il n'y a pas duplication. Toutes les copies possèdent la même histoire locale ; l'utilisateur accédant toujours une copie reflétant la "dernière" écriture sur l'objet logique. Le terme "dernière" peut se référer soit au temps logique, soit au temps physique. Ainsi, nous définissons le modèle de cohérence locale à copie unique séquentiel basé sur le temps logique et le modèle à copie unique atomique basé sur le temps physique .

Définition 5.3 : *Modèle de cohérence locale à copie unique séquentiel*

Un modèle de cohérence locale à copie unique séquentiel garantit que $\widehat{H}_{O_j}$ admet une extension linéaire $\widehat{S}_{O_j} = (H_{O_j}, \rightarrow_{H_{O_j}})$ légale.

$\widehat{H}_{O_j}$ doit donc admettre un ordre total légal afin de respecter ce modèle de cohérence locale, autrement dit cela signifie que toutes les copies doivent percevoir l'ensemble des écritures H_{O_j} dans le même ordre.

Afin de définir le modèle de cohérence locale atomique, il est nécessaire de redéfinir la relation d'ordre $\rightarrow_{H_{O_j}}$. Celle-ci doit également porter sur le temps physique : si op_1 a eu lieu avant op_2 par rapport au temps physique ($op_1 <_T op_2$) alors $op_1 \rightarrow_{H_{O_j}} op_2$.

Définition 5.4 : *Modèle de cohérence locale à copie unique atomique*

Soit $\widehat{H}_{O_j}$ défini comme dans la définition 5.2 et (iv) $op_1 <_T op_2$, alors un modèle de cohérence locale à copie unique atomique garantit que $\widehat{H}_{O_j}$ admet une extension linéaire $\widehat{S}_{O_j} = (H_{O_j}, \rightarrow_{H_{O_j}})$ légale.

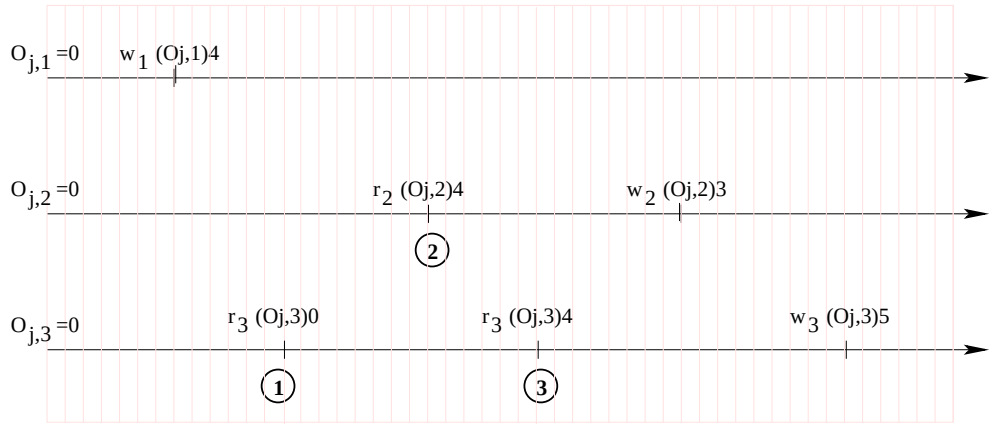


FIG. 5.3 – Modèle de cohérence locale à copie unique séquentiel

Exemples d'exécution. La figure 5.3 présente une exécution sur un objet logique O_j composé de trois copies tel qu'à $t = 0$, $O_{j,1} = O_{j,2} = O_{j,3} = 0$ et $\widehat{H}_{O_j} = (\emptyset, \rightarrow_{H_{O_j}})$. L'exécution est la suivante : une écriture est réalisée sur la copie 1 ($w_1(O_{j,1})4$), puis une lecture sur la copie 3 ($r_3(O_{j,3})0$), une lecture sur la copie 2 ($r_2(O_{j,2})4$), une lecture sur la copie 3 ($r_3(O_{j,3})4$), une écriture sur la copie 2 ($w_2(O_{j,2})3$) et enfin une écriture sur la copie 3 ($w_3(O_{j,3})5$).

Cette exécution satisfait le modèle de cohérence locale à copie unique séquentiel. L'ordre total légal (la vue commune perçue par l'ensemble des copies) est le suivant : $r_3(O_{j,3})0$, $w_1(O_{j,1})4$, $r_2(O_{j,2})4$, $r_3(O_{j,3})4$, $w_2(O_{j,2})3$, $w_3(O_{j,3})5$. L'écriture faite sur la copie $O_{j,1}$ n'est perçue qu'aux points 2 et 3 sur les copies $O_{j,2}$ et $O_{j,3}$. La lecture faite sur la copie $O_{j,3}$ au point 1 après l'écriture sur la copie $O_{j,1}$ par rapport au temps physique est supposée être antérieure vis à vis du temps logique.

Par contre cette exécution ne satisfait pas le modèle de cohérence locale atomique car, au point 1, la copie $O_{j,3}$ devrait avoir la valeur de la copie $O_{j,1}$, car $w_1(O_{j,1})4 <_T r_3(O_{j,3})0$ implique que $w_1(O_{j,1})4 \rightarrow_{H_{O_j}} r_3(O_{j,3})0$.

Protocoles de la littérature. Les protocoles ROWA, ROWAA, ou à base de quorum s'appuient sur un modèle de cohérence locale à copie unique atomique. C'est aussi le cas des protocoles de duplication active, passive ou semi-active dans les SCG. Dans les SGBDR, les protocoles de duplication impatiente (mise à jour de toutes les copies dans une même transaction) reposent sur le modèle séquentiel et bien souvent sur le modèle atomique. Dans les mémoires partagées réparties, les protocoles pour cohérence séquentielle et atomique utilisent un protocole de cohérence locale à copie unique du même nom. Nous reviendrons sur les protocoles de cohérence locale à utiliser afin de mettre en œuvre les protocoles de cohérence des MPR dans le chapitre 6.

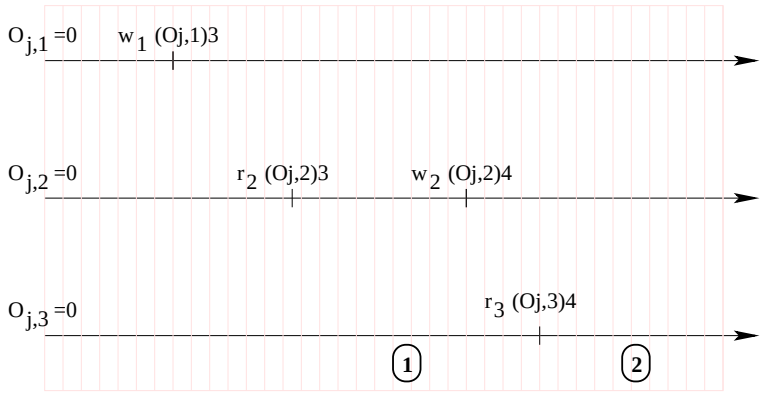


FIG. 5.4 – Modèle de cohérence locale à copies divergentes causale

5.2.2 Modèles à copies divergentes

Les modèles considérés ici permettent aux différentes copies d'un même objet de diverger les unes par rapport aux autres. Il est possible de caractériser cette divergence en donnant des garanties sur l'histoire globale $\widehat{H}_{O_j}$. On peut ainsi définir différents modèles suivant la garantie souhaitée. Nous en présentons trois dans la suite : le modèle de cohérence locale à copies divergentes causale, le modèle de cohérence locale à copies divergentes FIFO et le modèle de cohérence locale à copies divergentes à la longue.

Le modèle de cohérence locale à copies divergentes causale.

Ce modèle garantit que toutes les copies de l'objet logique perçoivent les opérations d'écriture causalement liées dans le même ordre. Il n'y a aucune contrainte sur les opérations concurrentes (sans lien de causalité).

La définition 5.2 (section 5.1.5) de l'histoire globale $\widehat{H}_{O_j}$ d'un objet dupliqué fait ressortir les liens de causalité entre les opérations ($op1 \rightarrow_{H_{O_j}} op2$ si (i), (ii) ou (iii)). Ainsi, afin d'assurer le modèle de cohérence locale à copies divergentes causale, il est nécessaire que l'histoire $h_{O_{j,k}}$ de chacune des copies $O_{j,k}$ soit légale. Ici, $h_{O_{j,k}}$ est l'histoire induite à la fois par l'ensemble de toutes les écritures de $\widehat{H}_{O_j}$ perçues par $O_{j,k}$ et aussi par toutes les lectures effectuées sur cette copie.

La figure 5.4 explicite le terme "écriture perçue". La copie $O_{j,3}$ ne perçoit pas l'écriture faite sur la copie $O_{j,1}$. L'exécution respecte le modèle de cohérence locale à copie unique causale. Par contre, si la copie $O_{j,3}$ avait perçu cette écriture, deux cas seraient possibles. Si elle perçoit l'écriture :

– au point 1, l'exécution respecte le modèle,

– au point 2, l'exécution ne respecte pas le modèle. En effet, on aurait $w_1(O_{j,1})3 \rightarrow_{H_{O_j}} w_2(O_{j,2})4 \rightarrow_{H_{O_j}} r_3(O_{j,3})4 \rightarrow_{H_{O_j}} r_3(O_{j,3})3$ donc $h_{O_{j,3}}$ ne serait pas légale et le modèle ne serait pas respecté.

Définition 5.5 : *Modèle de cohérence locale à copies divergentes causale*

Soit $\widehat{h}'_{O_{j,k}} = (h'_{O_{j,k}}, \rightarrow_{H_{O_j}})$ tel que $h'_{O_{j,k}}$ est défini comme :
– l'ensemble des écritures de H_{O_j} perçues par $O_{j,k}$ et
– toutes les lectures effectuées sur la copie $O_{j,k}$.

Un modèle de cohérence locale à copies divergentes causale garantit que $\forall\ O_{j,k}$, $\widehat{h}'_{O_{j,k}}$ est légale.

Le modèle de cohérence locale à copies divergentes FIFO.

Nous définissons un second modèle où les modifications faites sur une copie particulière sont perçues dans le même ordre sur les autres copies. L'histoire $\widehat{h}_{O_{j,k}}$ d'une copie $O_{j,k}$ est l'union des relations d'ordre des écritures perçues sur chacune des autres copies et des relations d'ordre de ses propres évènements d'accès. Il n'y a aucune garantie sur la façon dont est effectuée cette union.

Ce modèle diffère du modèle de cohérence locale causale car certaines transitivités dues à la relation de causalité ne sont pas considérées ici. Le modèle de cohérence locale à copies divergentes FIFO n'influe que sur la relation entre deux copies alors que le modèle de cohérence locale à copie divergentes causale influe sur la relation entre toutes les copies.

Afin de définir ce modèle, il est donc nécessaire de redéfinir la transitivité de $\widehat{H}_{O_j}$ pour ne conserver que la transitivité sur une même copie. On remplace le fait qu'on a $op1 \rightarrow_{H_{O_j}} op2$ par transitivité entre copies par le fait qu'on a $op1 \rightarrow_{H_{O_j}} op2$ uniquement par transitivité sur une même copie : $\exists op_3 : (op_1 \rightarrow_{h_{O_{j,k}}} op_3)$ et $(op_3 \rightarrow_{h_{O_{j,k}}} op_2)$.

Le terme "écriture perçue" prend le même sens que pour le modèle précédent.

Définition 5.6 : *Modèle de cohérence locale à copies divergentes FIFO*

Soit $\widehat{H}_{O_j}$ défini comme dans la définition 5.2 et en remplaçant (iii) par $\exists op_3 : (op_1 \rightarrow_{h_{O_{j,k}}} op_3)$ et $(op_3 \rightarrow_{h_{O_{j,k}}} op_2)$.
Soit $\widehat{h}'_{O_{j,k}} = (h'_{O_{j,k}}, \rightarrow_{H_{O_j}})$ tel que $h'_{O_{j,k}}$ est défini comme :
– l'ensemble des écritures de H_{O_j} perçues par $O_{j,k}$ et
– toutes les lectures effectuées par la copie $O_{j,k}$.

Un modèle de cohérence locale à copies divergentes FIFO garantit que $\forall O_{j,k}$, $\widehat{h}'_{O_{j,k}}$ est légale.

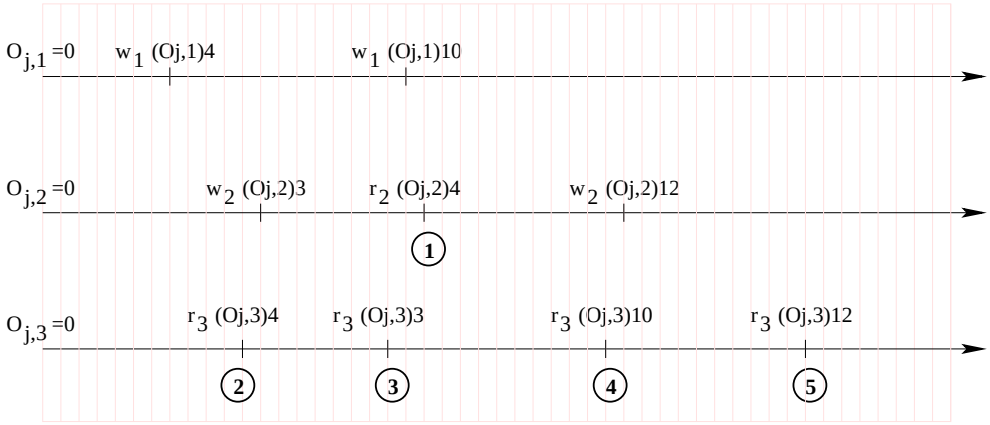


FIG. 5.5 – Modèle de cohérence locale à copies divergentes FIFO

Exemples d'exécution. La figure 5.5 présente une exécution sur un objet logique O_j composé de trois copies tel qu'à $t = 0$, $O_{j,1} = O_{j,2} = O_{j,3} = 0$ et $\widehat{H}_{O_j} = (\emptyset, \rightarrow_{H_{O_j}})$. L'exécution selon le temps physique est la suivante : $w_1(O_{j,1})4$, $r_3(O_{j,3})4$, $w_2(O_{j,2})3$, $r_3(O_{j,3})3$, $w_1(O_{j,1})10$, $r_2(O_{j,2})4$, $r_3(O_{j,3})10$, $w_2(O_{j,2})12$, $r_3(O_{j,3})12$.

Cette exécution satisfait le modèle de cohérence locale à copies divergentes FIFO. L'ordre partiel $\widehat{h}_{O_{j,3}}$=($w_1(O_{j,1})4$, $r_3(O_{j,3})4$, $w_2(O_{j,2})3$, $r_3(O_{j,3})3$, $w_1(O_{j,1})10$, $r_3(O_{j,3})10$, $w_2(O_{j,2})12$ et $r_3(O_{j,3})12$) est légal et aux points 2 et 4 (respectivement 3 et 5) la copie $O_{j,3}$ perçoit les écritures sur la copie $O_{j,1}$ (respectivement $O_{j,2}$) dans leur ordre d'exécution, satisfaisant donc le point (iii) de $\widehat{H}_{O_j}$. Cette exécution satisfait également le modèle de cohérence locale à copie unique causale. D'après le point 1, $O_{j,2}$ voit d'abord 3 puis 4 et la copie $0_{j,3}$ 4 puis 3 ce qui respecte le modèle car $w_1(O_{j,1})4$ et $w_2(O_{j,2})3$ sont concurrentes.

Le modèle de cohérence locale à copies divergentes à la longue.

La divergence, poussée à l'extrême, peut être de considérer qu'il n'y a aucune garantie sur $\widehat{H}_{O_j}$. Tout événement d'accès sur une copie $O_{j,k}$ peut advenir même si $\forall k'$, $k \neq k'$ on a $\widehat{h}_{O_{j,k}} \neq \widehat{h}_{O_{j,k'}}$. On peut ainsi imaginer d'autres garanties conduisant à différentes variantes de ce modèle.

Protocoles de la littérature. Les protocoles implantant les modèles à copies divergentes sont largement utilisés dans le cadre des mémoires partagées réparties pour obtenir des cohérences causale, PRAM, à l'entrée ou au relâchement (section 2.4). Nous reviendrons sur les protocoles de cohérence locale à utiliser afin de mettre en œuvre ces protocoles des MPR dans le chapitre suivant (chapitre 6).

5.2.3 Modèles à copies convergentes avec lecture sur les copies divergentes

Pour de nombreuses applications accéder des copies non à jour n'est pas un problème et permet d'augmenter les performances et de faciliter le passage à l'échelle (chapitre 2, section 2.1). Cependant certaines applications ont besoin qu'à un certain moment les différentes copies convergent vers le même état.

Les modèles à copies divergentes de la section précédente (section 5.2.2) posent des garanties sur l'ordre des opérations. On peut également considérer une métrique [PL91, WYP97] sur les états de l'information permettant de quantifier la divergence entre deux états d'une même information dupliquée. Grâce à la quantification de cette divergence, il est possible de limiter la divergence entre les copies en posant des conditions de délais, de périodicité, de moment précis dans le temps, de version, numériques, sur l'objet ou des évènements (voir section 2.1.3) sur le déclenchement du processus de synchronisation.

Définition 5.7 : *Fonction de quantification de la divergence* $\mu_{O_{j,k}}(O_{j,k'})$

$\mu_{O_{j,k}}(O_{j,k'})$ est la fonction quantifiant la divergence de l'état de la copie $O_{j,k}$ par rapport à la copie $O_{j,k'}$.

La fonction μ permet ainsi de calculer le temps écoulé depuis la dernière synchronisation de $O_{j,k}$ par rapport à $O_{j,k'}$, le nombre de modifications faites sur $O_{j,k'}$, les degrés de divergence entre une partie des états des copies (par exemple 10% de différence), le nombre de fois qu'a été modifié $O_{j,k'}$, etc.

La fonction *ConditionValide*, appliquée sur le résultat de la fonction μ, pose une limite sur cette divergence : pas plus de 10 minutes entre les synchronisations, pas plus de 4 modifications sur la copie $O_{j,k}$, etc. Cette fonction, spécifique à chaque modèle, renvoie faux si la divergence dépasse la limite définie, vrai sinon. Ainsi, on obtient différents modèles de cohérence locale suivant la limite fixée par cette fonction.

Nous distinguons deux types de modèle de cohérence locale à copies convergentes : un premier type autorisant uniquement des lectures sur les copies divergentes et un second autorisant des écritures sur les copies divergentes (section 5.2.4).

Pour les modèles à copies convergentes avec lecture sur les copies divergentes, il existe une copie, nommée copie non divergente ($O_{j,cnd}$). Cette copie reflète l'histoire de référence $\widehat{h}_{O_{j,cnd}}$ car elle est la seule acceptant les écritures, les mises à jour étant envoyées par la suite aux autres copies. Les copies divergentes présentent une histoire $\widehat{h}_{O_{j,k}}$ pouvant être différente de $\widehat{h}_{O_{j,cnd}}$ tant que la condition n'est pas violée. Lorsque ces copies divergent, leur histoire $\widehat{h}_{O_{j,k}}$ est seulement en retard par rapport à l'histoire $\widehat{h}_{O_{j,cnd}}$. Une fois, ou avant, que la condition ne soit violée, les copies $O_{j,k}$ doivent récupérer l'histoire $h_{O_{j,cnd}}$.

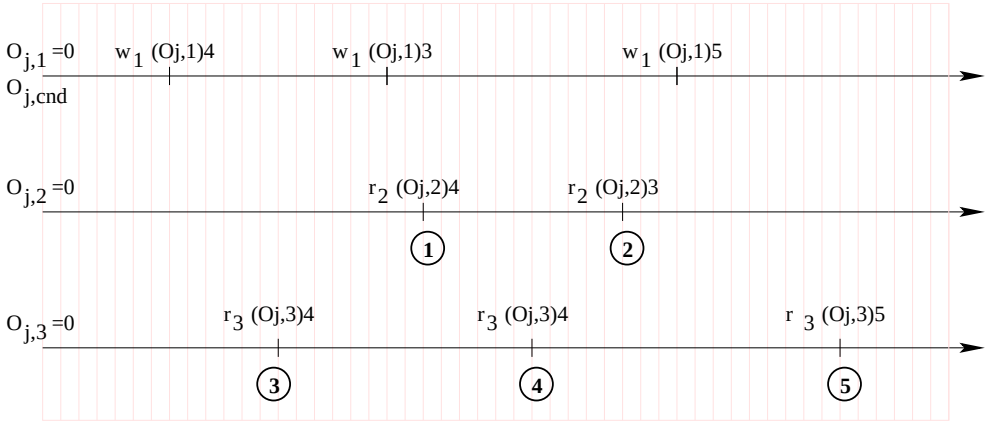


FIG. 5.6 – Modèle à copies convergentes avec lecture sur les copies divergentes

Définition 5.8 : *Modèle de cohérence locale à copies convergentes avec lecture sur les copies divergentes*

Soit $O_{j,cnd}$ la copie non divergente en lecture écriture.
Soit $O_{j,k} \in$ {copies divergentes} en lecture seule.
Soit $\widehat{h}'_{O_{j,k}} = (h'_{O_{j,k}}, \rightarrow_{H_{O_j}})$, tel que $h'_{O_{j,k}}$ est défini comme
– toutes les lectures effectuées par la copie $O_{j,k}$ et
– l'ensemble des écritures de $h_{O_{j,cnd}}$ perçues par $O_{j,k}$.

Un modèle de cohérence locale à copies convergentes avec lecture sur les copies divergentes garantit que $\forall O_{j,k} \in$ {copies divergentes}, $\widehat{h}'_{O_{j,k}}$ est légale et $ConditionValide(\mu_{O_{j,k}}(O_{j,cnd}))$.

Autrement dit, étant donné que (1) $O_{j,cnd}$ est la seule copie faisant des écritures, (2) les histoires $\widehat{h}'_{O_{j,k}}$ sont définies en respectant $\rightarrow_{h_{O_{j,cnd}}}$ et (3) ces histoires doivent être légales, toutes les copies $O_{j,k}$ divergentes vont percevoir les écritures faites sur $O_{j,cnd}$ dans leur ordre d'exécution. De plus, toutes ces copies divergentes doivent percevoir les mises à jours venant de $O_{j,cnd}$ avant que la condition limite ne soit violée. En fait ce modèle est une combinaison du modèle à copie unique séquentiel et du modèle à copie unique atomique. Pour le modèle atomique la perception des écritures est immédiate, pour le modèle séquentiel il n'y a aucune limite dans le temps logique et pour ce modèle la perception des écritures est limitée par une condition (temporelle ou non).

Ce modèle n'est pas du type modèle à copie unique, car il n'y a pas unicité dans la vision des copies. Il assure (1) qu'une lecture sur la copie de référence $O_{j,cnd}$ se fait sur la dernière écriture par rapport au temps physique, et (2) qu'une lecture sur une copie divergente se fait sur la dernière écriture acceptable, c'est à dire sans violer la fonction *conditionValide*.

Exemples d'exécution. La figure 5.6 présente une exécution sur un objet logique O_j composé de trois copies tel qu'à $t = 0$, $O_{j,1} = O_{j,2} = O_{j,3} = 0$ et $\widehat{H}_{O_j} = (\emptyset, \rightarrow_{H_{O_j}})$ et $O_{j,1} = O_{j,cnd}$. L'exécution est la suivante par rapport au temps physique : $w_1(O_{j,1})4$, $r_3(O_{j,3})4$, $w_1(O_{j,1})3$, $r_2(O_{j,2})4$, $r_3(O_{j,3})4$, $r_2(O_{j,2})3$, $w_1(O_{j,1})5$, et $r_3(O_{j,3})5$.

$\mu_{O_{j,k}}(O_{j,cnd})$ calcule le nombre d'écritures faites sur $O_{j,cnd}$ depuis la dernière synchronisation de $O_{j,k}$. La fonction *ConditionValide* renvoie vrai si ce nombre d'écritures est inférieur à 2.

L'exécution de la figure 5.6 respecte le modèle de cohérence locale à copies convergentes avec lecture sur les copies divergentes où la fonction *ConditionValide* est celle définie ci-dessus. Aux points 1 et 2 (respectivement 3 et 4) les lectures faites sur la copie $O_{j,2}$ (respectivement $O_{j,3}$) respectent le modèle de cohérence locale défini. Au point 5, si le processus P_3 lisait la valeur 4 sur la copie $O_{j,3}$ alors l'exécution ne respecterait plus le modèle de cohérence défini car la fonction *ConditionValide* renverrait faux. Par contre, si P_3 lit 3 ou 5, le modèle est respecté. Notons que si la copie $O_{j,3}$ renvoie 4 suite à une opération de lecture au point 5, cette exécution est correcte pour un modèle à copie unique séquentiel.

Protocoles de la littérature. Ce modèle est orienté SGBDR où les copies doivent convergeer afin d'assurer la cohérence du système. Des protocoles comme la duplication paresseuse maître/esclaves ou l'ϵ-sérialisabilité s'appuient sur ces modèles. Ce type de modèle est également utilisé dans le contexte du travail collaboratif. Nous reviendrons sur les protocoles de cohérence locale à utiliser afin de mettre en œuvre ces protocoles dans le chapitre suivant (chapitre 6).

5.2.4 Modèles à copies convergentes avec écriture sur les copies divergentes

Pour ce quatrième type de modèle de cohérence locale, toutes les copies sont potentiellement divergentes. Comme pour le modèle précédent, on limite cette divergence à l'aide de conditions.

Malheureusement, les écritures et les lectures pouvant se faire sur des copies non à jour, il n'existe pas de copie possédant à tout moment l'histoire de référence sur laquelle se synchroniser. Lorsqu'une copie $O_{j,k}$ diverge, elle construit sa propre histoire $\widehat{h}_{O_{j,k}}$ différente de celles des autres copies. Lorsque la convergence de plusieurs copies vers un même état est nécessaire, l'histoire locale de chaque copie doit être réécrite en prenant en compte les histoires locales des autres copies afin de construire ensemble une histoire commune.

Définition 5.9 : *Fonction Réconciliation*$(\widehat{h}_{O_{j,k}}, \widehat{h}_{O_{j,k'}})$

La fonction *Réconciliation*$(\widehat{h}_{O_{j,k}}, \widehat{h}_{O_{j,k'}})$ construit une histoire commune à partir de deux histoires divergentes $\widehat{h}_{O_{j,k}}$ et $\widehat{h}_{O_{j,k'}}$

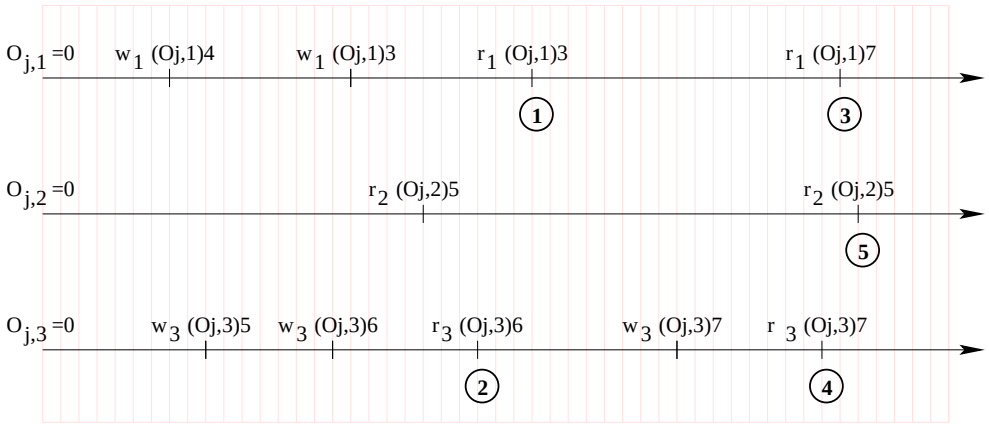


FIG. 5.7 – Modèle à copies convergentes avec écriture sur les copies divergentes

Afin de construire l'histoire commune à deux histoires divergentes, il est nécessaire de faire appel à une fonction de réconciliation. Ainsi, la fonction *Réconciliation*$(\widehat{h}_{O_{j,k}}, \widehat{h}_{O_{j,k'}})$ permet, à partir de l'histoire de deux copies $O_{j,k}$ et $O_{j,k'}$, de construire une nouvelle histoire. Nous extrayons cette fonction du modèle car l'information nécessaire à la réconciliation de deux copies peut être détenue par l'application, cette information n'étant pas propre au modèle. Cette fonction "installe" une nouvelle histoire en lieu et place des histoires $\widehat{h}_{O_{j,k}}$ et $\widehat{h}_{O_{j,k'}}$.

Définition 5.10 : *Modèle de cohérence locale à copies convergentes avec écriture sur les copies divergentes*

Soit $\widehat{h}'^{O_{j,k'}}_{O_{j,k}} = (h'^{O_{j,k'}}_{O_{j,k}}, \rightarrow_{H_{O_j}})$, tel que $h'^{O_{j,k'}}_{O_{j,k}}$ est défini comme :
- toutes les lectures effectuées par la copie $O_{j,k}$ et
- l'ensemble des écritures sur $O_{j,k'}$ perçues par $O_{j,k}$.

Un modèle de cohérence locale à copies convergentes avec écriture sur les copies divergentes garantit que $\forall O_{j,k}, O_{j,k'}$, (1) $ConditionValide(\mu_{O_{j,k}}(O_{j,k'}))$ et (2) qu'à certains moments *Réconciliation*$(\widehat{h}'^{O_{j,k'}}_{O_{j,k}}, \widehat{h}'^{O_{j,k}}_{O_{j,k'}})$ est légale.

Le point (1) indique que deux copies ne divergent pas au delà d'une certaine limite fixée par une certaine condition. Le point (2) indique quant à lui que la réécriture de l'histoire de deux copies est possible à certains moments grâce à la fonction *Réconciliation* et que cette nouvelle histoire est légale. Le moment de cette réécriture n'est pas explicite : il a lieu avant que la condition énoncée par le point (1) ne soit violée.

Exemples d'exécution. La figure 5.7 présente une exécution respectant le modèle à copies convergentes avec écritures sur les copies divergentes où :

- $\mu_{O_{j,k}}(O_{j,k'})$ calcule le nombre d'écritures faites sur $O_{j,k'}$ depuis la dernière synchronisation de $O_{j,k}$.
- *ConditionValide* renvoie vrai si ce nombre d'écriture est inférieur à 3.
- *Réconciliation* réordonnance les opérations suivant le temps physique donné par une horloge globale. En cas d'écritures simultanées, on donne priorité à l'écriture de la copie ayant le plus petit indice k.

L'objet logique O_j se compose de trois copies tel qu'à $t = 0$, $O_{j,1} = O_{j,2} = O_{j,3} = 0$ et $\widehat{H}_{O_j} = (\emptyset, \rightarrow_{H_{O_j}})$. L'exécution est la suivante : $w_1(O_{j,1})4$, $w_3(O_{j,3})5$, $w_3(O_{j,3})6$, $w_1(O_{j,1})3$, $r_2(O_{j,2})5$, $r_3(O_{j,3})6$, $r_1(O_{j,1})3$, $w_3(O_{j,3})7$, $r_3(O_{j,3})7$, $r_1(O_{j,1})7$ et $r_2(O_{j,2})5$.

Au point 1 (respectivement 2) le processus P_1 (respectivement 3) lit la valeur qu'il a écrit sur la copie $O_{j,1}$ (respectivement $O_{j,3}$). Par contre au point 3 et 4 les copies 1 et 3 renvoient toutes les deux la valeur 7 suite à une lecture. En effet, elles se sont synchronisées avant que la fonction *ConditionValide* soit fausse. La fonction réconciliation a réécrite l'histoire des deux copies en : $w_1(O_{j,1})4$, $w_3(O_{j,3})5$, $w_3(O_{j,3})6$, $r_3(O_{j,3})6$, $w_1(O_{j,1})3$, $r_1(O_{j,1})3$, $w_3(O_{j,3})7$, $r_3(O_{j,3})7$, $r_1(O_{j,1})7$. Au point 5 la copie $O_{j,2}$ renvoie encore la valeur 5 suite à une lecture car la fonction *ConditionValide* renvoie vrai. Par contre si lors de sa première lecture elle renvoyait une écriture faite sur la copie $O_{j,1}$, l'exécution ne respecterait plus le modèle (elle devrait renvoyer 7 lors de sa deuxième écriture).

Protocoles de la littérature. Des protocoles comme multi-airline reservation, ceux utilisés dans un contexte mobile ou ceux supportant les partitions réseau s'appuient sur ce modèle. Ce type de modèle est également utilisé dans le contexte du travail collaboratif. Nous reviendrons sur les protocoles de cohérence locale utilisés afin de mettre en œuvre ces protocoles de la littérature dans le chapitre suivant (chapitre 6).

5.2.5 Conclusion

Les quatre types de modèles présentés dans cette section modélisent la partie maximale partagée par l'aspect duplication. Cependant, afin de modéliser l'ensemble des protocoles existant de la littérature, il est possible de combiner les modèles de cohérence locale présentés ci-dessus afin d'obtenir des modèles plus complexes. L'ensemble des copies d'un même objet est partitionné en plusieurs sous-ensembles, chacun respectant un modèle particulier. Les modèles présentés précédemment ne sont que la factorisation maximale des protocoles existants. Il s'agit du dénominateur commun et par ce fait sont pris en charge par RS2.7.

La figure 5.8 décrit un protocole utilisé dans un contexte comportant des bases de données mobiles et des serveurs dupliqués sur un réseau fixe. Le protocole contrôlant les copies présentes sur les serveurs respecte le modèle de cohérence locale à copie unique atomique. Les copies présentes sur les hôtes mobiles sont contrôlées soit par un protocole respectant le modèle de cohérence locale à copies convergentes avec écriture sur les copies divergentes, soit celui où les copies divergentes sont uniquement consultables. Cette approche permet de bien

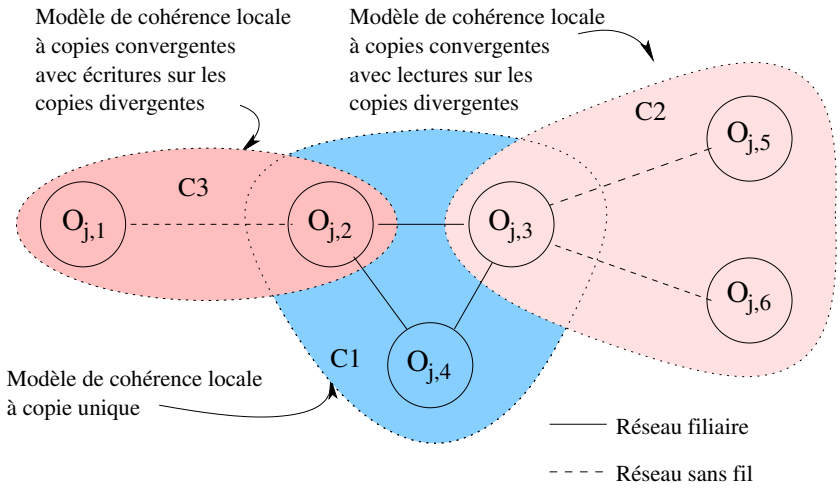


FIG. 5.8 – Modèles de cohérence locale pour contexte mobile

isoler tous les aspects conduisant ainsi à une certaine adaptabilité et réutilisabilité. Ainsi, on a trois groupes de copies pour un même objet dupliqué :

- C_1 le groupe des copies présentes sur les différents serveurs du réseau fixe. Ce groupe est géré suivant le modèle de cohérence locale à copie unique atomique,
- C_2 constitué des copies mobiles consultables et d'une copie présente sur le réseau fixe et
- C_3 constitué des copies mobiles modifiables et d'une copie présente sur le réseau fixe.

Les différents groupes sont connectés par les copies qu'ils ont en commun. Par exemple, lorsque le groupe C_3 va se synchroniser afin de converger vers une histoire commune, il va également synchroniser la copie du groupe C_1 qui à son tour va synchroniser les copies du groupe C_1, pouvant engendrer également la synchronisation du groupe C_2 par ricochet.

On peut également subdiviser les groupes C_2 et C_3 en plusieurs sous-groupes avec différentes fonctions *ConditionValide* et *Réconciliation*.

Du point de vue algorithmique cette approche nous paraît satisfaisante. Cependant, il nous semble nécessaire d'expérimenter certaines combinaisons de modèle de cohérence locale, notamment celles avec des modèles à copies divergentes.

5.3 Modèle de cohérence locale et séparation des considérations

La mise en œuvre des modèles de cohérence locale est affectée par la concurrence et les fautes. Ainsi, il est nécessaire de définir les interactions avec ces aspects. Ces aspects ne font

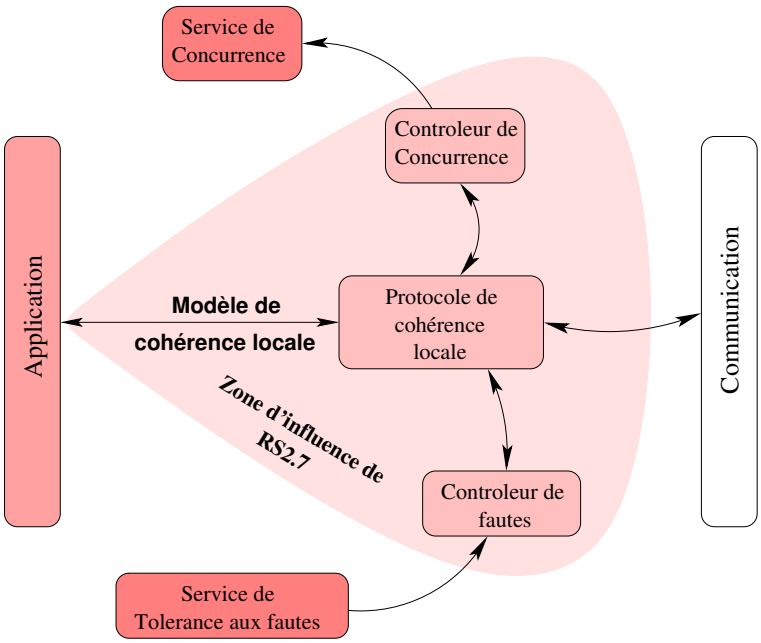


FIG. 5.9 – Modèle de cohérence locale et séparation des aspects

pas partie des protocoles de cohérence locale afin de maximiser la réutilisabilité et l'adaptation de RS2.7. Dans un contexte avec concurrence (resp. risque de fautes) et suivant le modèle de cohérence locale à mettre en œuvre, le protocole de cohérence locale interagit d'une manière particulière avec un service de gestion de la concurrence (resp. tolérance aux fautes).

Ainsi, nous isolons deux composants dédiés à la gestion des interactions entre le protocole de cohérence locale, la concurrence et les fautes. Ces deux composants sont le contrôleur de concurrence et le contrôleur de fautes. Ils font partie du protocole de cohérence locale, car ils contiennent toute l'information engendrée par la duplication, et nécessaire pour contrôler les interactions avec ces deux aspects. Par contre, ils ne mettent pas en œuvre ces aspects. Cela reste à la charge d'un service de contrôle de la concurrence et d'un service de tolérance aux fautes indépendant du fait qu'il y ait duplication ou non (figure 5.9).

Nous commençons par présenter les intéractions avec la concurrence (section 5.3.1) puis celles avec les fautes (section 5.3.2).

5.3.1 Contrôleur de concurrence

Il existe de nombreux protocoles de gestion de la concurrence pouvant être classés selon deux critères : le mode d'accès aux objets et la mise en œuvre. On trouve quatre modes d'accès aux objets : (1) un écrivain à un moment donné et plusieurs lecteurs lorsqu'il n'y a

pas d'écrivain, (2) plusieurs écrivains en même temps et plusieurs lecteurs lorsqu'il n'y a pas d'écrivain, (3) un écrivain et plusieurs lecteurs en même temps et (4) plusieurs écrivains et lecteurs en même temps. Pour la mise en œuvre, on distingue les protocoles pessimistes et les protocoles optimistes.

En présence d'objets dupliqués, le problème est légèrement différent. En effet, la gestion de la concurrence doit se faire sur un ensemble de copies d'un même objet et non plus sur un objet. Ainsi, certains protocoles de la littérature gèrent la concurrence sur l'ensemble des copies, d'autres sur un sous-ensemble de copies et certains sur une unique copie.

Il est nécessaire de ne pas déléguer des considérations propres à la duplication au service de gestion de la concurrence. Ainsi, le protocole de cohérence locale prend les décisions concernant la gestion de la concurrence sur l'objet logique, mais ne s'occupe pas de sa mise en œuvre. Ces décisions sont prises par le contrôleur de concurrence, composant faisant partie du protocole de cohérence locale. Celui-ci présente l'interface suivante au protocole de cohérence locale :

Interface du contrôleur de concurrence :

- `boolean readIntention` [2] `({Replica} rs)` informe le contrôleur de concurrence d'un accès en lecture sur un groupe de copies `rs`.
- `boolean readIntention({Replica} rs, Field f)` informe le contrôleur de concurrence d'un accès en lecture sur le champ `f` du groupe de copies `rs`.
- `boolean writeIntention({Replica} rs)` informe le contrôleur de concurrence d'un accès en écriture sur une un groupe de copies `rs`.
- `boolean writeIntention({Replica} rs, Field f)` informe le contrôleur de concurrence d'un accès en écriture sur le champ `f` du groupe de copies `rs`.
- `boolean readCompletion({Replica} rs)` informe le contrôleur de concurrence de la terminaison d'un accès en lecture sur un groupe de copies `rs`.
- `boolean readCompletion({Replica} rs, Field f)` informe le contrôleur de concurrence de la terminaison d'un accès en lecture sur le champ `f` du groupe de copies `rs`.
- `boolean writeCompletion({Replica} rs)` informe le contrôleur de concurrence de la terminaison d'un accès en écriture sur un groupe de copies `rs`.
- `boolean writeCompletion({Replica} rs, Field f)` informe le contrôleur de concurrence de la terminaison d'un accès en écriture sur le champ `f` du groupe de copies `rs`.

∎

Suite à une opération d'intention sur un groupe de copies, le contrôleur de concurrence met en œuvre un protocole de gestion de la concurrence particulier sur cet ensemble d'objets : il bloque l'ensemble des copies, un sous-ensemble, etc. Pour cela, il s'appuie sur un service de contrôle de concurrence qui voit chaque copie comme un objet classique (non dupliqué). Les opérations de terminaison (completion) indiquent au contrôleur de concurrence de terminer l'accès concurrent sur un groupe de copies. Dans le cas d'un protocole optimiste de gestion de la concurrence, lors de cette phase, le contrôleur de concurrence peut vérifier qu'il n'y a pas eu d'accès concurrent sur l'ensemble des copies. Toutes ces fonctions renvoient `true` si

[2] readIntention ou writeIntention ne se réfèrent pas aux lectures ou aux écritures par intention utilisés dans les protocoles de verouillage des SGBD.

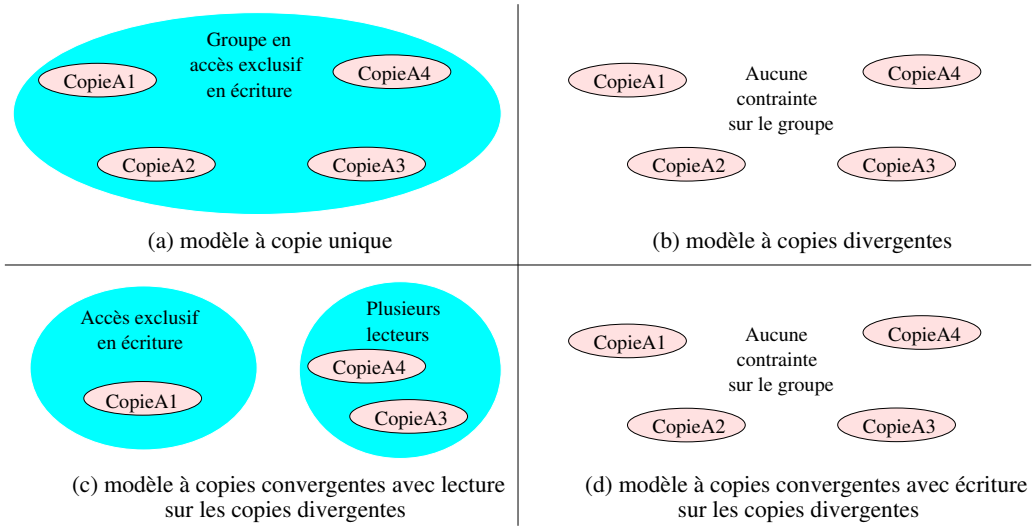


FIG. 5.10 – Rôle du gestionnaire de contrôle de concurrence du protocole de cohérence locale

l'intention ou la terminaison de l'opération c'est déroulé convenablement.

Selon le type de modèle de cohérence locale à mettre en œuvre, nous pouvons caractériser le rôle du contrôleur de concurrence (figure 5.10).

Pour le premier type de modèle de cohérence, les modèles à copie unique (section 5.2.1), le contrôleur de concurrence doit garantir qu'il ne peut y avoir qu'un écrivain à un moment donné sur un sous-ensemble particulier des copies, ou bien, plusieurs lecteurs en même temps (figure 5.10 a). En effet les modèles à copie unique doivent garantir un ordre total sur les écritures. Il peut être implanté de diverses façons : centralisé, réparti, majoritaire, etc. En ce qui concerne le service de contrôle de la concurrence celui-ci peut également être mis en œuvre de diverses façons : approche pessimiste ou optimiste.

Pour le deuxième type de modèle, les modèles à copies divergentes (section 5.2.2), aucun contrôle sur l'accès concurrent des copies n'est requis (figure 5.10 b). Il peut y avoir plusieurs copies accédées en écriture et en lecture en même temps.

Pour le troisième type de modèle, les modèles à copies convergentes avec lecture sur les copies divergentes (section 5.2.3), le contrôleur de concurrence locale doit permettre des écritures sur la copie modifiable et des lectures simultanées sur le groupe des copies en consultation (figure 5.10 c). De même que pour le premier type de modèle il est possible d'utiliser de nombreux protocoles afin de mettre en œuvre le contrôleur de concurrence et le service de contrôle de concurrence.

Comme pour le deuxième type de modèle, pour le quatrième type de modèle, les modèles à copies convergentes avec écriture sur les copies divergentes (section 5.2.4), il n'y a aucune

contrainte pour le contrôleur de concurrence (figure 5.10 d). Plusieurs écrivains et lecteurs sont envisageables sur différentes copies au même moment.

Nous présenterons des exemples de contrôleur de concurrence et de services de contrôle de concurrence dans le chapitre suivant (chapitre 6). Nous présenterons également l'interface d'un service de contrôle de la concurrence.

5.3.2 Contrôleur de fautes

Afin d'isoler les interactions entre la tolérance aux fautes et la duplication nous utilisons un contrôleur de fautes. Nous proposons l'interface suivante pour le contrôleur de fautes. Cette interface est utilisée par le service de tolérance aux fautes et non, comme pour la concurrence, par le protocole de duplication. Toute l'information concernant la gestion de la concurrence sur un objet logique est détenue par le protocole de cohérence locale, alors que c'est le service de tolérance qui a conscience des objets défaillants.

Interface du contrôleur de fautes :

- `dead(Replica r)` indique au contrôleur de fautes que la copie `r` est défaillante.
- `alive(Replica r)` indique au contrôleur de fautes que la copie `r` est opérationnelle.

∎

Pour le premier type de modèle, les modèles à copie unique (section 5.2.1), toute copie suspecte ne doit pas être accédée. Une faute va perturber l'ordre total. Ainsi, les copies défaillantes sont supprimées du groupe. Une fois de nouveau opérationnelles, elles doivent d'abord rattraper leur retard avant de pouvoir être accédées.

Pour le deuxième type de modèle, les modèles à copies divergentes (section 5.2.2), selon la garantie désirée il peut être acceptable que certaines copies défaillantes à nouveau opérationnelles acceptent tout de suite les événements d'accès avant de rattraper leur retard. Une histoire sur une copie en retard doit seulement être légale.

Pour le troisième type de modèle, les modèles à copies convergentes avec lecture sur les copies divergentes (section 5.2.3), il faut garantir qu'il y a toujours une copie possédant l'histoire de référence : une copie $O_{j,cnd}$. Il faut donc des mécanismes permettant d'élire une nouvelle copie modifiable. La nouvelle copie de référence peut être en retard sur l'ancienne copie de référence. Lorsqu'une copie divergente défaille aucun mécanisme n'est nécessaire. Lorsqu'elle est de nouveau opérationnelle elle doit rattraper son retard sur la copie de référence avant de réintégrer l'objet logique.

Pour le quatrième type de modèle, les modèles à copies convergentes avec écriture sur les copies divergentes (section 5.2.4), il n'y a besoin d'aucun mécanisme particulier. Une copie de nouveau opérationnelle se synchronise.

5.4 Conclusion

Dans ce chapitre, nous avons commencé par modéliser une exécution sur un objet dupliqué : une relation d'ordre sur les opérations faites sur les différentes copies d'un objet logique. Nous avons ensuite caractérisé les différents services qui peuvent être construits à partir de RS2.7 . Ces services correspondent aux différents types de modèles de cohérence locale présentés. Un modèle de cohérence locale spécifie les accès aux copies et leur perception par l'utilisateur du service. L'application, utilisatrice d'un service construit via RS2.7, détermine les propriétés qu'elle souhaite pour ses observations ; le protocole de cohérence locale met en œuvre un protocole d'accès aux copies qui garantit effectivement que les opérations effectuées satisfont les propriétés spécifiées.

Nous avons défini quatre types de modèle de cohérence locale. Les modèles à copies uniques ont en commun qu'ils offrent une vue unique de l'ensemble des copies. Toutes les copies sont d'accord sur une histoire commune. La différence entre le modèle séquentiel et le modèle atomique porte sur la relation d'ordre (prise en compte du temps logique ou physique pour cette relation d'ordre suivant le modèle). Les modèles à copies divergentes admettent que les copies soient différentes les unes des autres. Ces modèles permettent de définir certaines garanties sur l'ordre de perception des opérations entre les copies. Nous pouvons proposer d'autres modèles dans cette catégorie en ajoutant, par exemple, des conditions portant sur la divergence entre les copies grâce à la fonction *ConditionValide*. Les modèles à copies convergentes autorisent la coexistence de copies différentes les unes des autres mais avec la garantie qu'à certains moments elles vont converger vers le même état. Ces modèles fixent les limites de divergence acceptables. Les modèles à copies convergentes avec lecture sur les copies divergentes autorisent la divergence des copies uniquement consultables. Les modèles à copies convergentes avec écriture sur les copies divergentes ne discernent pas de copies particulières comme pour les modèles précédents. A certains moments une histoire commune des copies est réécrite.

A partir des différents modèles de cohérence locale, il est possible de construire des modèles plus complexes (les modèles hybrides). L'approche ainsi définie est basée sur l'identification du plus grand ensemble d'éléments communs aux protocoles de cohérence locale afin de supporter une grande variété de protocoles. Nous nous sommes attachés à factoriser au mieux la partie commune aux différents protocoles de cohérence locale dans RS2.7.

Nous avons vu également que chacun de ces modèles a des besoins particuliers en terme de gestion de la concurrence et de tolérance aux fautes (figure 5.9). Ces deux aspects ne font pas partie du protocole afin de maximiser la réutilisabilité et l'adaptation. La façon de mettre en œuvre, par exemple, la gestion de la concurrence sur un objet est indépendante du modèle de cohérence locale souhaité. Par contre l'information nécessaire à cette gestion sur l'objet logique est quant à elle complètement dépendante de l'aspect duplication. Le contrôleur de concurrence et le contrôleur de fautes du protocole de cohérence locale isolent les parties où le code propre à la duplication se trouve entremêlé au code propre à l'aspect contrôle de concurrence et à l'aspect tolérance aux fautes.

La notion de modèle de cohérence locale existe plus ou moins dans la littérature mais

n'est pas clairement définie. Ainsi on retrouve dans les protocoles de duplication existants la gestion de nombreux aspects. De plus, la plupart des travaux se sont plus attachés à classifier les protocoles suivant leur architecture (par exemple impatient/paresseux, maître-esclaves/n'importe où dans les SGBDR [GHOS96]).

Les modèles de cohérence locale à copie unique séquentiel et atomique, ainsi que les modèles de cohérence locale à copies divergentes FIFO et causale ne sont que la transposition des modèles de cohérence présentés au chapitre 2 section 2.4.2 sur une exécution sur un objet dupliqué. Nous avons redéfini une partie de ces modèles existants en ne conservant que la partie propre à la duplication. Nous n'avons pas repris les modèles de la section 2.4.3 car ceux-ci sont des protocoles de cohérence portant sur des groupes d'objets logiques intégrant des patrons d'interactions entre les copies identiques aux modèles de cohérence sans synchronisation. Les modèles à copies convergentes sont généralement orientés SGBDR et systèmes répartis dans lesquels le besoin de convergence entre les copies se fait sentir.

Ainsi, dans le chapitre suivant (chapitre 6) nous montrons comment nous construisons cette gestion de la cohérence entre les objets du système à partir des modèles de cohérence locale, c'est à dire comment nous obtenons l'adaptabilité au contexte non-fonctionnel. Ensuite (chapitre 7), nous présentons une factorisation des protocoles de duplication existants afin d'obtenir l'adaptabilité dans tout ou partie des protocoles de cohérence locale supportés.

Tout dans la nature se modèle sur la sphère, le cône et le cylindre, il faut apprendre à peindre sur ces figures simples, on pourra ensuite faire tout ce qu'on voudra.
Cézanne, Paul

Chapitre 6

Adaptabilité de RS2.7 au contexte non fonctionnel

Sommaire

es deux chapitres précédents ont présenté les différents services de duplication pouvant être obtenus à partir de RS2.7. Ce chapitre montre de quelle manière nous obtenons l'adaptabilité de RS2.7 au contexte non fonctionnel. L'objectif est de produire, à partir de RS2.7, des services de duplication utilisables dans différents contextes : transactionnels, mémoires partagées réparties, systèmes à communication de groupe, etc. Pour cela, il est nécessaire de définir les intéractions entre les services de duplication et d'autres aspects portant sur les objets du système, objets pas nécessairement dupliqués.

Dans une première section (section 6.1), nous présentons la notion de modèle de cohérence globale, le contrat donné au programmeur d'application décrivant comment la mémoire ou les données apparaissent pour l'application. Nous poursuivons par une décomposition de la cohérence globale (section 6.2). Cette décomposition, nous permet de montrer comment la notion de modèle de cohérence locale participe à la mise en œuvre de ces divers modèles de cohérence globale (section 6.3). Nous verrons qu'un même modèle de cohérence locale peut participer à plusieurs modèles de cohérence globale. Ainsi, un protocole de cohérence locale mis en œuvre par RS2.7 peut être utilisé dans divers protocoles de cohérence globale. Nous terminons ce chapitre par une conclusion (section 6.4).

6.1 De la nécessité de séparer cohérence globale et locale

RS2.7 permet d'obtenir des protocoles implantant les quatre types de modèles de cohérence locale présentés dans le chapitre précédent (chapitre 5, section 5.2) et des modèles hybrides. Cependant, le programmeur d'une application s'appuie sur un modèle de cohérence globale cachant la répartition, la concurrence, la duplication, la tolérance aux fautes, etc.

Dans cette section, nous commençons par présenter la notion de modèle de cohérence globale largement étudiée dans la littérature (section 6.1.1), puis nous montrons que, contrairement aux travaux existants, il est nécessaire d'aller plus loin afin d'obtenir l'adaptabilité de RS2.7 au contexte non fonctionnel en isolant la duplication à l'intérieur des protocoles de cohérence globale (section 6.1.2).

6.1.1 Modèle de cohérence globale

Un modèle de cohérence globale[1] est une définition plus ou moins formelle de comment la mémoire (ou les données dans le contexte SGBDR) apparait pour l'application. Un modèle de cohérence peut être considéré comme l'ensemble des contraintes prémunissant l'application contre les incorrections engendrées par la concurrence, la duplication, la tolérance aux fautes, etc. C'est un contrat donné au programmeur d'application lui précisant comment se comporte la mémoire (ou les données).

Définition 6.1 : *Modèle de cohérence globale*

Un modèle de cohérence globale spécifie l'ordre dans lequel les accès d'un processus à la mémoire (ou sur les données) sont observés par l'ensemble des autres processus.

Définition 6.2 : *Protocole de cohérence globale*

Un protocole de cohérence globale est une mise en œuvre particulière d'un modèle de cohérence globale. Il assure l'ordre sur les accès spécifié par le modèle de cohérence globale, ordre remis en cause par la répartition, la concurrence, la duplication et/ou la tolérance aux fautes suivant le contexte.

Un modèle de cohérence globale donné peut être mis en œuvre par différents protocoles de cohérence globale. Un protocole de cohérence locale gère la cohérence d'un ensemble de copies d'un même objet logique, alors qu'un protocole de cohérence globale s'occupe de la cohérence de l'ensemble des objets du système.

On retrouve la notion de modèle de cohérence globale dans de nombreux domaines. Dans

[1] Nous parlons de cohérence globale (*consistency* en anglais) car portant sur l'ensemble du système.

les mémoires partagées réparties, un modèle de cohérence globale définit la valeur devant être retournée par un évènement de lecture durant l'exécution des programmes parallèles. Dans ce contexte, de nombreux modèles existent : séquentiel, causal, PRAM, cohérence faible, cohérence à l'entrée ou bien encore cohérence au relâchement (voir chapitre 2, section 2.4). Les protocoles utilisés dans ce domaine prennent en charge la duplication (l'ordre d'accès entre les différentes copies) et l'ordre d'accès entre les différents objets.

Dans les systèmes répartis à échange de messages, les protocoles de duplication active, passive, etc. (chapitre 2, section 2.2), sont des protocoles de cohérence globale. Ils assurent une mise en œuvre de la gestion de la duplication, mais également de la gestion de la concurrence, de la tolérance aux fautes et de l'ordre d'accès entre les objets. Ils ne gèrent pas uniquement la cohérence des objets logiques, mais ils veillent également au maintient de la cohérence entre les objets. Le plus souvent, ces protocoles mettent en œuvre le modèle de cohérence globale séquentiel afin d'offrir la tolérance aux fautes.

Le contexte des bases de données réparties a lui aussi donné lieu à l'élaboration de modèles de cohérence globale, le plus populaire étant la sérialisabilité. Le support de transactions ACID constitue un protocole implantant ce modèle de cohérence. Dans le cas des bases de données dupliquées le modèle de cohérence globale est souvent la sérialisabilité sur une copie (voir chapitre 2, section 2.3). Ce critère de correction assure qu'une exécution sur des données dupliquées est équivalente à une exécution sur une seule copie et que l'exécution des transactions est sérialisable. Dans ce contexte, le protocole de cohérence globale doit prendre en charge le contrôle de la concurrence, la tolérance aux fautes, l'ordre d'accès entre des groupes d'opérations ainsi que la duplication. On trouve également d'autres modèles de cohérence globale tels la quasi-sérialisabilité [DE89], la M-sérialisabilité [RKC93], l'ϵ-sérialisabilité [PL91, RP95], la sérialisabilité causale [TK97], etc.

6.1.2 De la nécessité d'isoler la duplication

Un modèle de cohérence globale permet de dégager le programmeur d'application de la gestion de la répartition, la concurrence, la tolérance aux fautes et/ou la duplication. Ainsi, une claire séparation entre ce qui est propre à l'application et ce qui relève du système est établie.

Cependant, s'il est possible d'obtenir l'adaptabilité du système par rapport à l'application, on n'obtient pas l'adaptabilité de chacun des aspects intervenant dans les protocoles de cohérence globale. Afin d'obtenir cette adaptabilité, il nous semble nécessaire de séparer et de définir les intéractions entre chacun des aspects intervenant dans la construction d'un protocole de cohérence globale et notamment la duplication. Dans la littérature, très peu de travaux adoptent cette approche (chapitre 3, section 3.5), l'ensemble des aspects non fonctionnels est géré par le protocole de cohérence globale. Ainsi, si l'on veut rajouter la gestion de la duplication à un protocole de cohérence globale gérant déjà la répartition et la concurrence, ce protocole doit être entiérement modifié. Il n'existe pas d'abstraction de chacune de ces propriétés permettant de raisonner indépendamment de l'une de l'autre.

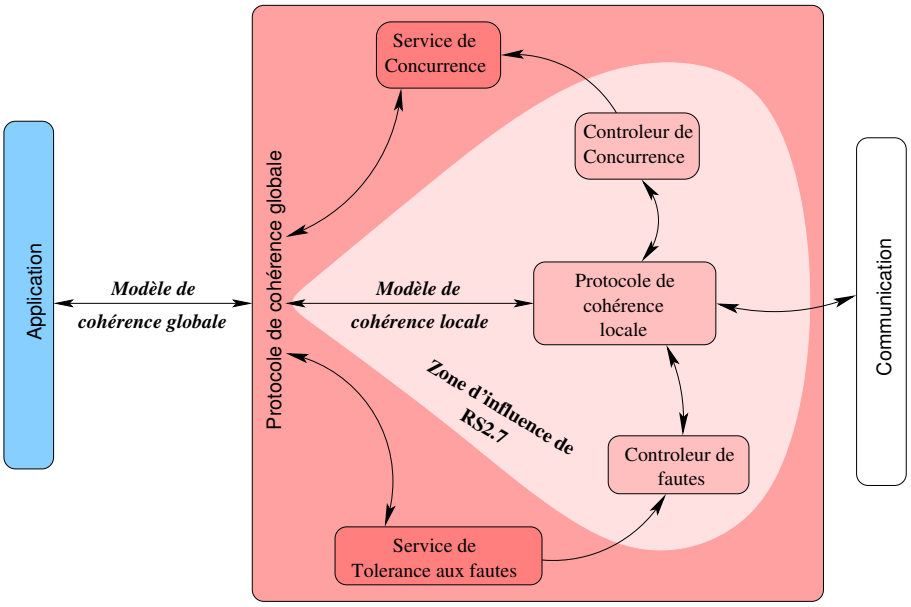


FIG. 6.1 – Isolation de la duplication au sein de la cohérence globale

Avec la notion de modèle de cohérence locale (chapitre 5, section 5.2), nous donnons une abstraction pour la duplication. Cependant, il est clair que RS2.7 a des intéractions avec le controle de la concurrence, la tolérance aux fautes et la couche communication pour mettre en œuvre à la fois les modèles de cohérence locale (chapitre 5) et le protocole de cohérence globale (figure 6.1). Cette reformulation de la problématique de la cohérence permet de préciser la séparation des rôles entre un système de duplication et les applicatifs qui l'utilisent.

6.2 Décomposition d'un protocole de cohérence globale

Cette section présente la décomposition des protocoles de cohérence globale. Cette décomposition extrait chacun des aspects qu'un tel protocole doit gérer. Notre objectif n'étant pas de proposer un canevas de service de cohérence globale, la décomposition présentée ici porte sur les aspects ayant de fortes intéractions avec le protocole de cohérence locale : comment se place la duplication dans un protocole de cohérence globale.

Un protocole de cohérence globale (section 6.2.1) intéragit avec l'application et se décompose en (figure 6.2) : gestion de la concurrence (section 6.2.2), gestion de la tolérance aux fautes (section 6.2.3), gestion des communications (section 6.2.4) et gestion de la duplication (section 6.2.5).

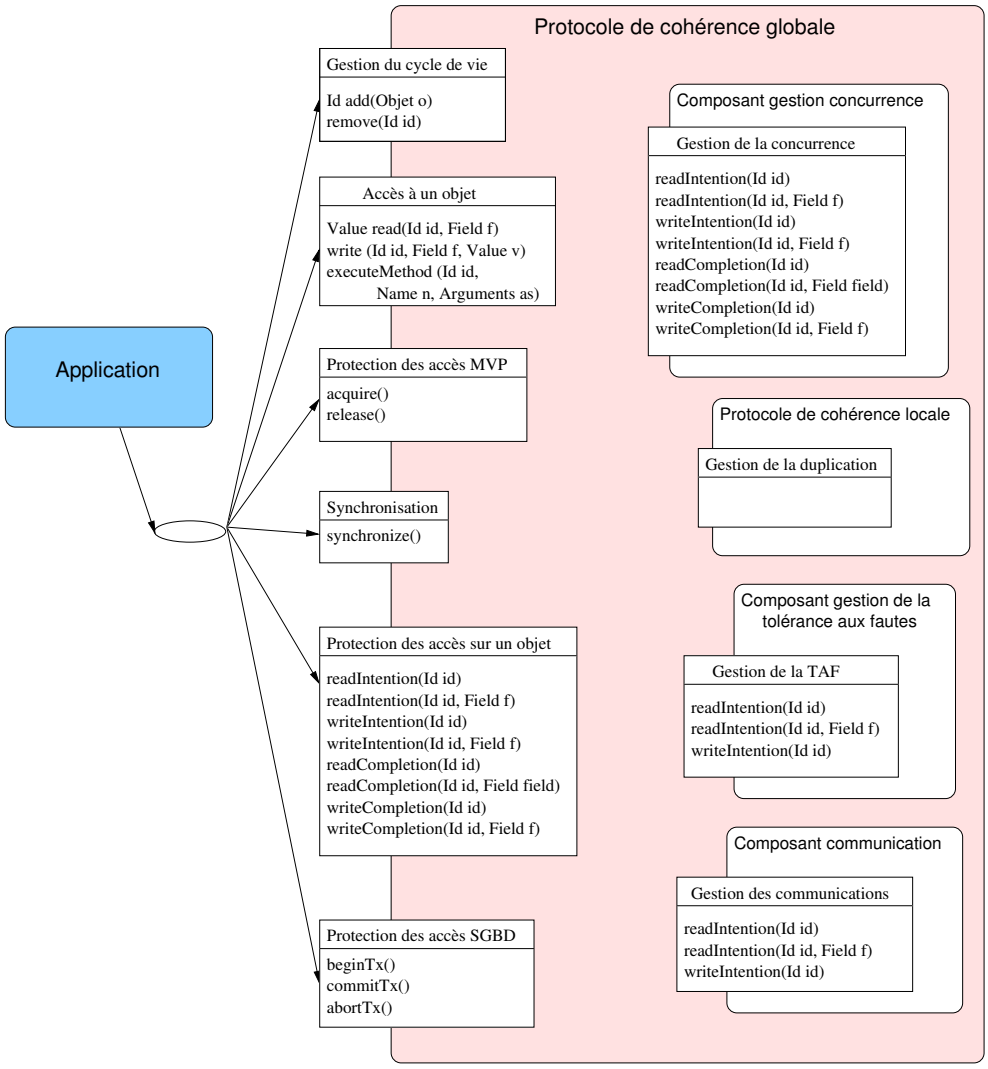


FIG. 6.2 – Décomposition du protocole de cohérence globale

6.2.1 Composant protocole de cohérence globale

Un protocole de cohérence globale gére l'ordre des accès sur la mémoire ou sur les données. Pour celà, il a besoin, comme un service de duplication, de connaitre les objets qu'il doit gérer ainsi que les accès faits sur ces objets. Il présente les deux interfaces suivantes à l'application :

Interface de gestion du cycle de vie :

– `Id add(Objet o)` informe le protocole de cohérence globale qu'il doit gérer la cohérence globale de l'objet `o`. Cette fonction renvoie un identificateur unique désignant l'objet géré.

- `remove(Id id)` informe le protocole de cohérence globale qu'il ne doit plus gérer la cohérence globale de l'objet désigné par l'identificateur `id`.

∎

Interface d'accès à un objet :

- `Value read(Id id, Field f)` renvoie la valeur `v` du champ `f` de l'objet désigné par l'identificateur `id`.
- `write(Id id, Field f, Value v)` écrit la valeur `v` dans le champ `f` de l'objet désigné par l'identificateur `id`.
- `executeMethod(Id id, Name n, Arguments as)` exécute la méthode de nom `n` avec les arguments `as` sur l'objet désigné par l'identificateur `id`.

∎

L'application peut elle-même diriger la gestion de la concurrence sur un objet. Ainsi, le composant protocole de cohérence globale présente l'interface suivante à l'application :

Interface de protection des accès sur un objet :

- `intention(Id id)` informe le protocole de cohérence globale d'un accès sur l'objet désigné par `id`.
- `readIntention(Id id)` informe le protocole de cohérence globale d'un accès en lecture sur l'objet désigné par `id`.
- `writeIntention(Id id)` informe le protocole de cohérence globale d'un accès en écriture sur l'objet désigné par `id`.
- `completion(Id id)` informe le protocole de cohérence globale de la terminaison d'un accès sur l'objet désigné par `id`.
- `readCompletion(Id id)` informe le protocole de cohérence globale de la terminaison d'un accès en lecture sur l'objet désigné par `id`.
- `writeCompletion(Id id)` informe le protocole de cohérence globale de la terminaison d'un accès en écriture sur l'objet désigné par `id`.

∎

Pour plus de clarté, nous avons omis dans l'interface le pendant de chaque opération permettant de préciser le champ de l'objet accédé (par exemple `readIntention(Id id, Field f)` informe le protocole de cohérence globale d'un accès en lecture sur le champ `f` de l'objet désigné par `id`). Cette remarque reste valide pour la suite du chapitre.

Bien souvent, l'application peut elle-même prendre les décisions concernant la gestion de la concurrence sur des groupes d'objets en informant des débuts et fins de sections critiques ou grâce à la notion de transaction dans les modèles utilisés dans les SGBDR (voir chapitre 2 section 2.3). On a également l'une des deux interfaces suivantes pour le composant protocole de cohérence globale suivant le contexte :

Interface de protection des accès sur des groupes d'objets :

- `intention({Id id})` informe le protocole de cohérence globale d'un accès sur un groupe d'ob-

jets, chacun des objets étant désigné par `id`.
- `readIntention({Id id})` informe le protocole de cohérence globale d'un accès sur un groupe d'objets, chacun des objets étant désigné par `id`. Les accès ne comporteront que des lectures.
- `writeIntention({Id id})` informe le protocole de cohérence globale d'un accès sur un groupe d'objets, chacun des objets étant désigné par `id`. Les accès ne comporteront que des écritures.
- `completion({Id id})` informe le protocole de cohérence globale de la terminaison d'un accès sur un groupe d'objet, chacun des objets étant désigné par `id`.
- `readCompletion({Id id})` informe le protocole de cohérence globale de la terminaison d'un accès sur un groupe d'objet, chacun des objets étant désigné par `id`. Les accès n'ont été que des opérations de lecture.
- `writeCompletion({Id id})` informe le protocole de cohérence globale de la terminaison d'un accès sur un groupe d'objet, chacun des objets étant désigné par `id`. Les accès n'ont été que des opérations d'écriture.

▮

Interface de protection des accès SGBDR :

- `Transaction beginTx({Id id})` informe du commencement d'une transaction portant sur un ensemble d'objets désigné par `id`.
- `Transaction beginReadTx({Id id})` informe du commencement d'une transaction portant sur un ensemble d'objets désigné par `id`. Cette transaction ne fera que des lectures.
- `commitTx(Transaction T)` demande la validation de la transaction T.
- `commitReadTx(Transaction T)` demande la validation de la transaction T. Cette transaction n'a comporté que des lectures.
- `abortTx(Transaction T)` demande l'annulation de la transaction T.
- `abortReadTx(Transaction T)` demande l'annulation de la transaction T. Cette transaction n'a comporté que des lectures.

▮

Il peut également être demandé explicitement de synchroniser les objets du système. Il ne s'agit pas ici de synchroniser les copies d'un même objet dupliqué (rôle du protocole de cohérence locale), mais les accès sur les objets gérés par le protocole de cohérence globale. C'est une notion présente dans les modèles de cohérence avec synchronisation des MPR (voir chapitre 2 section 2.4)

Interface de synchronisation des MPR :

- `acquire()` informe du début d'une zone de synchronisation.
- `release()` informe de la fin d'une zone de synchronisation.
- `acquire(Id id)` informe du début d'une zone de synchronisation sur la variable de synchronisation désignée par `id`.
- `release(Id id)` informe de la fin d'une zone de synchronisation sur la variable de synchronisation désignée par `id`.
- `synchronize()` informe le protocole de cohérence globale d'une demande de type "rendez-vous".

▮

6.2.2 Composant gestion de la concurrence

Un des composants du protocole de cohérence globale est le composant de gestion de la concurrence. Il a en charge le contrôle de la concurrence sur les objets du système.

Ce composant est le même que celui défini dans le chapitre 5 section 5.3.1. Cependant, deux instances différentes entrent en jeu selon que l'on se trouve au niveau global ou local. Au niveau global il s'agit du problème classique d'isolation, dirigé par le protocole de cohérence globale. Au niveau local il s'agit de la gestion de la concurrence entre les copies, dirigée par le contrôleur de concurrence du protocole de cohérence locale. Ainsi on peut imaginer qu'une seule copie soit accédée en écriture ou lecture à un moment donné (gestion de la cohérence au niveau local), mais par plusieurs objets (gestion de la concurrence au niveau global) afin de relâcher l'isolation. Le cas inverse est également envisageable. Dans ce cas, on simule, au niveau global, l'accès à un objet logique par un unique objet à un moment donné ; au niveau local cela peut se traduire par un accès simultané à différentes copies. Cette décomposition permet au composant gestion de la concurrence d'être réutilisé et instancié suivant les besoins à chacun des niveaux. L'interface de ce composant utilisée par le protocole de cohérence locale ou globale est la suivante :

Interface du gestionnaire de concurrence :

- `readIntention(Id id)` informe le composant de gestion de la concurrence d'un accès en lecture sur l'objet désigné par `id`.
- `writeIntention(Id id)` informe le composant de gestion de la concurrence d'un accès en écriture sur l'objet désigné par `id`.
- `readCompletion(Id id)` informe le composant de gestion de la concurrence de la terminaison d'un accès en lecture sur l'objet désigné par `id`.
- `writeCompletion(Id id)` informe le composant de gestion de la concurrence de la terminaison d'un accès en écriture sur l'objet désigné par `id`.

6.2.3 Composant gestion de la tolérance aux fautes

Précedemment, nous avons vu que les fautes ont des répercussions sur le protocole de cohérence locale (voir chapitre 5 section 5.3.2). Le protocole de cohérence globale doit également les prendre en compte. Afin de s'assurer de ne pas perdre de donnée en cas de défaillance, le protocole de cohérence globale peut s'appuyer sur un support persistant, dupliquer les données sensibles, etc. L'interface du composant assurant la tolérance aux fautes est la suivante :

Interface de suivi des fautes :

- `control(Replica r)` demande la surveillance d'un objet.
- `boolean isALive(Replica r)` renvoie vrai si la copie est correcte.

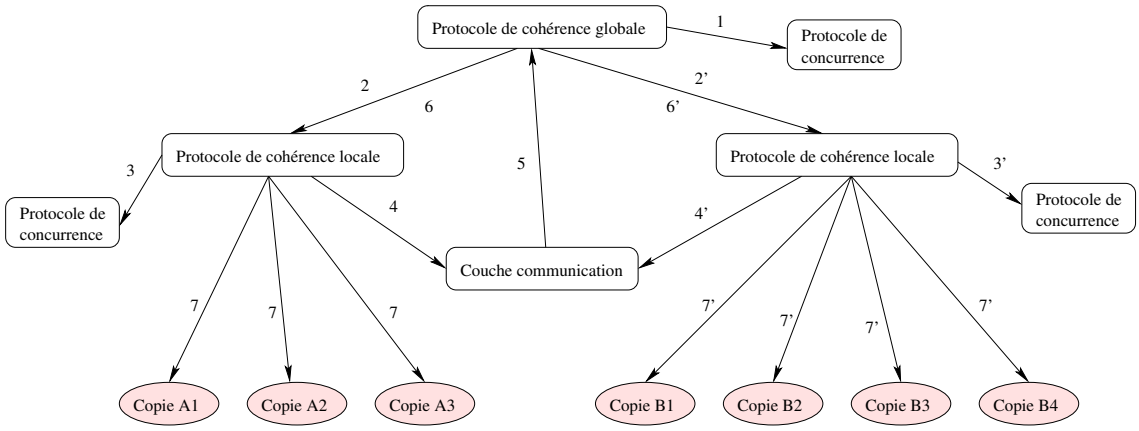


FIG. 6.3 – Intéractions entre les composants du protocole de cohérence globale

6.2.4 Composant protocole de communication

Dans un système réparti, l'ordre des messages échangés a des répercussions sur le protocole de cohérence locale et sur le protocole de cohérence globale même en l'absence de duplication. Ainsi, il existe différents protocoles de communication suivant l'ordre dans lequel sont délivrés les messages (ordre total, FIFO, causal) et la fiabilité (fiable ou non) (chapitre 2, section 2.1.9). Le composant protocole de communication comporte une interface afin d'envoyer des messages, ainsi qu'une interface pour le protocole de cohérence globale lui permettant de les recevoir.

Interface descendante du protocole de communication :

– `send(Message m, Destinataires d)` envoie un message à l'ensemble des destinaires d.

Interface ascendante du protocole de communication :

– `receive(Message m)` permet de recevoir un message.

6.2.5 Composant protocole de cohérence locale

Ce composant met en œuvre le protocole de cohérence locale (chapitre 5). Il s'appuie sur la couche de communication pour propager les mises à jour entre les copies. Ces mises à jour doivent repasser par le composant protocole de cohérence globale (figure 6.3) afin de garantir l'ordre requis sur les accès aux objets. Lorsque le composant de cohérence globale traite une opération, il informe son gestionnaire de concurrence (point 1). Ensuite, cette opération est transmise aux protocoles de cohérence locale gérant les objets de l'opération à traiter (points

2 et 2'). Chaque protocole de cohérence locale informe son gestionnaire de concurrence afin de garantir l'ordre sur l'objet dupliqué (point 3 et 3'). Ensuite, en fonction de la cohérence à assurer entre les copies, le protocole de cohérence locale envoie les mises à jour aux différentes copies par l'intermédiaire de la couche communication (points 4 et 4'). Celle-ci délivre le message de synchronisation au protocole de cohérence globale (point 5). Celui-ci, une fois l'ordre d'accès valide par rapport aux autres objets, renvoie chaque opération au protocole de cohérence locale approprié (point 6 et 6'). Enfin, le protocole de cohérence locale installe les mises à jour (points 7 et 7').

6.3 Séparation des considérations locales et globales

Dans le chapitre 2, nous avons fait apparaître trois types de modèle de cohérence globale :

- les modèles sans synchronisation des MPR (section 2.4.2) ordonnançant les accès sur des objets.
- les modèles avec synchronisation des MPR (section 2.4.3) faisant intervenir des points de synchronisation.
- les modèles transactionnels des SGBDR (section 2.3) ordonnançant des transactions qui sont elles-mêmes des groupes d'opérations ordonnées.

L'objectif de cette section est de montrer qu'avec les quatres types de modèle de cohérence locale présentés dans le chapitre précédent (chapitre 5), il est possible de construire les trois types de modèles de cohérence globale présentés ci-dessus.

Cette section est organisée de la manière suivante : nous commençons par traiter les modèles de cohérence globale sans synchronisation des mémoires partagées réparties (section 6.3.1), puis les modèles de cohérence avec synchronisation (section 6.3.2) et les modèles de cohérence globale utilisés dans les bases de données réparties (section 6.3.3). Pour chaque type de modèle de cohérence globale, nous présentons les principes d'intéraction entre cohérence globale et cohérence locale. Ensuite, nous précisons pour divers modèles de cohérence globale, le modèle de cohérence locale nécessaire. Nous montrons également les interfaces entre cohérence locale et globale qu'il convient d'ajouter afin de mettre en œuvre les modèles de cohérence globale.

6.3.1 Modèles de cohérence globale sans synchronisation pour MPR

Nous présentons ici comment construire des modèles de cohérence globale sans synchronisation pour mémoires partagées réparties à partir des services de RS2.7.

6.3.1.1 Principes

Les modèles de cohérence sans synchronisation (ou forts) ordonnancent les opérations de lecture et d'écriture en ne s'appuyant sur aucune information provenant de l'application

(voir chapitre 2 section 2.4.2). Dans ce contexte, un protocole de cohérence globale se charge d'ordonnancer les opérations portant sur les objets qu'il gère, afin d'obtenir une exécution dont l'histoire $\widehat{H} = (H, \rightarrow_H)$ respecte le modèle de cohérence globale désiré.

Lorsqu'il y a duplication, le protocole de cohérence locale se charge d'ordonnancer les opérations portant sur les copies d'un même objet logique : $\forall j$, l'histoire $\widehat{H}_{O_j} = (H_{O_j}, \rightarrow H_{O_j})$ doit respecter un certain modèle de cohérence locale. Ainsi, dans le cas où il y a duplication, $\widehat{H}$ est construit à partir d'un ensemble de $\widehat{H}_{O_j}$ et d'un ordonnancement des opérations portant sur des objets différents. Le modèle de cohérence locale doit donc au minimum respecter le modèle de cohérence globale ou alors être plus contraignant. En effet, si $\widehat{H}_{O_j}$ ne définit pas certaines relations entre les opérations portant sur un objet logique O_j (relations devant être définies dans $\widehat{H}$) alors le modèle de cohérence globale ne pourra être garanti.

Afin de montrer les intéractions entre modèle de cohérence locale et globale, nous prenons comme exemple les modèles de cohérence globale séquentielle, causale et PRAM.

Modèle de cohérence globale séquentielle. Dans la section 2.4.2.2, nous avons présenté le modèle de cohérence séquentielle. Plus formellement, on peut le définir de la façon suivante :

> **Définition 6.3** : *Modèle de cohérence globale séquentielle*
>
> Un modèle de cohérence globale séquentielle garantit que $\widehat{H}$ admet une extension linéaire $\widehat{S} = (H, \rightarrow_H)$ légale.

Toutes les relations d'ordre sur les exécutions locales (sur les P_i) doivent être identiques. Les protocoles de cohérence globale mettant en œuvre ce modèle doivent garantir un ordre total. Le modèle de cohérence locale doit également garantir un ordre total sur chaque objet logique : il doit être à copie unique séquentiel. Il est possible également d'utiliser un modèle de cohérence locale à copie unique atomique qui est plus contraignant que le séquentiel mais qui garantit toutes les relations de celui-ci.

De nombreux travaux confondent cohérence atomique et cohérence séquentielle. La confusion vient du fait que construire un mécanisme optimal s'assurant de la "séquentialisabilité" d'une exécution est NP complet [MRZ94]. Ainsi, de nombreuses approximations de complexité polynômiale sont proposées. La plus simple est l'ordonnancement total de toutes les opérations correspondant à un modèle de cohérence atomique, qui utilise donc un modèle de cohérence locale à copie unique atomique.

Modèle de cohérence globale causale. Le modèle de cohérence causale se définit de la manière suivante (modèle présenté en section 2.4.2.3) :

Définition 6.4 : *Modèle de cohérence globale causale*

Soit $\widehat{h}'_i = (h'_i, \rightarrow_H)$ tel que $h'_i =$
– l'ensemble des écritures de H perçues par P_i et
– toutes les lectures effectuées par le processus P_i.

Un modèle de cohérence globale causale garantit que $\forall$ P_i, $\widehat{h}'_i$ est légale.

Ce modèle ne garde que les relations de causalité entre les opérations, le modèle de cohérence locale doit en faire de même ; on peut utiliser un modèle de cohérence locale à copies divergentes causale. Ce dernier peut également être plus contraignant comme un modèle à copie unique séquentiel ou atomique sans remettre en cause le modèle de cohérence globale.

Modèle de cohérence globale PRAM. Dans la section 2.4.2.4, nous avons défini le modèle de cohérence séquentielle. Plus formellement cela donne :

Définition 6.5 : *Modèle de cohérence globale PRAM*

Soit $\widehat{H}$ défini comme dans la section 5.1.3 et en remplaçant (iii) par $\exists op_3$: $(op_1 \rightarrow_{h_i} op_3)$ et $(op_3 \rightarrow_{h_i} op_2)$.
Soit $\widehat{h}'_i = (h'_i, \rightarrow_H)$ tel que $h'_i =$
– l'ensemble des écritures de H perçues par P_i et
– toutes les lectures effectuées par le processus P_i.

Un modèle de cohérence globale PRAM garantit que $\forall P_i$, $\widehat{h}'_i$ est légale.

Ce modèle relâche encore certaines contraintes en ne gardant que la transitivité sur un même processus, le modèle de cohérence locale doit en faire de même ; on peut ainsi utiliser un modèle de cohérence locale à copies divergentes FIFO. On peut également utiliser un modèle plus contraignant comme à copies divergentes causale ou à copie unique séquentiel ou atomique.

6.3.1.2 Mise en œuvre

Les seules informations venant de l'application dont le protocole de cohérence globale a besoin sont les accès faits sur les objets. Pour cela, l'application utilise l'interface d'**accès à un objet** (section 6.2.1) du protocole de cohérence globale.

De même, les seules informations venant du protocole de cohérence globale dont le protocole de cohérence locale a besoin sont les accès faits sur un objet logique. Pour cela, le protocole de cohérence globale utilise l'interface d'**accès à un objet dupliqué** (chapitre 4, section 4.2) du protocole de cohérence locale.

Pour mettre en œuvre ces modèles, il est nécessaire de séparer l'information d'ordonnancement des opérations sur les objets au niveau protocole de cohérence globale et l'information d'ordonnancement des opérations portant sur les copies au niveau du protocole de cohérence locale. Par exemple, pour le modèle de cohérence causale, le protocole de cohérence globale a en charge les liens de causalité entre les objets du système et le protocole de cohérence locale les liens de causalité entre les copies d'un même objet. Afin d'entremêler les deux ordres il est nécessaire que les synchronisations entre les copies repassent par le protocole de cohérence globale comme présenté en section 6.2.5.

6.3.2 Modèles de cohérence globale avec synchonisation pour MPR

Nous présentons ici comment construire des modèles de cohérence globale avec synchronisation pour mémoires partagées réparties.

6.3.2.1 Principes

Les modèles de cohérence avec synchronisation ordonnancent les opérations d'écriture et de lecture sur des groupes d'objets en s'appuyant sur des variables de synchronisation (voir chapitre 2 section 2.4.3).

Pour ces modèles de cohérence on peut distinguer trois types d'ordonnancement : (1) $\widehat{H}$ respectant un certain modèle comme précédemment, (2) l'ordre sur les variables de synchronisation dépendant également d'un certain modèle et (3) une des principales spécificités de ces modèles de cohérence globale vient du fait que le point 1 est dirigé par le point 2. Ainsi, nous devons définir l'histoire restreinte aux variables de synchronisation ($\widehat{H}_s = (H_s, \rightarrow_{H_s})$) et l'histoire des opérations de synchronisation et des événements d'accès ($\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow_{\mathcal{H}})$) pour définir ces modèles.

Lorsqu'il y a duplication des données le modèle de cohérence locale est seulement un sous ensemble du point 1 comme dans la section précédente (section 6.3.1). De même, si les variables de synchronisation sont dupliquées, on associe un modèle de cohérence locale au modèle de cohérence globale dirigeant la cohérence de ces variables.

Histoire globale restreinte aux variables de synchronisation $\widehat{H}_s = (H_s, \rightarrow_{H_s})$. L'ensemble des opérations sur les variables de synchronisation sur un processus P_i est noté h_{s_i}. Si les opérations sur les variables de synchronisation op_1 et op_2 sont effectuées sur le processus P_i et op_1 a eu lieu en premier, alors op_1 précède op_2 sur le processus P_i ($op_1 \rightarrow_{h_{s_i}} op_2$). Nous appelons l'ensemble des séquences ayant eu lieu sur un processus P_i l'histoire locale de synchronisation du processus P_i et nous la notons $\widehat{h}_{s_i} = (h_{s_i}, \rightarrow_{h_{s_i}})$.

Une exécution sur les variables de synchronisation est représentée par un ordre partiel sur l'ensemble des opérations H_s exécutées par tous les processus. Nous appelons cet ordre partiel l'histoire globale de synchronisation $\widehat{H}_s$, définie comme dans la section 5.1.3.

Notation	*Signification*
$s_i(O_j)$	Opération de synchronisation sur l'objet O_j par le processus P_i.
h_{s_i}	Ensemble des opérations de synchronisation du processus P_i.
H_s	Ensemble des opérations de synchronisation.
$\rightarrow_{h_{s_i}}$	Relation d'ordre entre les opérations de synchronisation exécutées par le processus P_i.
$\rightarrow_{H_s}$	Relation d'ordre entre les opérations de synchronisation.
$\widehat{h}_{s_i} = (h_{s_i}, \rightarrow_{h_{s_i}})$	Histoire locale du processus P_i sur les opérations de synchronisation.
$\widehat{H}_s = (H_s, \rightarrow_{H_s})$	Histoire globale sur les opérations de synchronisation.

FIG. 6.4 – Notations utilisées pour modéliser l'histoire sur les variables de synchronisation

Notation	*Signification*
$\mathcal{H}_i$	Ensemble des opérations survenues sur le processus P_i.
$\mathcal{H}$	Ensemble des opérations survenues dans le système réparti.
$\rightarrow_{\mathcal{H}_i}$	Relation d'ordre entre les opérations exécutées par le processus P_i.
$\rightarrow_{\mathcal{H}}$	Relation d'ordre entre les opérations de exécutées dans le système.
$\widehat{\mathcal{H}_i} = (\mathcal{H}_i, \rightarrow_{\mathcal{H}_i})$	Histoire locale du processus P_i.
$\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow_{\mathcal{H}})$	Histoire globale du système réparti.

FIG. 6.5 – Notations utilisées pour modéliser les MPR avec synchronisations

Définition 6.6 : *Histoire globale* $\widehat{H}_s = (H_s, \rightarrow_{H_s})$

$\widehat{H}_s = (H_s, \rightarrow_{H_s})$ est l'histoire globale des variables de synchronisation tel que :
- H_s = l'ensemble des opérations sur les variables de synchronisation et
- $\rightarrow_{H_s}$ est définie comme dans la section 5.1.3.

Histoire des opérations de synchronisation et des événements d'accès ($\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow_{\mathcal{H}}$)). L'histoire $\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow_{\mathcal{H}})$ est l'histoire du système réparti prenant en compte les opérations d'accès ainsi que les opérations sur les variables de synchronisation.

Définition 6.7 : *Histoire globale* $\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow_{\mathcal{H}})$

$\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow_{\mathcal{H}})$ est une histoire globale, tel que
- $\mathcal{H} = H \bigcup H_s$ et
- $\rightarrow_{\mathcal{H}}$ est définie comme dans la section 5.1.3.

Notation	*Signification*
$s_i(O_{j,k})$	Opération de synchronisation sur la copie $O_{j,k}$ par P_i.
$h_{s_{O_{j,k}}}$	Ensemble des opérations de synchronisation sur la copie $O_{j,k}$.
$H_{s_{O_j}}$	Ensemble des opérations de synchronisation sur l'objet O_j.
$\rightarrow_{h_{s_{O_{j,k}}}}$	Relation d'ordre entre les opérations de synchronisation exécutées sur la copie $O_{j,k}$.
$\rightarrow_{H_{s_{O_j}}}$	Relation d'ordre entre les opérations de synchronisation exécutées sur l'objet logique O_j.
$\widehat{h}_{s_{O_{j,k}}} = (h_{s_{O_{j,k}}}, \rightarrow_{h_{s_{O_{j,k}}}})$	Histoire locale des opérations de synchronisation sur la copie $O_{j,k}$.
$\widehat{H}_{s_{O_j}} = (H_{s_{O_j}}, \rightarrow_{H_{s_{O_j}}})$	Histoire globale des opérations de synchronisation. sur l'objet logique O_j.

FIG. 6.6 – Notations utilisées pour les variables de synchronisation d'un objet dupliqué

Dans la suite, nous prenons comme exemple les modèles de cohérence faible et au relâchement. Le modèle de cohérence à l'entrée (chapitre 2, section 2.4.3.3) n'entre pas dans cette catégorie d'intéraction avec le protocole de cohérence locale. Il permet de mettre en œuvre des sections critiques et se rapproche plus des modèles rencontrés dans les SGBDR (section 6.3.3)

Modèle de cohérence globale faible. Dans la section 2.4.3.1, nous avons défini le modèle de cohérence faible. Plus formellement, nous définissons ce modèle de la façon suivante :

Définition 6.8 : *Modèle de cohérence faible*

Un modèle de cohérence globale faible garantit que :
- (i) $\widehat{H} = (H, \rightarrow_H)$ respecte le modèle de cohérence globale PRAM,
- (ii) $\widehat{H}_s = (H_s, \rightarrow H_s)$ respecte le modèle de cohérence globale séquentielle,
- (iii) $\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow \mathcal{H})$ est tel que
  - l'accès aux variables de synchronisation ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites et
  - les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations sur les variables de synchronisation sont terminées sur tous les sites.

S'il y a duplication des données, le modèle de cohérence locale sur un objet dupliqué doit être à copies divergentes FIFO ou plus contraignant comme le modèle à copie unique séquentiel ou atomique (point (i)).

D'autre part, pour ce modèle les opérations de synchronisation ne portent que sur une variable de synchronisation. Si cette variable de synchronisation est dupliquée alors le modèle de cohérence locale doit être à copie unique séquentiel ou plus contraignant comme le modèle

de cohérence locale à copie unique atomique (point (ii)).

La construction de l'histoire globale $\widehat{\mathcal{H}}$ reste à la charge du protocole de cohérence globale. Il doit garantir le point (iii) de la définition 6.8.

Modèle de cohérence globale au relâchement. Dans la section 2.4.3.2, nous avons défini le modèle de cohérence au relâchement. Plus formellement, on définit ce modèle de la façon suivante :

Définition 6.9 : *Modèle de cohérence au relâchement*
Un modèle de cohérence globale au relâchement garantit que :
- (i) $\widehat{H} = (H, \rightarrow_H)$ respecte le modèle de cohérence globale PRAM,
- (ii) $\widehat{H_s} = (H_s, \rightarrow H_s)$ respecte le modèle de cohérence globale PRAM,
- (iii) $\widehat{\mathcal{H}} = (\mathcal{H}, \rightarrow \mathcal{H})$ est tel que
  - Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations d'acquisition (acquire) précédentes sont terminées sur tous les sites ;
  - L'opération de relâchement (release) ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites.

Ainsi, s'il y a duplication des données, le modèle de cohérence locale doit être à copies divergentes FIFO ou plus contraignant comme le modèle à copie unique séquentiel ou atomique (point (i)).

Les opérations de synchronisation ne portent egalement ici que sur un objet. Il n'existe qu'une variable de synchronisation dans ce modèle. Deux opérations sont possibles sur celle-ci : acquérir et relâcher. Il n'est pas nécessaire de préciser l'ordre de ces opérations dans le modèle, l'opération acquérir n'est pas forcement suivie de l'opération relâcher (cela reste la responsabilité de l'utilisateur). Si la variable de synchronisation est dupliquée alors le modèle de cohérence locale doit être également à copies divergentes FIFO ou plus contraignant comme le modèle à copie unique séquentiel ou atomique (point (ii)).

La construction de l'histoire globale $\widehat{\mathcal{H}}$ reste à la charge du protocole de cohérence globale. Il doit garantir le point (iii) de la définition.

6.3.2.2 Mise en œuvre

Les intéractions entre l'application et le protocole de cohérence globale ne sont plus implicites. Le développeur de l'application intéragit avec le protocole par l'intermédiaire de l'interface de **synchronisation des MPR**.

Afin de mettre en œuvre le modèle de cohérence globale sur les données, les intéractions entre cohérence locale et globale sont les mêmes que dans la section précédente (section 6.3.1). Cependant, afin d'assurer le point (iii) des définitions présentées, le protocole de cohérence

globale doit diriger la mise à jour des différentes copies d'un objet logique et être informé quand celle-ci est terminée. La synchronisation du niveau globale ne peut se faire que si la mise à jour du niveau locale est terminée. Ainsi, le protocole de cohérence locale offre au protocole de cohérence globale l'interface suivante :

Interface de mise à jour des copies :

- `synchronize()` informe le protocole de cohérence locale d'une demande de synchronisation de la part du protocole de cohérence globale.

∎

Une fois le processus de synchronisation d'un objet logique terminé, le protocole de cohérence locale informe le protocole de cohérence globale.

Interface de synchronisation LG :

- `synchronizationDone(Id id)` informe le protocole de cohérence globale de la terminaison de la synchronisation de l'objet logique désignée par `id`.

∎

6.3.3 Modèles de cohérence globale pour SGBDR

Nous présentons ici comment construire des modèles de cohérence globale pour SGBDR, dans le cas où il y a duplication à partir des services obtenus à partir de RS2.7.

6.3.3.1 Principes

Les modèles de cohérence utilisés dans les SGBDR portent sur des groupes d'opérations appellés transactions (voir chapitre 2 section 2.3). Les transactions peuvent être vues comme des sections critiques et non plus comme des points de synchronisation comme dans la section précédente (section 6.3.2). De plus, dans le contexte des SGBD, les transactions doivent garantir les propriétés ACID : atomicité, cohérence, isolation et durabilité (chapitre 2, section 2.3). Ainsi, nous reprenons, la modélisation d'une exécution répartie présentée en section 5.1.3 afin de définir une exécution au niveau des transactions et non plus des opérations.

Modélisation de l'exécution. Chaque processus P_i est vu comme un gestionnaire de transactions qui exécute les transactions de façon séquentielle $T_i^1, T_i^2, ..., T_i^n$ où T_i^n est la n^{ime} transaction exécutée par P_i.

Soit h_i l'ensemble des transactions effectué par P_i. Si un processus P_i effectue la transaction T_i^1 puis T_i^2, alors T_i^1 précède T_i^2 dans l'ordre du programme de P_i ($T_i^1 \rightarrow_{h_i} T_i^2$). L'ensemble des séquences du processus P_i est appelé histoire locale $\widehat{h_i} = (h_i, \rightarrow_{h_i})$ du processus P_i.

Définition 6.10 : *Histoire globale des transactions* $\widehat{H} = (H, \rightarrow_H)$

L'histoire globale des transactions est un ordre partiel $\widehat{H} = (H, \rightarrow_H)$ tel que :
- $H = \bigcup_i h_i$.
- $T^1 \rightarrow_H T^2$ si :
  (i) $\exists P_i : T^1 \rightarrow_{h_i} T^2$ ou
  (ii) $T^1_i = T^1$, $T^2_{i'} = T^2$ tel que $w_1(O_1)x \in T^1_i$ et $r_2(O_1)x = T^2_{i'}$ ou
  (iii) $\exists T^3 : (T^1 \rightarrow_H T^3)$ et $(T^3 \rightarrow_H T^2)$.

Définition 6.11 : *Légalité d'une transaction*

Une transaction T^a_i est légale si $\forall r_a(O_1)v$, $\exists T^{a'}_{i'}, w_{a'}(O_1)v$ telle que :
- $T^{a'}_{i'} \rightarrow_H T^a_i$,
- $\nexists w_{a''}(O_1)v'$ telle que $w_{a'}(O_1)v \rightarrow_H w_{a''}(O_1)v' \rightarrow_H r_a(O_1)v$ (il n'existe pas d'opération d'écriture intermédiaire)

Une transaction est légale si elle lit la dernière valeur écrite. Une histoire $\widehat{H}$ est légale si toutes ses transactions sont légales.

Le contexte des SGBDR apporte une spécificité : le protocole de cohérence locale s'occupe de l'ordonnancement des opérations sur les copies d'un objet logique et le protocole de cohérence globale de l'ordonnancement de groupe d'opérations portant sur des objets différents. Afin de présenter les interactions entre cohérence globale et locale, nous prenons comme exemple les modèles de cohérence globale suivants : la sérialisabilité sur une copie, l'epsilon sérialisabilité et les modèles où les copies peuvent diverger temporairement mais tout en assurant la sérialisabilité.

Sérialisabilité. Nous définissons la sériabilisabilité (chapitre 2, section 2.3.1.3) de manière formelle de la façon suivante :

Définition 6.12 : *Sérialisabilité*

Une histoire $\widehat{H} = (H, \rightarrow_H)$ est sérialisable s'il existe une extension linéaire $\widehat{S} = (H, \rightarrow_S)$ légale.

Les protocoles de cohérence globale mettant en œuvre ce modèle doivent garantir un ordre total sur les transactions. Le modèle de cohérence locale doit également garantir un ordre total sur chaque copie (à l'intérieur d'une transaction) : il est nécessaire d'utiliser pour mettre en œuvre ce modèle un protocole de cohérence locale à copie unique atomique ou séquentiel.

ϵ-sérialisabilité : L'ϵ-sérialisabilité [PL91, RP95] est un critère de correction affaiblissant la sérialisabilité en relâchant l'isolation. Ce modèle permet à des transactions en lecture (*"query*

transaction") de voir les traitements de transactions non validées. L'ϵ-sérialisabilité se définit sur des critères applicatifs : la consultation du compte en banque peut être imprécise de 80% du nombre des opérations effectuées, de 100 Euros, etc. Concrètement, chaque mise à jour sur une donnée modifie la valeur d'une grandeur (*export-consistency*). Chaque transaction en lecture spécifie une grandeur (*import-limit*) définissant l'imprécision qu'elle tolère sur la valeur de ses lectures. La convergence de la base est assurée en interdisant aux transactions contenant des mises à jour de lire des valeurs non validées (*import-limit=0* pour ces transactions). Cependant, dans le modèle, il est également possible de spécifier des imprécisions de lecture pour ces transactions mais l'effet sur la base ne peut plus être controlé.

Toutes les relations d'ordre sur les exécutions locales sont identiques. Ainsi, les protocoles de cohérence globale mettant en œuvre ce modèle doivent garantir un ordre total. Le modèle de cohérence locale doit également garantir un ordre total sur chaque copie. Il est nécessaire d'utiliser pour mettre en œeuvre ce modèle un protocole de cohérence locale à copie unique. Cependant, les transactions en lecture n'ont pas besoin d'être ordonnées de manière stricte ; on peut utiliser le modèle à copies convergentes avec lectures sur les copies divergentes.

Modèle de cohérence globale assurant la sérialisabilité et relâchant la cohérence entre les copies. Les différentes copies d'un même objet logique sont mises à jour dans des transactions différentes rendant plus difficile l'obtention de la sérialisabilité.

Etant donné que les copies ne sont plus mises à jour dans les mêmes transactions mais que celles-ci doivent converger vers le même état, il est nécessaire de s'appuyer sur un modèle à copies convergentes pour gérer les copies. Si le modèle de cohérence locale est à copies convergentes avec les copies divergentes en lecture alors il est possible d'obtenir la sérialisabilité en positionnant les copies de référence et les copies en lecture sur les sites adéquates en fonction de la nature des transactions (chapitre 2, section 2.3.2.2).

Si le modèle de cohérence locale est à copies convergentes avec les copies divergentes en écriture le problème est plus complexe. La sérialisabilité est obtenue à certain moment par réécriture des histoires locales et de l'histoire globale. Le protocole de cohérence locale réécrit l'histoire de chaque objet logique et le protocole de cohérence globale décide si un ordonnancement entre les transactions est correct en s'appuyant sur la fonction de réconciliation du protocole de cohérence locale. En effet, cette fonction de réconciliation caractérisant différents modèles de cohérence locale à copies convergentes avec écriture sur les copies divergentes (chapitre 5, section 5.2.4) permet de montrer au protocole de cohérence globale l'ordre retenu au niveau local sur les différentes copies. Le protocole de cohérence globale peut donc à partir des différents ordres retenus sur les différents objets logiques voir s'il est capable de construire un ordre entre les transactions. Si ce n'est pas le cas, il annule certaines transactions.

Modèle à base de sections critiques. Les modèles où toutes données accédées dans une section critique ne peuvent être perçues par d'autres processus ne possédant pas cette section critique suivent le même principe que les modèles que l'on rencontre dans les SGBD. La seule différence est que l'on ne retrouve pas les propriétés ACID. Le protocole de cohérence globale se charge d'assurer l'isolation et le protocole de cohérence locale est à copie unique.

Le modèle de cohérence à l'entrée des MVP entre dans cette catégorie d'intéractions entre cohérence locale et cohérence globale.

6.3.3.2 Mise en œuvre

Le protocole de cohérence globale doit informer le protocole de cohérence locale sur les débuts et fins d'accès concurrents. Le protocole de cohérence locale, suivant les cas, peut alors synchroniser les copies en début ou fin de section critique, regrouper plusieurs écritures dans un seul message de synchronisation, gérer la concurrence sur l'objet logique, etc. On a l'interface suivante pour le protocole de cohérence locale utilisée par le protocole de cohérence globale :

Interface de protection des accès :

- `intention()` informe le protocole de cohérence locale d'un accès.
- `readIntention()` informe le protocole de cohérence locale d'un accès en lecture.
- `writeIntention()` informe le protocole de cohérence locale d'un accès en écriture.
- `completion()` informe le protocole de cohérence locale de la terminaison d'un accès.
- `readCompletion()` informe le protocole de cohérence locale de la terminaison d'un accès en lecture.
- `writeCompletion()` informe le protocole de cohérence locale de la terminaison d'un accès en écriture.

∎

Le protocole de cohérence locale doit également informer le protocole de cohérence globale des messages de synchronisation dans le cas où les mises à jour sont faites dans des transactions différentes.

Interface de mise à jour PG :

- `synchronisation(Opérations op, Group replicas)` informe le protocole de cohérence globale d'un ensemble d'opérations `op` de synchronisation portant sur le groupe de copies `replicas`.
- `synchronisation(State state, Group replicas)` informe le protocole de cohérence globale d'un état `state` à appliquer aux différentes copies du groupe `replicas`.

∎

Ces demandes de synchronisation de la part du protocole de cohérence locale entraînent la construction de transactions de mises à jour par le protocole de cohérence globale.

6.4 Conclusion

Dans ce chapitre, nous avons montré comment RS2.7 est adaptable au contexte non fonctionnel. Le contexte non fonctionnel est caractérisé par un modèle de cohérence globale

décrivant comment apparait la mémoire (ou les données) à l'application. Nous distinguons trois types de modèles de cohérence globale : les modèles ordonnançant les accès sur des objets, ceux faisant intervenir des points de synchronisation et ceux ordonnançant des groupes d'opérations. Un modèle est mis en œuvre par un protocole de cohérence globale prenant en charge la répartition, le contrôle de la concurence, la duplication, etc.

Nous avons montré qu'il est possible de décomposer le protocole de cohérence globale et d'isoler l'aspect duplication. A partir de cela, nous avons défini les interactions entre cohérence globale et cohérence locale. Pour chaque type de modèle de cohérence globale, nous avons commencé par définir comment s'insère le modèle de cohérence locale dans le modèle de cohérence globale. Ensuite nous avons défini pour quelques modèles de cohérence globale, le modèle de cohérence locale nécessaire. A partir de cette étude formelle, nous avons donné les interfaces nécessaires afin que protocole de cohérence locale et protocole de cohérence globale intéragissent.

Ainsi, nous avons vu que les protocoles de cohérence locale (mis en œuvre par un service de RS2.7) peuvent être réutilisés pour mettre en œuvre différents modèles de cohérence globale. Il est également possible de substituer un protocole de cohérence locale par un autre mettant en œuvre le même modèle de cohérence locale tout en gardant le même modèle de cohérence globale.

Dans le chapitre suivant (chapitre 7), nous présentons comment nous obtenons l'adaptabilité dans tout ou partie des protocoles de cohérence locale.

Le contexte des mots, c'est le monde.
Lec, Stanislaw Jerzy - Nouvelles pensées échevelées

Chapitre 7

Adaptabilité des protocoles de cohérence locale

Sommaire

ans les chapitres 4 et 5, nous avons vu que RS2.7 peut supporter différents protocoles de cohérence locale et dans le chapitre 6 qu'il peut s'adapter à différents contextes non fonctionnels. Dans ce chapitre nous montrons comment nous construisons un protocole de cohérence locale et comment nous obtenons l'adaptabilité dans tout ou partie de ces protocoles.

Pour obtenir cette adaptabilité, nous proposons deux factorisations des protocoles de cohérence locale. Le but de ces factorisation est de regrouper les points communs des protocoles afin de construire des composants réutilisables et d'introduire des points d'adaptabilité. Il est possible de remplacer un composant par un autre, d'en enlever, d'en rajouter ou encore de modifier les intéractions entre eux afin d'adapter les protocoles aux besoins des applications et du contexte non fonctionnel.

La première factorisation est une factorisation structurelle. Elle permet d'exhiber les phases communes aux protocoles de cohérence locale ainsi que l'ordonnancement de ces phases. La deuxième, la factorisation fonctionnelle, extrait les différentes fonctions présentes dans ces protocoles.

Ce chapitre est organisé en trois sections. Dans une première section (section 7.1) nous présentons un protocole abstrait de cohérence locale qui est une factorisation structurelle

des protocoles de cohérence locale de la littérature. Ensuite (section 7.2), nous présentons la factorisation fonctionnelle. Nous terminons ce chapitre par une conclusion (section 7.3).

7.1 Protocole abstrait de cohérence locale

Au vu de l'état de l'art sur les protocoles de duplication présenté au chapitre 2, on remarque que les protocoles de la littérature diffèrent principalement dans la manière et l'ordre dans lesquels ils accomplissent différentes phases. Nous définissons une phase comme un regroupement d'instructions ayant une certaine sémantique. Par exemple, le protocole ROWA (chapitre 2, section 2.3.2.1) ne se différencie de ROWAA que par l'exécution répétée par ce dernier de l'écriture sur les copies si l'une des copies défaille. De même, ROWA et un protocole de duplication paresseuse utilisé dans le contexte SGBDR (chapitre 2, section 2.3.2.2) se différencient par la phase d'écriture sur les copies qui est différée dans un cas (protocole paresseux) et qui est immédiate dans l'autre (ROWA). Ces phases sont identiques quel que soit le contexte (SGBDR, MPR, SCG, etc.).

Cette section est organisée de la manière suivante : nous commençons par présenter les cinq phases composant le protocole abstrait de cohérence locale (section 7.1.1), puis nous définissons cinq types d'ordonnancement des phases (section 7.1.2). Nous poursuivons par la présentation de la mise en œuvre du protocole abstrait de cohérence locale (section 7.1.3).

7.1.1 Cinq phases pour le protocole abstrait de cohérence locale

Les cinq phases du protocole abstrait de cohérence locale sont : la phase d'**accès**, la phase de **coordination**, la phase d'**exécution**, la phase de **validation** et la phase de **réponse**. Chacune des phases peut être mise en œuvre d'une façon ou d'une autre selon les besoins du protocole de cohérence locale. Chaque phase présente une interface générique. Les protocoles se différencient suivant l'approche utilisée pour chaque phase et l'ordre selon lequel elles sont exécutées. Les différentes phases et l'ordonnanceur qui leur est associé sont des éléments du protocole de cohérence locale (figure 7.1). Lors d'un accès à l'objet logique, l'application intéragit avec le protocole de cohérence locale par l'intermédiaire des interfaces présentées au chapitre 4.

Définition 7.1 : *Protocole abstrait de cohérence locale*
Le protocole abstrait de cohérence locale est une factorisation structurelle des protocoles de cohérence locale dégageant cinq phases génériques et leur ordonnancement.

Phase d'accès. Lors de la phase d'accès, l'objet client soumet une requête (une opération) à un objet dupliqué. La duplication est dite transparente si l'objet client intéragit avec l'objet logique et n'a pas conscience de l'existence des copies, leur nombre et leur localisation. Par

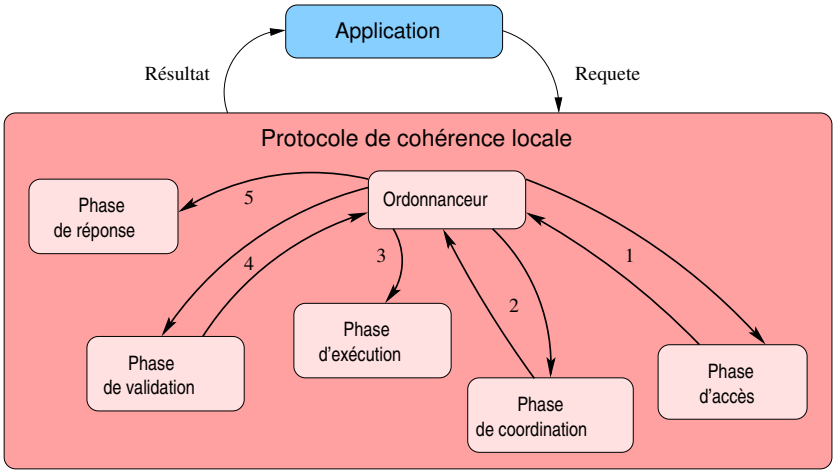


FIG. 7.1 – Protocole de cohérence locale et ordonnancement des phases

contre, si la duplication n'est pas transparente, l'objet client envoie ses requêtes directement à une ou plusieurs copies (peut être à toutes). Nous supposons que la duplication est transparente afin de ne pas déléguer des aspects propres à la duplication à l'application ou à d'autres aspects non fonctionnels.

Ainsi, lorsqu'une opération est soumise à un objet dupliqué, la phase d'accès a pour rôle de décider quelles sont les copies concernées : une copie particulière, un groupe de copies ou l'ensemble des copies. Lors de cette phase, il est décidé également quelles sont les copies pouvant recevoir des opérations en provenance de l'application. Elle traite également les messages pouvant être dupliqués suite à un appel provenant d'un autre objet dupliqué. Tous les traitements nécessaires lors de l'accès à un objet dupliqué font partie de cette phase. Enfin, cette phase peut discerner la nature de l'opération. Pour certains protocoles, une lecture ne concerne qu'une copie et une écriture l'ensemble des copies.

Phase de coordination. Cette phase inclut les traitements préliminaires sur les copies (coordination), avant l'exécution de la requête. La phase de coordination detient l'information permettant de construire les mises à jour à envoyer aux différentes copies, les copies à impliquer dans le processus de synchronisation, le moment de déclenchement de cette synchronisation, la (les) copie(s) à jour, etc. De plus, suivant le protocole mis en œuvre, si une copie est défaillante, il peut se réveler nécessaire de la supprimer de l'objet logique ou de la mettre en veille. Ce processus de coordination entre les copies peut également être déclenché sans qu'il y ait une requête provenant de l'extérieur. C'est le cas pour les protocoles où la synchronisation des copies est asynchrone.

Ces traitements préliminaires peuvent également dépendre de l'opération à exécuter (lecture ou écriture) mais également de la nature de l'opération (opération de lecture et écriture ou opération de contrôle). Par exemple, pour certains protocoles, une lecture nécessite la syn-

chronisation préalable de la copie afin d'obtenir la dernière valeur. Pour d'autres protocoles, l'ensemble de l'objet logique devra se synchroniser.

La phase de coordination peut également avoir des interactions avec le contrôle de concurrence et de tolérance aux fautes.

Phase d'exécution. Lors de cette phase, l'opération est effectivement exécutée sur la (les) copie(s). Cette phase prend en charge les liens avec les copies. Elle est capable d'accéder aux copies, de les charger ou de les décharger de la mémoire, etc.

Phase de validation. Cette phase inclut tous les traitements requis suite à l'exécution d'une requête. Il est vérifié que toutes les copies (ou une majorité) sont d'accord sur le résultat de l'exécution. Il peut être décidé que l'exécution n'est pas correcte si une ou plusieurs copies ne répondent pas. Le rôle de cette phase est également de décider s'il faut défaire ou refaire certains traitements.

La phase de validation peut inclure des interactions avec le contrôle de concurrence et de tolérance aux fautes.

Phase de réponse. Cette phase décide du résultat à renvoyer. La réponse peut être renvoyée une fois que tous les traitements ont été réalisés ou le plus vite possible, même si tout n'a pas été effectué sur toutes les copies. La phase de réponse décide de la valeur de l'objet logique suite à une opération.

7.1.2 Cinq patrons d'ordonnancement

Les protocoles diffèrent par la manière dont chaque phase est implantée mais aussi par l'ordre dans lequel elles apparaissent. Cet ordre est mis en œuvre par l'ordonnanceur (figure 7.1). Pour certains protocoles, des phases sont inexistantes (en pointillé sur la figure 7.2), des cycles entre les phases apparaissent, ou bien encore elles peuvent s'exécuter en parallèle. On peut distinguer cinq protocoles d'ordonnancement entre les phases :

Protocole de type (AER). Dans ce protocole (figure 7.2 (a)) une demande faite par un client est traitée de manière séquentielle par la phase d'accès, puis par la phase d'exécution, et enfin par la phase de réponse. Le protocole ne comporte pas de phase de coordination et de validation.

Protocole de type (ACER). Dans ce protocole, (figure 7.2 (b)) on intercale par rapport au protocole précédent une phase de coordination entre la phase d'accès et la phase d'exécution.

Protocole de type (A(CEV)*R). Dans ce protocole (figure 7.2 (c)), une demande faite par un client est d'abord traitée par la phase d'accès, puis la phase de coordination, la phase d'exécution, la phase de validation, et enfin la phase de réponse. Il peut y avoir une cycle suite à la phase de validation ramenant à une phase de coordination afin de traiter à nouveau la requête.

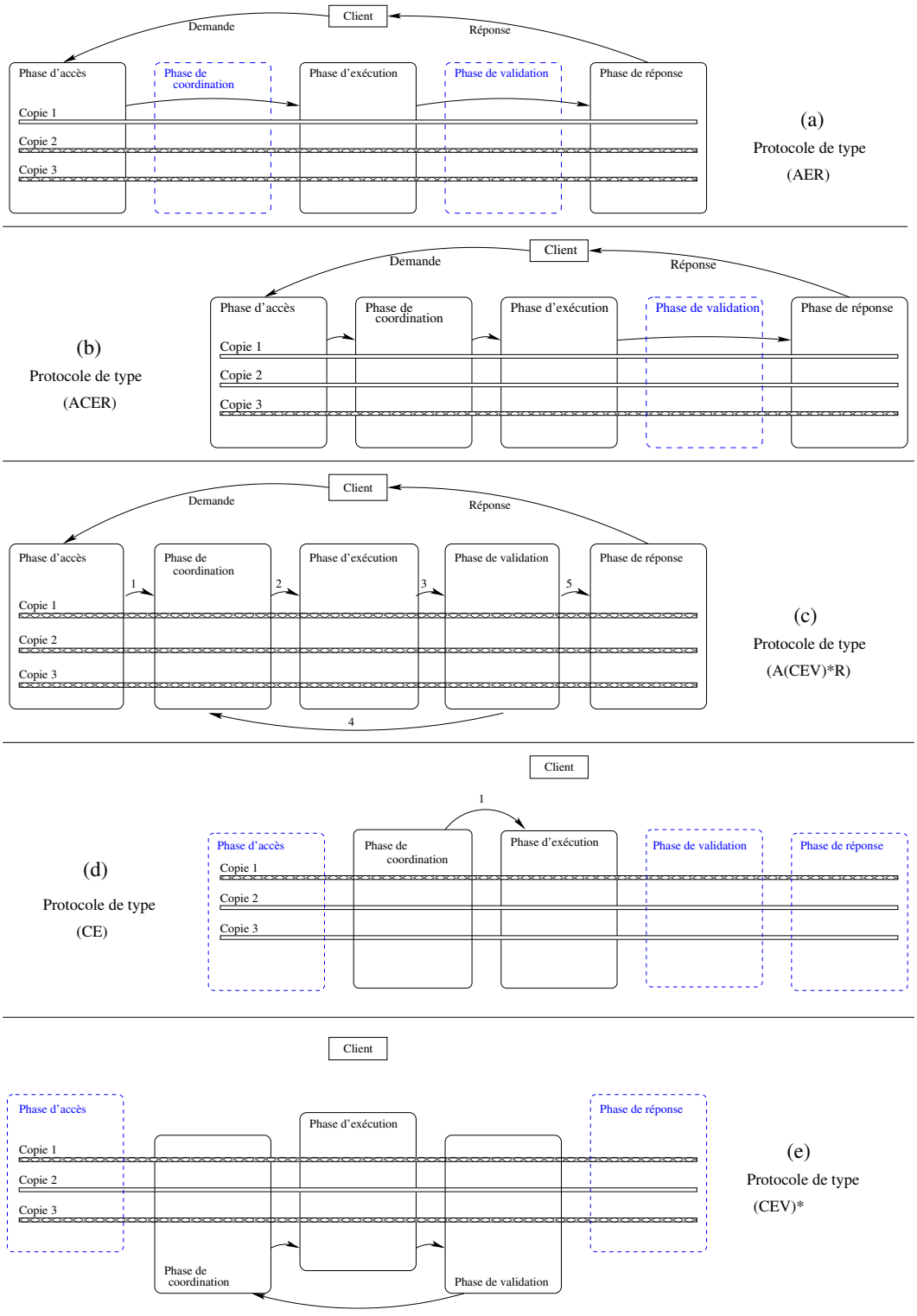


FIG. 7.2 – Ordonnancements possibles des cinq phases

Protocole de type (CE). Dans ce protocole (figure 7.2 (d)) une opération est d'abord traitée par la phase de coordination puis par la phase d'exécution. Ce protocole se justifie dans le cas des protocoles de cohérence locale où la synchronisation des copies se fait de manière asynchrone.

Protocole de type (CEV)*. Dans ce protocole (figure 7.2 (e)) une opération est d'abord traitée par la phase de coordination, puis la phase d'exécution et enfin la phase de validation. Si cette dernière phase échoue, il est possible de revenir à la phase de coordination. Ce protocole se justifie dans le cas des protocoles de cohérence locale où la synchronisation des copies se fait de manière asynchrone.

Ces cinq patrons proviennent de notre étude de l'état de l'art. Ils nous paraissent couvrir un bon nombre de protocoles existants. Cependant, le canevas ayant une approche ouverte, il est tout à fait possible de proposer d'autres patrons qui seront mis en œuvre par des ordonnanceurs spécifiques.

7.1.3 Construction du protocole abstrait de cohérence locale

Cette section introduit les interfaces des différents éléments participant à un protocole de cohérence locale. La figure 7.3 donne un aperçu des interactions entre ces différents éléments.

De la répartition. Par nature un protocole de cohérence locale est réparti ; une approche centralisée présentant un intérêt limité. Généralement, la répartition du protocole correspond à la répartition des copies, c'est à dire que la mise en œuvre du protocole de cohérence locale est répartie sur chacune des copies. Ainsi, chacune des phases peut être répartie.

Nous nommons représentant d'une phase la mise en œuvre locale d'une phase. Dans la suite, nous parlons de phase pour parler des représentants en général. Afin de construire une phase, les différents représentants de chacune des phases doivent pouvoir communiquer entre eux. Chaque représentant de phase implante une interface ascendante et une descendante afin de communiquer avec ses pairs :

Interface de communication descendante (*downcall*) :

- `send(Message m, Iterator it)` permet d'envoyer un message `m` au groupe de copies `it`.
- `send(Message m, Id id)` permet d'envoyer un message `m` à la copie désignée par `id`.

∎

Interface de communication ascendante (*upcall*) :

- `receive(Message m)` permet de recevoir un message `m`.

∎

Les représentants de phase s'appuient sur deux composants faisant le lien avec la couche communication : le gestionnaire de groupe de communication et le gestionnaire de diffusion.

Le gestionnaire de groupe prend en charge le cycle de vie d'un groupe d'objets et maintient la liste des membres. Il fournit un support pour joindre et quitter le groupe. Cette fonctionnalité est offerte par le service de communication ou bien peut être fournie par RS2.7.

Interface de gestionnaire de groupe de communication :

- `join(Id id)` ajoute une copie, identifiée par `id`, au groupe.
- `leave(Id id)` enlève une copie, identifiée par `id`, du groupe.
- `removeFailedMember(Id id)` supprime une copie défaillante, identifiée par `id`, du groupe.
- `Iterator getGroup()` renvoie un Iterator sur l'ensemble des copies du groupe.

Le gestionnaire de diffusion fournit un support pour l'envoi de messages à tous les membres du groupe. Cet envoi peut se faire selon diverses qualités de service (fiable ou non, certains ordres). Cette fonctionnalité est offerte par le service de communication ou bien peut être fournie par RS2.7.

Interface du gestionnaire de diffusion :

- `sendAll(Message m, Iterator it)` permet d'envoyer un message `m` au groupe de copies `it`.

L'ordonnanceur. L'ordonnanceur reçoit les opérations capturées par le protocole de cohérence locale et les transmet aux différentes phases. Une fois qu'une phase a terminé son traitement elle renvoie ses résultats à l'ordonnanceur. Celui-ci décide ensuite quelle phase doit poursuivre le traitement.

L'ordonnanceur est également réparti. On nomme représentant de l'ordonnanceur la mise en œuvre locale d'un ordonnanceur. Un représentant ordonne les appels entre les représentants de phase situés sur un même site. Il n'y a pas de communication entre les représentants d'ordonnanceur, la communication se faisant par l'intermédiaire des phases.

Les patrons de la section 7.1.2 décrivent l'ordonnancement des phases. Il existe également une deuxième dimension dans l'ordonnancement, les phases pouvant communiquer entre elles, les représentants d'ordonnanceur agissent en parallèle. Ils sont ordonnancés suivant les échanges entre les représentants de phases. Par exemple, dans le cas du protocole (AER), un représentant de phase d'accès peut envoyer un message à un groupe d'autres représentants de phases d'accès. Chaque représentant d'ordonnanceur peut à son tour se voir solicité par son représentant de phase d'accès et ensuite appeler le représentant de la phase d'exécution puis celui de la phase de réponse. De plus, pour un protocole de cohérence locale donné, chaque représentant d'ordonnanceur peut suivre un protocole d'ordonnancement différent.

Le protocole abstrait de cohérence locale fait ressortir la structure des protocoles de cohérence locale. Les informations échangées entre les représentants de phases et l'ordonnanceur portent sur la nature des opérations. Dans la suite, on utilise la structure de données `Operation` désignant l'ensemble des opérations. `Operation` contient un identifiant unique, le

nom de l'opération, ses arguments et le type de l'opération. Ce type peut être `READ, WRITE, METHOD` pour désigner les opérations classiques et `CONTROL` pour désigner les opérations de contrôle. Les opérations de contrôle sont toutes les opérations qui permettent de gérer les interactions avec d'autres aspects non-fonctionnels (`readIntention, writeIntention`, etc.).

L'information circulant entre les phases et l'ordonnanceur concerne les opérations à traiter d'où des interfaces similaires.

Dans la suite, nous présentons pour chaque représentant de phase son interface, la nature des messages échangés entre les représentants d'une phase et l'interface avec le représentant de l'ordonnanceur.

Représentant de la phase d'accès. Lors de cette phase la requête à traiter est distribuée aux différentes copies. Son interface, utilisée par l'ordonnanceur, est la suivante :

Interface du représentant de la phase d'accès :

- `readOperation(Operation op)` informe le représentant de la phase d'accès qu'une opération de lecture `op` est soumise à l'objet logique.
- `writeOperation(Operation op)` informe le représentant de la phase d'accès qu'une opération d'écriture `op` est soumise à l'objet logique.
- `operation(Operation op)` informe le représentant de la phase d'accès qu'une opération `op` est soumise à l'objet logique. L'opération peut être une écriture, une lecture ou une méthode à exécuter.
- `controlOperation(Operation op)` informe le représentant de la phase d'accès qu'une opération de contrôle `op` est à traiter.

∎

Un représentant de phase peut envoyer l'opération à d'autres représentants de phase. Les messages échangés entre les représentants des phases d'accès contiennent les opérations à traiter.

Une fois le traitement du représentant de la phase d'accès terminé, chaque représentant renvoie l'opération à traiter à son représentant de l'ordonnanceur. Ce dernier présente l'interface suivante utilisée par la phase d'accès :

Interface du représentant de l'ordonnanceur PA :

- `readOperationA(Operation op)` informe le représentant d'ordonnanceur que la phase d'accès à traiter est une opération de lecture `op`.
- `writeOperationA(Operation op)` informe le représentant d'ordonnanceur que la phase d'accès à traiter est une opération d'écriture `op`.
- `operationA(Operation op)` informe le représentant d'ordonnanceur que la phase d'accès à traiter est une opération `op`. L'opération peut être une écriture, une lecture ou une méthode à exécuter.
- `controlOperationA(Operation op)` informe le représentant d'ordonnanceur que la phase d'accès à traiter est une opération de contrôle `op`.

∎

Représentant de la phase de coordination. Lors de cette phase le protocole coordonne les différentes copies. L'interface de cette phase doit lui permettre d'être informée des opérations ou des opérations de contrôle à traiter. Cette interface, utilisée par l'ordonnanceur, est la suivante :

Interface du représentant de la phase de coordination :

- `readOperation(Operation op)` informe le représentant de la phase de coordination qu'une opération de lecture `op` est soumise à la copie.
- `writeOperation(Operation op)` informe le représentant de la phase de coordination qu'une opération d'écriture `op` est soumise à la copie.
- `operation(Operation op)` informe le représentant de la phase de coordination qu'une opération `op` est soumise à la copie. L'opération peut être une écriture, une lecture ou une méthode à exécuter.
- `controlOperation(Operation op)` informe le représentant de la phase de coordination qu'une opération de contrôle `op` est soumise.

∎

La phase de coordination peut être répartie. Une copie particulière peut diffuser ses mises à jour à d'autres copies (protocole de type diffusion) : à une copie en particulier, à un groupe de copies ou à toutes. Elle peut également demander des mises à jour à d'autres copies (protocole de type demande/diffusion). Les messages échangés contiennent donc des informations de synchronisation ou des demandes de synchronisation.

Une fois le traitement de la phase de coordination terminé, celle-ci renvoie au représentant de l'ordonnanceur les opérations à exécuter suite à la coordination. Cette opération peut être la requête initiale ou des opérations nécessaires à la synchronisation (opérations décidées par le représentant de la phase de coordination). L'ordonnanceur présente l'interface suivante utilisée par la phase de coordination :

Interface du représentant de l'ordonnanceur PC :

- `readOperationC(Operation op)` informe le représentant de l'ordonnanceur qu'une opération de lecture `op` est soumise par la phase de coordination.
- `writeOperationC(Operation op)` informe le représentant de l'ordonnanceur qu'une opération d'écriture `op` est soumise par la phase de coordination.
- `operationC(Operation op)` informe le représentant de l'ordonnanceur qu'une opération `op` est soumise par la phase de coordination. L'opération peut être une écriture, une lecture ou une méthode à exécuter.
- `controlOperationC(Operation op)` informe le représentant de l'ordonnanceur qu'une opération de contrôle `op` est soumise par la phase de coordination.

∎

Représentant de la phase d'exécution. Cette phase exécute la requête sur la copie. Son interface, utilisée par l'ordonnanceur, est la suivante :

Interface du représentant de la phase d'exécution :

- `readOperation(Operation op)` informe le représentant de la phase d'exécution qu'une opération de lecture `op` doit être exécutée sur la copie.
- `writeOperation(Operation op)` informe le représentant de la phase d'exécution qu'une opération d'écriture `op` doit être exécutée sur la copie.
- `operation(Operation op)` informe le représentant de la phase d'exécution qu'une opération `op` doit être exécutée sur la copie. L'opération peut être une écriture, une lecture ou une méthode à exécuter.
- `readOperationTentative(Operation op)` informe le représentant de la phase d'exécution qu'une opération de lecture `op` doit être exécutée mais non répercutée sur la copie.
- `writeOperationTentative(Operation op)` informe le représentant de la phase d'exécution qu'une opération d'écriture `op` doit être exécutée mais non répercutée sur la copie.
- `operationTentative(Operation op)` informe le représentant de la phase d'exécution qu'une opération `op` doit être exécutée mais pas répercutée sur la copie. L'opération peut être une écriture, une lecture ou une méthode à exécuter.

∎

On distingue les opérations devant être effectivement exécutées de celles devant uniquement être simulées (opérations `Tentative`). Cela se révèle nécessaire pour certains protocoles où, avant de répercuter des modifications sur les copies, il faut vérifier que tout s'est déroulé convenablement.

Une fois le traitement de la phase d'exécution terminé, celle-ci renvoie des informations sur le déroulement de l'exécution à l'ordonnanceur. Ce dernier présente l'interface suivante utilisée par la phase d'exécution :

Interface du représentant de l'ordonnanceur PE :

- `resultE(Operation op, State state)` donne au représentant de l'ordonnanceur l'état `state` de la copie suite à l'opération `op`.
- `resultE(Operation op, Field field, Value value)` donne au représentant de l'ordonnanceur la valeur `value` du champ `field` de la copie suite à l'opération `op`.

∎

Représentant de la phase de validation. Cette phase vérifie les traitements réalisés sur les copies. La vérification peut être explicitement demandée (`verify`) ou peut se produire suite à une opération de contrôle. L'interface de la phase de validation utilisée par l'ordonnanceur est la suivante :

Interface du représentant de la phase de validation :

- `verify(Operation op, State state)` informe le représentant de la phase de validation de l'état `state` de la copie suite à l'opération `op`.
- `verify(Operation op, Field field, Value value)` informe le représentant de la phase de validation de la valeur `value` du champ `field` de la copie suite à l'opération `op`.

– `controlOperation(Operation op)` informe le représentant de la phase de validation d'une opération de contrôle `op`.

∎

La phase de validation peut être répartie. Le processus de vérification peut être fait sur une copie particulière qui demande à d'autres copies leur état afin de décider s'il y a validation. Les messages échangés entre les copies contiennent les opérations traitées ainsi que les résultats obtenus.

Une fois le traitement de la phase de validation terminé, celle-ci renvoie le resultat de son analyse à l'ordonnanceur. Celui-ci présente l'interface suivante utilisée par la phase de validation :

Interface du représentant de l'ordonnanceur PV :

– `resultTrue(Operation op)` informe le représentant de l'ordonnanceur que l'opération `op` s'est déroulée convenablement.
– `resultTrue(Operation op, Operation op2, Replicas rs)` informe le représentant de l'ordonnanceur que l'opération `op` s'est déroulée convenablement, mais qu'il faut exécuter l'opération `op2` sur l'ensemble de copies `rs`.
– `resultFalse(Operation op, Replicas rs)` informe le représentant de l'ordonnanceur que l'opération `op` ne s'est pas déroulé convenablement sur l'ensemble de copies `rs`.
– `resultFalse(Operation op)` informe le représentant de l'ordonnanceur que l'opération `op` ne s'est pas déroulé convenablement.

∎

Représentant de la phase de réponse. Cette phase renvoie la valeur de l'objet logique. Elle peut être répartie afin de décider quelle réponse renvoyer et/ou à quel moment. La phase de réponse présente l'interface suivante à l'ordonnanceur :

Interface du représentant de la phase de réponse :

– `resultFor(Operation op, Field f, Value v)` donne le résultat `v` pour l'opération `op` faite sur le champ `f` de la copie.

∎

Une fois le traitement de la phase de réponse terminé, celle-ci renvoie le résultat à l'ordonnanceur. Ce dernier présente l'interface suivante utilisée par la phase de réponse :

Interface du représentant de l'ordonnanceur PR :

– `result(Operation op, State state)` donne au représentant de l'ordonnanceur la valeur de l'état `state` de l'objet logique suite à l'opération `op`.
– `result(Operation op, Field field, Value value)` donne au représentant de l'ordonnanceur la valeur `value` du champ `field` de l'objet logique suite à l'opération `op`.

∎

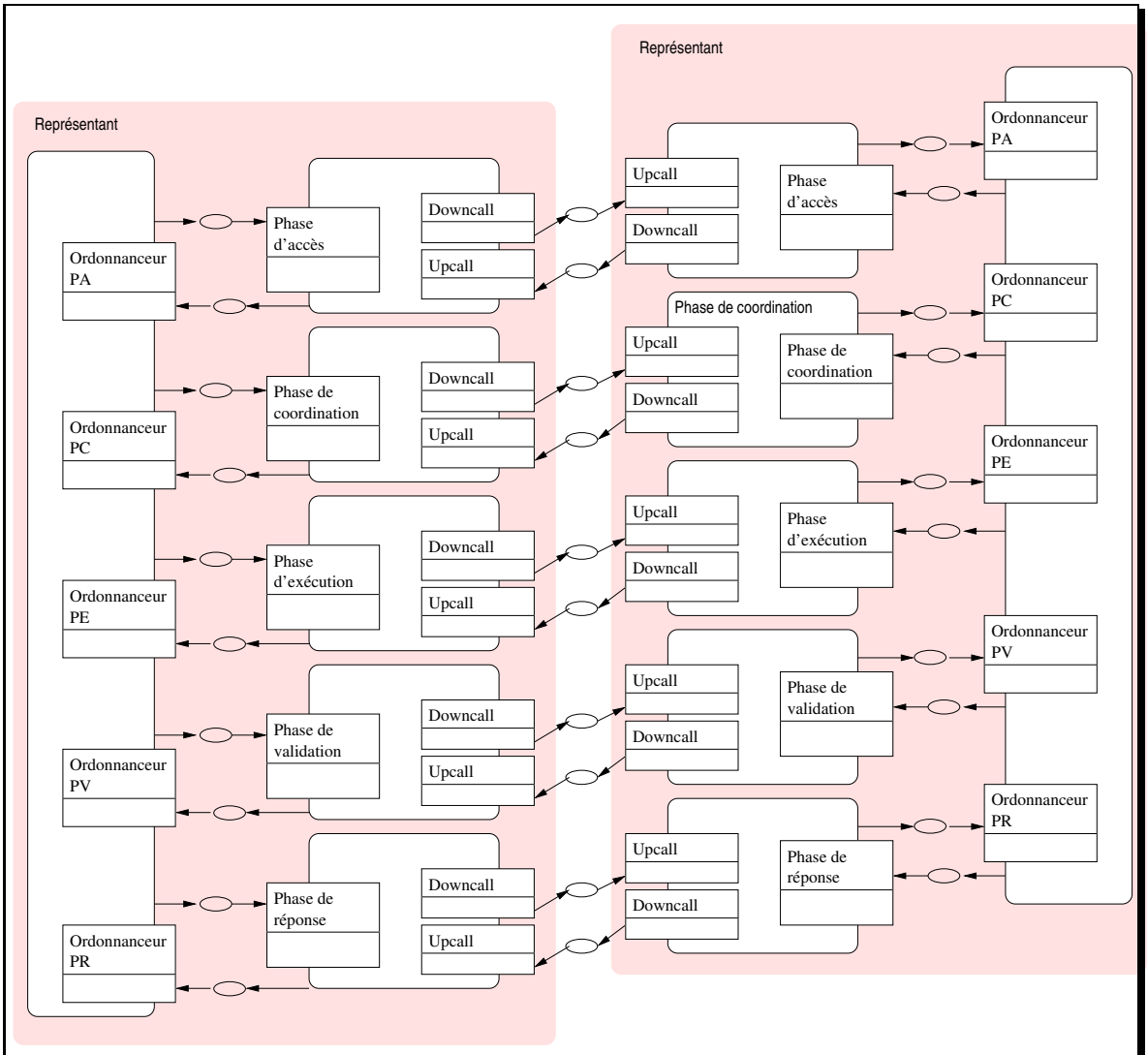


FIG. 7.3 – Structure des protocoles de cohérence locale

7.2 Architecture fonctionnelle des protocoles de cohérence locale

Le protocole de cohérence locale abstrait ne fait pas apparaitre les fonctionnalités offertes par les protocoles. Dans cette section nous extrayons les fonctionnalités communes des protocoles de cohérence locale, afin d'obtenir une architecture fonctionnelle discernant les composants impliqués dans la construction des protocoles de cohérence locale. Ces composants fonctionnels servent à mettre en œuvre les phases présentées dans la section précédente (section 7.1). Chaque composant propose une interface couvrant une fonction particulière

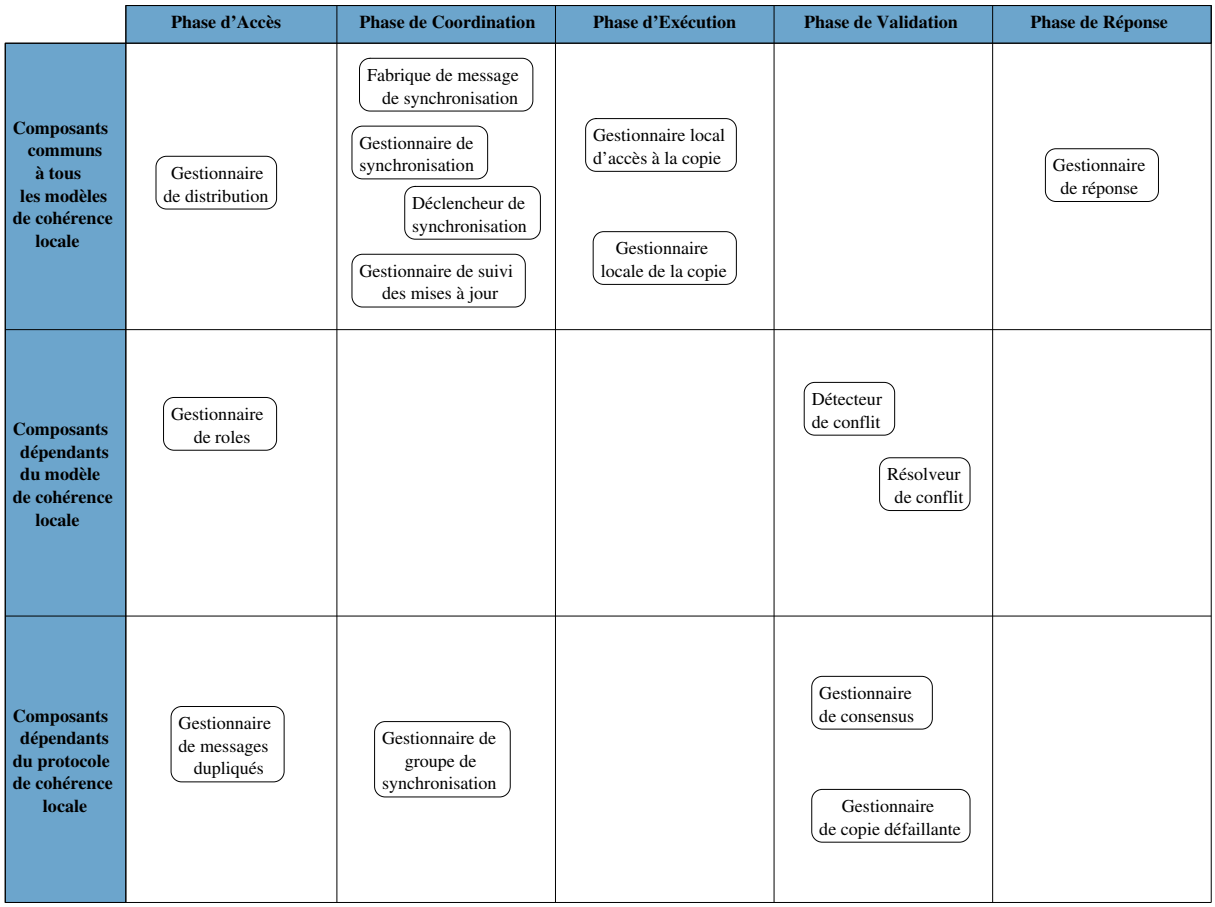


FIG. 7.4 – Architecture fonctionnelle des protocoles de cohérence locale

pouvant être implantée de différentes façons. Ces composants reprennent les points présentés dans le chapitre d'état de l'art sur les protocoles de duplication (chapitre 2, section 2.1). L'architecture fonctionnelle fournit trois catégories de composants (figure 7.4) :

- **Les composants communs à tous les modèles de cohérence locale** (section 7.2.1) sont considérés comme le niveau élémentaire pour construire des protocoles simples. Principalement, ces composants concernent la gestion du cycle de vie des copies, la synchronisation et les intéractions avec l'application utilisatrice.
- **Les composants dépendant du modèle de cohérence locale** (section 7.2.2).
- **Les composants dépendant du protocole de cohérence locale** (section 7.2.3).
  Ces deux catégories ajoutent des composants dépendants des modèles et des protocoles.

Cette classification sert principalement à organiser la présentation des composants fonctionnels.

7.2.1 Composants communs aux modèles

Ces composants matérialisent les fonctionnalités de base des protocoles de cohérence locale. Nous isolons des fonctionnalités de ce type pour les phases d'accès, de coordination, d'exécution et de réponse. Il n'y en a pas pour la phase de validation, car n'intervenant pas dans de nombreux protocoles.

7.2.1.1 Phase d'accès

Le gestionnaire de distribution est le seul composant de la phase d'accès à ce niveau.

Gestionnaire de distribution. Le gestionnaire de distribution décide vers quelles copies il faut retransmettre les requêtes provenant de l'extérieur. Suivant l'objectif, il peut retransmettre cette requête vers une copie, un groupe de copies ou à l'ensemble des copies (chapitre 2, section 2.1.1).

Le gestionnaire de distribution peut faire une distinction suivant la nature des opérations. Par exemple, pour certains protocoles une lecture n'est envoyée qu'à une copie et une écriture à l'ensemble des copies. D'autres protocoles peuvent propager une opération de contrôle vers l'ensemble des copies, un groupe ou vers une copie.

Interface du gestionnaire de distribution :

- `Group readOperation(Operation op)` retourne le groupe de copies auquel renvoyer l'opération de lecture `op`.
- `Group writeOperation(Operation op)` retourne le groupe de copies auquel renvoyer l'opération d'écriture `op`.
- `Group operation(Operation op)` renvoie le groupe de copies auquel renvoyer l'opération `op`. L'opération peut être une écriture, une lecture ou une méthode à exécuter.
- `Group controlOperation(Operation op)` renvoie le groupe de copies auquel renvoyer l'opération de contrôle `op`.

∎

7.2.1.2 Phase de coordination

Dans cette phase, on trouve un ensemble de composants permettant de synchroniser les copies : la fabrique de messages de synchronisation, le déclencheur de synchronisation, le gestionnaire de suivi des mises à jour et le gestionnaire de synchronisation.

Déclencheur de synchronisation Le déclencheur de synchronisation décide du moment de la synchronisation. Pour cela, suivant le protocole de cohérence locale à mettre en œuvre il peut s'appuyer sur l'occurence d'une lecture, d'une écriture, ou d'une opération survenue ou alors s'appuyer sur des evénements externes (par exemple, la synchronisation doit être

déclenchée à heure fixe) ou sur des événements internes aux protocoles de cohérence locale (par exemple, il n'y a pas eu de synchronisation depuis n requêtes externes). De manière générale, il est le garant des conditions présentées en section 2.1.3 au chapitre 2.

Interface du déclencheur de synchronisation :

- `externEvent(Event event)` informe le déclencheur de synchronisation de l'occurrence d'un événement externe `event` pouvant déclencher le processus de coordination.
- `internEvent(Event event)` informe le déclencheur de synchronisation de l'occurrence d'un événement `event` interne pouvant déclencher le processus de coordination.
- `operation(Operation op)` informe le déclencheur de synchronisation de l'occurrence d'un événement interne pouvant déclencher le processus de coordination. Cet événement interne est une opération `op` pouvant être une lecture, une écriture ou une méthode a exécuter.
- `controlOperation(Operation op)` informe le déclencheur de synchronisation de l'occurrence d'un événement interne pouvant déclencher le processus de coordination. Cet événement interne `op` est une opération de contrôle.

∎

Le déclenchement du processus de synchronisation peut être asynchrone. Le déclencheur informe la phase de coordination de la nécessité de déclencher la synchronisation en utilisant l'interface suivante (proposée par la phase de coordination) :

Interface de synchronisation PC :

- `synchronize()` informe la phase de coordination qu'il est necessaire de déclencher le processus de coordination.

∎

Gestionnaire de suivi des mises à jour Le gestionnaire de suivi des mises à jour garde une trace de ce qui est fait sur les copies. Cette information sert ensuite à construire les messages devant être échangés lors des synchronisations. Il peut être implanté par divers mécanismes comme des journaux, des triggers, des photographies, des copies ombres, etc (chapitre 2, section 2.1.7). Dans tous les cas, ce composant à besoin d'être informé de ce qui se passe sur les copies. L'information de synchronisation peut être un état ou une opération.

Interface du gestionnaire de suivi des mises à jour :

- `readOperation(Operation op)` informe le gestionnaire de suivi des mises à jour de l'occurrence d'une opération de lecture `op`.
- `writeOperation(Operation op)` informe le gestionnaire de suivi des mises à jour de l'occurrence d'une opération d'écriture `op`.
- `operation(Operation op)` informe le gestionnaire de suivi des mises à jour de l'occurrence d'une opération `op`. L'opération `op` peut être une lecture, une écriture ou une méthode a exécuter.
- `controlOperation(Operation op)` informe le gestionnaire de suivi des mises à jour de l'occurrence d'une opération de contrôle `op`.

- `setMark(Mark m)` permet de marquer un point `m` précis dans le suivi des mises à jour.
- `Operations getModifications()` renvoie les modifications faites sur une copie sous forme d'opérations.
- `Operations getModifications(Mark m)` renvoie les modifications faites sur une copie à partir du point `m` sous forme d'opérations.
- `State getModification()` renvoie les modifications faites sur une copie sous forme d'un état.
- `State getModification(Mark m)` renvoie les modifications faites sur une copie sous forme d'un état au moment marqué par le point `m`.

∎

Fabrique de messages de synchronisation. Ce composant se charge de la construction des messages de synchronisation. Suivant les cas, il récupère les informations de synchronisation auprés du gestionnaire de suivi des mises à jour ou bien il construit un message de demande de synchronisation (chapitre 2, section 2.1.4 et section 2.1.5).

Interface de la fabrique de message de synchronisation :

- `Message synchronisationMessage()` construit un message de synchronisation.
- `Message synchronisationRequest()` construit une demande de synchronisation.

∎

7.2.1.3 Phase d'execution

Dans la phase d'exécution on identifie deux composants : le gestionnaire local d'accès à la copie et le gestionnaire local de la copie. Ces deux composants contribuent à la générécité de notre canevas car ils permettent de dupliquer n'importe quel type d'objet (aussi bien des objets simples ou composites que des pages HTML par exemple).

Gestionnaire local d'accès à la copie. Le gestionnaire local d'accès à la copie offre une interface permettant les lectures/écritures sur une copie. Ce composant donne une représentation abstraite des copies.

Interface du gestionnaire local d'accès à la copie :

- `Value read(Field f)` lit le champ `f` de l'objet dupliqué.
- `write(Field f, Value v)` écrit la valeur `v` dans le champ `f` de l'objet dupliqué.
- `executeMethod(Name n, Arguments as)` exécute la méthode `n` avec l'ensemble d'arguments `as` sur l'objet dupliqué.

∎

Gestionnaire local de la copie. Le gestionnaire local de la copie encapsule tout ce qui a attrait à la gestion d'une copie (par exemple, le chargement de la copie en mémoire). Avant d'activer ou de passiver une copie, suivant le protocole de cohérence locale, des traitements peuvent être nécessaires.

Interface du gestionnaire local de la copie :

- `activate()` active une copie.
- `passivate()` passive une copie.

▮

7.2.1.4 Phase de réponse

La phase de réponse a un seul composant : le gestionnaire de réponse.

Composant de gestion de réponse. Le rôle de ce composant est de décider du résultat à retourner pour la requête. Il décide de la valeur reflétant l'objet logique. La valeur de l'objet logique ne représente pas forcément la valeur de toutes les copies. Cela dépend du modèle de cohérence locale mis en œuvre. Ainsi, suivant les cas, ce composant peut attendre les réponses d'un sous ensemble des copies, d'une copie particulière ou de l'ensemble des copies.

Suivant le protocole à mettre en œuvre, les gestionnaires de réponse des différents représentants de phase peuvent communiquer entre eux par l'intermédiaire des représentants de phase. Dans d'autre cas, il n'y a qu'un représentant de la phase de réponse possédant un gestionnaire de réponse.

Interface du gestionnaire de réponse :

- `resultFor(Operation op, Field f, Value v)` donne le résultat `v` pour l'opération `op` faite sur le champ `f` de la copie.
- `State getState(Operation op)` renvoie l'état de l'objet logique suite à l'opération `op`.
- `getField(Operation op, Field f, Value v)` renvoie la valeur `v` du champ `f` de l'objet logique suite à l'opération `op`.

▮

7.2.2 Composants dépendants du modèle de cohérence locale

Cette catégorie introduit l'ensemble de composants dépendants du type de modèle de cohérence locale à mettre en œuvre.

7.2.2.1 Phase d'accès

Les modèles de cohérence locale se caractérisent par le rôle des copies. Ainsi, on introduit un nouveau composant, le gestionnaire de rôles, décidant ce qu'a le droit de traiter une copie particulière.

Gestionnaire de rôle. Le rôle d'une copie caractérise ce qu'elle à le droit de faire suite à une requête externe.

Par exemple, ce rôle peut spécifier les copies pouvant être mises à jour par une requête externe et celles devant l'être exclusivement par d'autres copies (chapitre 2, section 2.1.2). En effet, deux cas sont envisageables dans la littérature : l'approche maître/esclaves et l'approche peer-to-peer. L'approche peer-to-peer permet à n'importe quelle copie d'être mise à jour par une requête externe, les modifications étant eventuellement transmises aux autres copies par la suite. A l'opposé, l'approche maître/esclaves distingue une (des) copie(s) maîtresse(s) pouvant être modifiée(s) par des requêtes externes et des copies esclaves uniquement mises à jour par le(s) maître(s). Plusieurs mises en œuvre sont possibles. Dans le cas d'une approche maître-esclaves, si une copie esclave reçoit une requête d'écriture, ce composant peut renvoyer une erreur ou renvoyer la requête au maître.

On peut également imaginer un protocole inverse de l'approche maître-esclaves où toutes les copies ont le droit d'écrire, mais une seule peut être lue, les autres copies ne faisant que de la sauvegarde.

Interface du gestionnaire de rôles :

- `Boolean requestTreatment(Operation op)` renvoie vrai si l'opération `op` peut être traitée par la copie.
- `Id requestTreatment(Operation Op)` renvoie l'identificateur de la copie devant traiter l'opération `op`.

∎

7.2.2.2 Phase de validation

Le traitement des modifications conflictuelles requiert l'introduction de nouveaux composants dans la phase de validation : le détecteur de conflit et le résolveur de conflit. Ces composants sont utilisés par des protocoles implantant le modèle de cohérence locale à copies convergentes avec écriture sur des copies divergentes. Ces protocoles doivent être à même de détecter les conflits et de les résoudre.

Il est à noter également que des intéractions avec le contrôle de concurrence peuvent être nécessaires au bon fonctionnement de ces composants.

Détecteur de conflit. Ce composant se charge d'apporter toute l'information nécessaire pour détecter les conflits. Pour cela, les protocoles existants offrent différentes politiques de résolution des conflits (basé sur des estampilles temporelles, des priorités, additive, maximum) pouvant être utilisées pour l'implantation de ces composants (chapitre 2, section 2.1.8). Il est possible qu'il intéragisse avec le service de contrôle de concurrence.

Interface du détecteur de conflit :

- `verify(Operation op, State state)` informe le détecteur de conflit de l'occurence d'une opération `op` renvoyant l'état `state` .
- `verify(Operation op, Field field, Value value)` informe le détecteur de conflit de l'occurence d'une opération `op` renvoyant la valeur `value` du champ `field`.
- `controlOperation(Operation op)` informe le détecteur de conflit de l'occurence d'une opération de contrôle `op`.
- `Conflict getConflict()` renvoie le conflit existant.

∎

Résolveur de conflit. Ce composant apporte toute l'information nécessaire à la résolution d'un conflit (chapitre 2, section 2.1.8).

Interface du résolveur de conflit :

- `resolve(Conflict c, Operations ops, Group g)` donne les opérations `ops` à exécuter sur les différentes copies du groupe `g` afin de résoudre le conflit `c`.

∎

7.2.3 Composants dépendants des protocoles

Cette catégories introduit des composants spécifiques à certains protocoles de cohérence locale.

7.2.3.1 Phase d'accès

On trouve ici un composant, le gestionnaire de messages dupliqués.

Gestionnaire de messages dupliqués. Pour certains protocoles, il est nécessaire de gérer le problème des appels dupliqués lors de la phase d'accès : problème soulevé lorsqu'un objet dupliqué A envoie une requête à un autre objet dupliqué B (si A est composé de n copies, B peut recevoir n copies de la requête). Cette situation peut se gérer de différentes façons, du coté du client (A) ou du coté du serveur (B). Le composant gestionnaire de messages dupliqués gére ce problème.

Interface du gestionnaire d'appels dupliqués :

- `Boolean newOperation(Opération operation)` renvoie vrai si l'opération peut être traitée. Pour certains protocoles, si l'opération est en cours de traitement ou a déjà était traitée cette fonction renvoie faux.

∎

7.2.3.2 Phase de coordination

Le gestionnaire de groupe de synchronisation est introduit comme composant de la phase de coordination.

Gestionnaire du groupe de synchronisation. Ce composant décide quelles sont les copies participant au processus de synchronisation. Par exemple, pour les protocole à base de quorum, le processus de coordination ne concerne qu'une certaine partie des copies pour une écriture et une autre pour une lecture.

Interface du gestionnaire du groupe de synchronisation :

- `Group readOperation(Operation op)` retourne le groupe de copies auquel renvoyer l'opération de lecture `op`.
- `Group writeOperation(Operation op)` retourne le groupe de copies auquel renvoyer l'opération d'écriture `op`.

∎

7.2.3.3 Phase de validation

Pour certains protocoles, lors de la phase de validation, certaines actions en rapport avec la tolérance aux fautes sont nécessaires. Deux composants sont introduit pour cela : le gestionnaire de consensus et le gestionnaire de copie défaillante.

Gestionnaire de consensus. Ce composant à pour charge d'obtenir un consensus.

Interface du gestionnaire de consensus :

- `State consensus(Operation op, State s)` permet d'obtenir un consensus sur l'état `s` suite à l'opération `op`
- `Value consensus(Operation op, Field f, Value v)` permet d'obtenir un consensus sur la valeur `v` du champ `f` suite à l'opération `op`.

∎

Gestionnaire de copie défaillante. Ce composant intéragit avec le service de tolérance aux fautes qui détecte les copies défaillantes. Il décide de ce qu'il faut faire de ces copies. Une copie défaillante peut être supprimée de l'objet logique définitivement, temporairement, etc. Une copie de nouveau opérationnelle quant à elle, pour certains protocoles, doit rattrapper son retard afin de réintégrer l'objet logique.

Interface du gestionnaire de copie défaillante :

- `deadReplica(Id id)` informe le gestionnaire de copie défaillante de la défaillance de la copie désignée par `id`.

– `State aliveReplica(Id id)` informe le gestionnaire de copie défaillante que la copie désignée par `id` est de nouveau opérationnelle. Le gestionnaire de copie défaillante renvoie l'état à jour de la copie.

∎

7.3 Conclusion

RS2.7 permet d'obtenir des protocoles de cohérence locale très différents mettant en œuvre les quatre types de modèle de cohérence locale présentés au chapitre 5. Afin d'obtenir l'adaptabilité dans les protocoles supportés nous proposons deux factorisations des protocoles de cohérence locale faisant ressortir leurs caractéristiques.

Le protocole abstrait de cohérence locale, factorisation de nature structurelle, définit cinq phases : l'accès, la coordination, l'exécution, la validation et la réponse. Une phase caractérise une unité sémantique permettant d'isoler un élément de la structure des protocoles de cohérence locale. Le protocole abstrait de cohérence locale fait également ressortir l'ordonnancement entre ces phases. Cette factorisation structurelle permet d'exhiber la structure des protocoles de cohérence locale afin d'introduire des points d'adaptabilité (figure 7.3). Cette structure peut être manipulée et est modifiable selon les besoins. On obtient ainsi l'adaptabilité au niveau des interactions entre les composants constituant un protocole de cohérence locale (chapitre 4, section 4.1.3). Il est possible de mettre en œuvre chacune des phases indépendamment des autres (voir section 7.2). Afin d'adapter un protocole de cohérence locale à un nouveau contexte ou tout simplement le faire évoluer pour obtenir un nouveau protocole, il est possible de ne changer que la mise en œuvre de certaines phases, leur ordonnancement ou encore les liens entre les représentants de la même phase. Le contrôle sur la structure du protocole de cohérence locale peut ainsi se faire à différents niveaux.

La souplesse du canevas à supporter divers protocoles de cohérence locale est également assurée par une architecture spécifiant les composants fonctionnels pour construire de tels protocoles. Ces composants reprennent les points isolés dans le chapitre 2 en section 2.1. Ils peuvent être implantés de diverses façons selon l'objectif recherché et le contexte. Ils permettent aussi une meilleure articulation entre un service de duplication et d'autres services comme la tolérance aux fautes ou le contrôle de concurrence. On obtient ainsi l'adaptabilité des composants du canevas (chapitre 4, section 4.1.3). Ces composants servent à mettre en œuvre chacune des phases du protocole abstrait de cohérence locale. Ainsi, un représentant de phase est un ordonnanceur de ces composants. Il est possible de choisir les composants fonctionnels nécessaires, ainsi que la mise en œuvre adéquate. On peut, dans chacune des phases, remplacer un composant par un autre, en enlever ou en ajouter ou modifier les ordonnancements entre ces composants.

Les interfaces présentées ne couvrent que ce qui nous paraît important d'exhiber afin d'obtenir l'adaptabilité. Les composants fonctionnels présentés peuvent être mis en œuvre de façon répartie, cependant nous ne montrons pas les interfaces necéssaires à la définition de ces composants. De plus, nous n'avons pas présenté les interfaces liées au fonctionnement interne

des composants. Par exemple, le premier composant fonctionnel présenté, le gestionnaire de distribution, à besoin de connaitre les copies ajoutées ou supprimées. De même, un gestionnaire de rôle mettant en œuvre un protocole maître/esclaves peut avoir besoin d'élire un nouveau maître. Ainsi, à chaque composant, structurel ou fonctionnel, il faut rajouter une interface de fonctionnement permettant de faire transiter la méta information (ajout/suppresion de copies, copie défaillante, election, etc.) nécessaire à leur fonctionnement.

De plus, les interfaces peuvent paraitre élémentaires. Notre objectif est de montrer qu'avec ces deux factorisations nous pouvons obtenir l'adaptabilité dans tout ou partie des protocoles de cohérence locale. Avec l'approche par composants de RS2.7, les protocoles de duplication peuvent être construits par assemblage de composants et les protocoles existants peuvent être construits incrémentalement en ajoutant de nouveaux composants, structurels ou fonctionnels, aux assemblages existants. Cela permet à notre canevas de s'adapter à diverses situations très différentes les unes des autres. Il est possible également de réutiliser des composants pour mettre en œuvre des protocoles différents. Dans le chapitre suivant (chapitre 8) nous présentons une mise en œuvre de RS2.7 et nos éléments de validation.

Celui qui a besoin d'un protocole n'ira jamais loin ; les génies lisent peu, pratiquent beaucoup et se font d'eux-mêmes.
Denis Diderot - Le neveu de Rameau

Troisième partie

Mise en œuvre et validation

Chapitre 8

Mise en œuvre et validation de RS2.7

Sommaire

ans la partie "éléments de solution" nous avons défini l'aspect duplication et montré comment lui donner la propriété d'adaptabilité. Dans ce chapitre, nous présentons une mise œuvre, ainsi qu'une validation de notre approche en montrant comment à partir de RS2.7 il est possible d'obtenir différents services de duplication appropriés à différents contextes non fonctionnels.

Dans une première section (section 8.1), nous montrons la mise en œuvre choisie. Elle repose sur la construction de chaînes de liaison mettant en œuvre le protocole de cohérence locale. A partir de cela, une expérimentation est décrite et propose de valider notre approche. Cette validation montre dans un premier temps la capacité d'adaptation de tout ou partie des protocoles de cohérence locale définis à partir du canevas (section 8.2). Dans un second temps, le canevas est utilisé dans des contextes non fonctionnels différents, montrant ainsi que les mêmes principes d'architecture de RS2.7 restent pertinents (section 8.3). Nous terminons ce chapitre par une conclusion (section 8.4).

8.1 Mise en œuvre de RS2.7

Cette section commence par présenter les principes architecturaux des services de duplication (section 8.1.1), puis nous poursuivons par la présentation de la construction de services

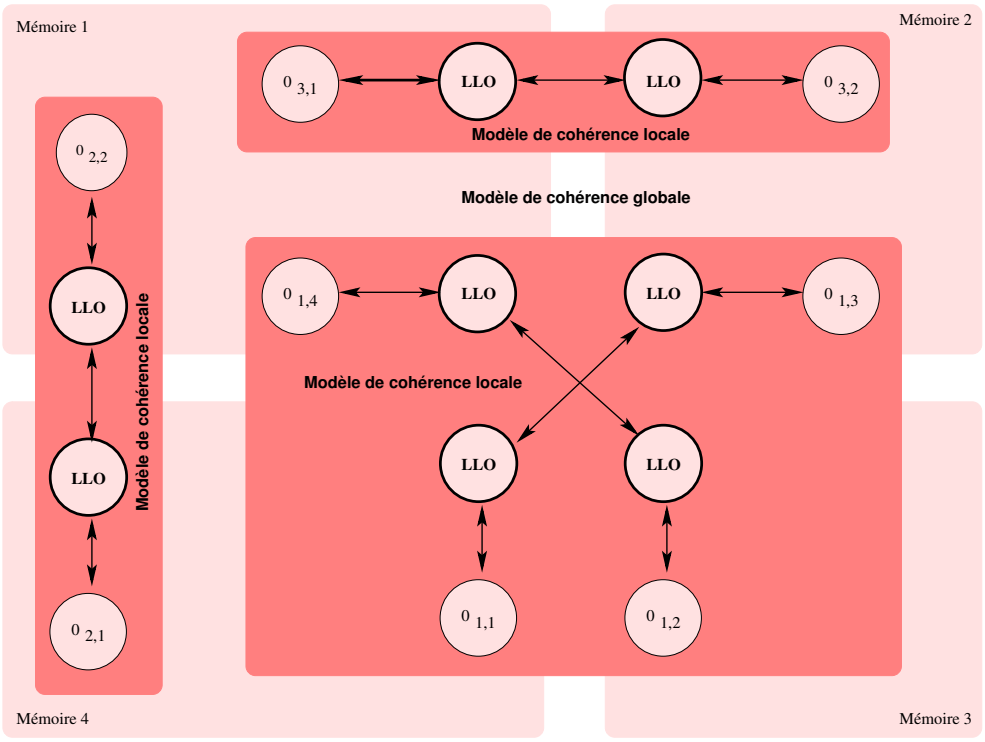


FIG. 8.1 – Chaînes de liaison entre copies

de duplication particuliers à partir de RS2.7 (section 8.1.2). Nous terminons cette section (section 8.1.3) en présentant comment l'utilisateur d'un service de duplication peut lier ses objets applicatif à dupliquer avec un service de duplication. La mise en œuvre a été faite en Java et Jonathan est utilisé pour les communications entre copies.

8.1.1 Principes d'architecture

Les principes architecturaux suivis par RS2.7 reprennent ceux utilisés dans Jonathan [Obj03]. Ces principes sont également repris dans JORM [BDD+00] (a Java Object Repository Mapping System) projet auquel nous avons participé. Nous interposons un objet de médiation (utilisation du patron "proxy") qui permet d'insérer de façon transparente pour l'application les actions mettant en œuvre le protocole de cohérence locale. Pour cela, ces objets de médiation sont connectés à un objet de liaison qui maintient les liens entre les différentes copies (utilisation du patron de nommage "export/bind"). Ainsi, une chaîne de liaison permet de lier les différentes copies d'un objet logique et d'y associer un certain protocole de cohérence locale (figure 8.1).

```
(1) Binding b1 = bf.createBinding();
(2) b1.setReplica(p1);
(3) Name np = binder.export(b1);
```

FIG. 8.2 – Code pour la création d'un objet dupliquable

Le protocole de cohérence locale qui gère un objet dupliqué est réparti sur toutes les copies. A chaque copie est associé un représentant d'objet de liaison. Dans la figure 8.1, on peut voir les objets dupliqués $O_{1,k}$, $O_{2,k'}$ et $O_{3,k''}$ dont les copies sont réparties sur les mémoires 1, 2, 3 et 4. Les objets *LLO* (*Local Logical Object*) sont des représentants d'objet de liaison. Ils permettent de lier les différentes copies d'un objet logique ($O_{1,k}$, $O_{2,k'}$ et $O_{3,k''}$). Ces objets mettent également en œuvre un certain protocole de cohérence locale. Ainsi, la construction d'un protocole particulier consiste à assembler les composants fonctionnels appropriés (chapitre 7, section 7.2) pour former des représentants de phase du protocole de cohérence locale abstrait (chapitre 7, section 7.1). Ces représentants de phase peuvent alors être intégrés à un représentant d'ordonnanceur qui met en œuvre un algorithme d'ordonnancement spécifique. Ces différents composants forment alors un LLO.

Construction des chaînes de liaison. Les chaînes de liaison sont construites autour de trois abstractions : l'identificateur, le contexte de nommage et le gestionnaire de liaison. Un **identificateur** est une notion générique des noms qui désigne de manière unique un objet et son contexte de nommage. Un **contexte de nommage** fournit la création de nom. Il garantit que chaque nom contrôle de manière non ambigüe un objet. Un **gestionnaire de liaison** est un contexte de nommage qui pour un certain nom, est capable de créer un chemin d'accès jusqu'à l'objet désigné par ce nom.

Ces définitions offrent une vue générique et uniforme des liaisons et sépare clairement l'identification des noms de l'accès des objets. Dans un contexte de nommage `nc` donné, un nouveau nom pour un objet `O` est obtenu en invoquant la méthode `nc.export(O)`. Des chaînes de référence peuvent être créées en exportant le nom à d'autres contextes de nommage. La création d'un chemin d'accès à un objet `O` désigné par un identificateur `id` est fait en invoquant la méthode `id.bind()` qui retourne un représentant prêt à être utilisé pour communiquer avec `O`.

Création d'un objet duplicable. La figure 8.2 montre les étapes mises en œuvre afin de rendre un objet applicatif ("p1") duplicable. Les figures 8.3(a) et 8.3(b) en sont une représentation graphique. Les étapes nécessaires afin de rendre un objet applicatif dupliquable sont les suivantes :

- un représentant local de la chaîne de liaison est créé à partir de l'usine à liaison adéquate "bf" (figure 8.2 ligne 1),
- l'objet applicatif "p1" est associé à la chaîne de liaison (son représentant) qui va ainsi

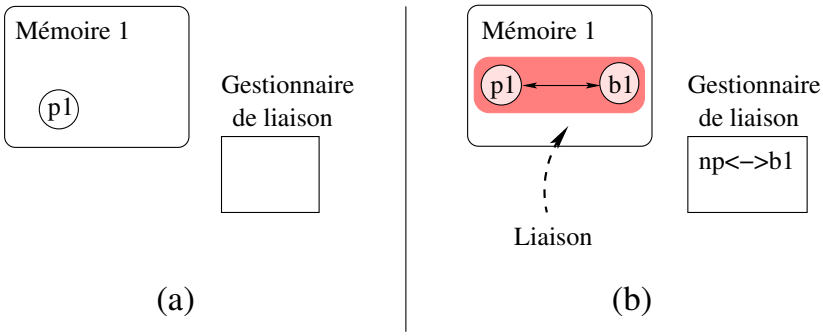


FIG. 8.3 – Création d'un objet dupliquable

```
(1) Binding b2 = bf.createBinding();
(2) b2.setReplica(p2);
(3) binder.bind(np,b2);
```

FIG. 8.4 – Code pour la création d'une copie

devenir la première copie (figure 8.2 ligne 2),
- la chaîne de liaison est exportée dans un domaine d'objets dupliqués caractérisé par le gestionnaire de liaison "binder" (figure 8.2 ligne 3). Ce gestionnaire de liaison est un contexte de nommage particulier qui va associer un nouveau nom "np" identifiant ainsi le nouvel objet dupliqué dans ce contexte. Ce gestionnaire de liaison garantit alors l'unicité de cette association entre le nouveau nom et cette chaîne de liaison.

Création d'une copie. Une fois qu'un objet est dupliquable, autant de copies que nécessaire peuvent venir s'enregistrer auprès de cet objet. Il leur suffit pour cela d'effectuer les étapes suivantes (figure 8.4 et figure 8.5 pour la représentation graphique) :

- un représentant local de la chaîne de liaison est créé à partir de l'usine à liaison adéquate "bf" (figure 8.4 ligne 1),
- la nouvelle copie, "p2", est associée à la chaîne de liaison (son représentant) qui va alors devenir la seconde (ou nième) copie (figure 8.4 ligne 2),
- la nouvelle copie se lie à l'objet dupliqué en utilisant le gestionnaire de liaison adéquate "binder". Le nom de l'objet dupliqué, "np", doit être indiqué (figure 8.4 ligne 3). A partir de ce moment, la copie "p2" est soumise au protocole de cohérence locale associé à cet objet dupliqué.

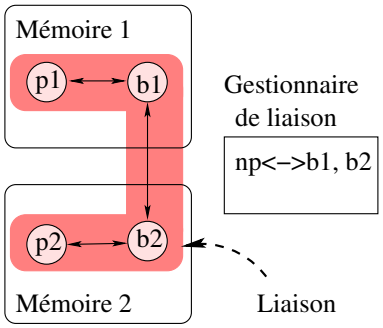


FIG. 8.5 – Création d'une copie

8.1.2 Construction d'un protocole de cohérence locale

RS2.7 isole un certain nombre de composants constituant un protocole de cohérence locale. Pour chaque composant, nous avons défini son rôle : les propriétés attendues, l'ensembles des ports fournis (les interfaces) et l'ensembles des composants avec lesquels il intéragit. La construction d'un protocole particulier consiste à ordonnancer les appels entre ces différents composants.

Ainsi, à chaque composant se trouve associé un contrôleur décidant de l'ordonnancement des appels entre les composants. Un contrôleur est lui même un composant :

- On trouve tout d'abord les composants permettant de mettre en œuvre l'architecture fonctionnelle (chapitre 7, section 7.2). Ces composants mettent en œuvre une fonctionnalité particulière.
- Les représentants de phase (chapitre 7, section 7.1) sont également des composants. Ils jouent le rôle de contrôleur des composants de l'architecture fonctionnelle. Ils ordonnancent les appels à ces composants fonctionnels qui sont donc des contrôlés.
- Les représentants d'ordonnanceur (chapitre 7, section 7.1) sont également des composants. Ils sont les contrôleurs des représentants de phase. Ces derniers jouent donc également le rôle de contrôlés.
- Les interactions avec le protocole de cohérence globale (chapitre 6, section 6.3) sont mis en œuvre sous forme de composant. Ils intéragissent avec les représentants d'ordonnanceur.
- Et enfin, les interactions avec les autres aspects non fonctionnels au niveau local (chapitre 5, section 5.3) sont également des composants. Ils intéragissent avec les représentants de phase.

RS2.7, ne définit que le rôle des composants (les propriétés attendues) et non la mise en œuvre aussi bien pour les contrôlés que pour les contrôleurs. Cette mise en œuvre dépend du protocole que l'on désire obtenir.

```
<!ELEMENT RSDef (Schema?,Import*,(Interface|Class))>
<!ELEMENT Schema (#PCDATA)>
 <!ELEMENT Import (#PCDATA)>
<!ELEMENT Interface (Inherit*,Attribute*,Method*) >
<!ATTLIST Interface
          Name CDATA #REQUIRED >
<!ELEMENT Class (Inherit?,Implement*,Attribute*,Method*) >
<!ATTLIST Class
          Name CDATA #REQUIRED >
<!ELEMENT Inherit (#PCDATA)>
<!ELEMENT Implement (#PCDATA)>
<!ELEMENT Method (Argument*)>
<!ATTLIST Method
          Name CDATA #REQUIRED
          ReturnType CDATA #IMPLIED
          Local CDATA #IMPLIED>
<!ELEMENT Argument EMPTY>
<!ATTLIST Argument
          Name CDATA #REQUIRED
          Type CDATA #REQUIRED
          GenClass CDATA #IMPLIED>
<!ELEMENT Attribute EMPTY>
<!ATTLIST Attribute
          Name CDATA #REQUIRED
                  Type CDATA #REQUIRED
                  GenClass CDATA #IMPLIED>
```

FIG. 8.6 – DTD XML de description des objets applicatifs

8.1.3 Liaison avec l'application

Afin que le protocole de duplication puisse accéder aux objets applicatifs, il doit connaitre la structure de ces objets. Pour cela, le développeur d'application décrit les objets qu'il veut dupliquer dans un fichier XML. La figure 8.6 présente la DTD à suivre afin de décrire ces objets applicatifs.

Un objet applicatif est défini dans un schéma, il peut importer un ensemble de définitions d'objets et être de type interface ou classe. Une interface ou une classe est définie par un nom et un ensemble d'attributs et de méthodes. Une classe peut hériter d'une autre classe et implanter plusieurs interfaces. Une interface peut hériter d'un ensemble d'interfaces. Une méthode se définit par son nom, le type de retour et un ensemble d'arguments. Un argument et un attribut se définissent par un nom et un type. De plus, ceux-ci peuvent être des tableaux ou des listes (attribut `GenClass` dans la DTD de la figure 8.6).

```
//*****************************
// Fichier genere par rsc
//*****************************

package rs27.tests.appli3;

import rs27.core.implementation.FieldImplem;
import rs27.core.implementation.*;
import rs27.core.Field;
import rs27.core.Value;
import rs27.components.kernel.ep.accessreplicamanager.AccessReplicaManager;

public class AccessReplicaManagerPersonne implements AccessReplicaManager {
    private Personne ao;

    public AccessReplicaManagerPersonne(Object ao){
        this.ao = (Personne)ao;
    }

    public void write(Field field, Value value){
        if (field.getName().equals("Nom")) {
            ao.setNom(((ValueString)value).getValue());}
        else {
            if (field.getName().equals("Prenom")) {
                ao.setPrenom(((ValueString)value).getValue());}
            else {
                if (field.getName().equals("Age")) {
                    ao.setAge(((ValueInteger)value).getValue());}
            }}
    }

    public Value read(Field field){
        Value value = null;
        if (field.getName().equals("Nom")) {
            value = new ValueString(ao.getNom());
        }
        else {
            if (field.getName().equals("Prenom")) {
                value = new ValueString(ao.getPrenom());
            }
            else {
                if (field.getName().equals("Age")) {
                    value = new ValueInteger(ao.getAge());
                }}}
        return value;}
}
```

FIG. 8.7 – Objet pour accèder à l'objet applicatif Personne

A partir des fichiers de description des objets applicatifs, le compilateur rsc génére les objets permettant au protocole de cohérence locale d'accéder aux objets applicatifs. La figure 8.7 présente les méthodes d'écriture et de lecture sur un objet applicatif de type `Personne`. Cet objet `AccessReplicaManagerPersonne` est généré par le compilateur rsc.

8.2 Adaptabilité des protocoles de cohérence locale

Pour valider l'adaptabilité proposée dans tout ou partie des protocoles de cohérence locale (chapitre 7), plusieurs protocoles ont été développés. Ils mettent en œuvre les quatre types de modèle de cohérence locale présentés au chapitre 5.

Dans ce chapitre, nous voulons montrer que les composants isolés peuvent être réutilisés et qu'il est possible d'adapter un protocole en ne modifiant qu'une partie de celui-ci afin d'obtenir le protocole adéquat.

Dans cette section, nous avons choisi de présenter la mise en œuvre de deux protocoles : un protocole maître/esclaves paresseux (section 8.2.1) mettant en œuvre le modèle de cohérence à copies convergentes avec lecture sur les copies divergentes et un protocole ROWA (section 8.2.2) mettant en œuvre le modèle de cohérence locale à copie unique atomique.

8.2.1 Un protocole de cohérence locale maître/esclaves paresseux

Considérons un protocole maître/esclaves paresseux assez simple. Les écritures sont permises seulement sur la copie maîtresse et les lectures peuvent être effectuées sur toutes les copies. Le maître diffuse les mises à jour de façon asynchrone à toutes les copies esclaves. Ce protocole met en œuvre le modèle à copies convergentes avec lectures sur les copies divergentes.

8.2.1.1 Mise en œuvre

Afin de mettre en œuvre ce protocole, nous définissons les représentants d'ordonnanceur, les représentants de phase et les composants fonctionnels.

Les représentants d'ordonnanceur. Les représentants d'ordonnanceur du maître et des esclaves sont différents. Sur le maître, le représentant d'ordonnanceur met en œuvre un protocole du type :

- (ACER) suite à une écriture. L'écriture est traitée de manière séquentielle par la phase d'accès, la phase de coordination, la phase d'exécution et enfin par la phase de réponse.
- (AER) suite à une lecture. La lecture est traitée par la phase d'accès, puis la phase d'exécution et enfin par la phase de réponse.

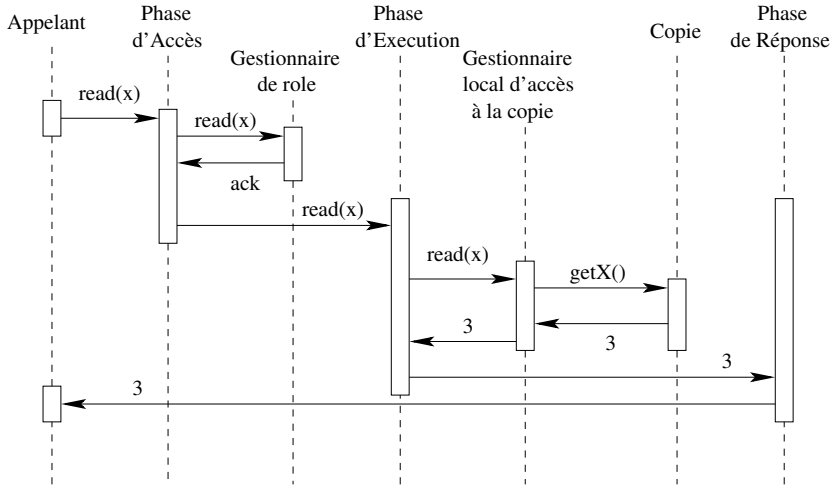


FIG. 8.8 – Lecture sur le maître ou sur les esclaves

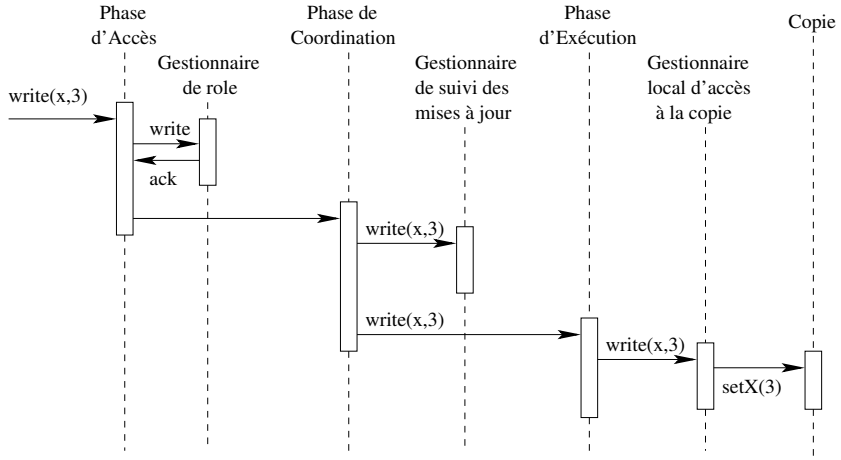


FIG. 8.9 – Ecriture sur le maître

– (CE) lors d'une synchronisation. En effet, de manière asynchrone la phase de coordination peut se déclencher, suivi d'une phase d'exécution.

Sur les esclaves, les représentants d'ordonnanceur mettent en œuvre un protocole du type :

– (AER) suite à une lecture. La lecture est traitée par la phase d'accès, puis la phase d'exécution et enfin par la phase de réponse.
– (CE) suite à une synchronisation. En effet, le représentant de la phase de coordination du maître peut déclencher le représentant de la phase de coordination des esclaves. La phase de coordination des esclaves est suivie d'une phase d'exécution.
– les écritures ne sont pas permises.

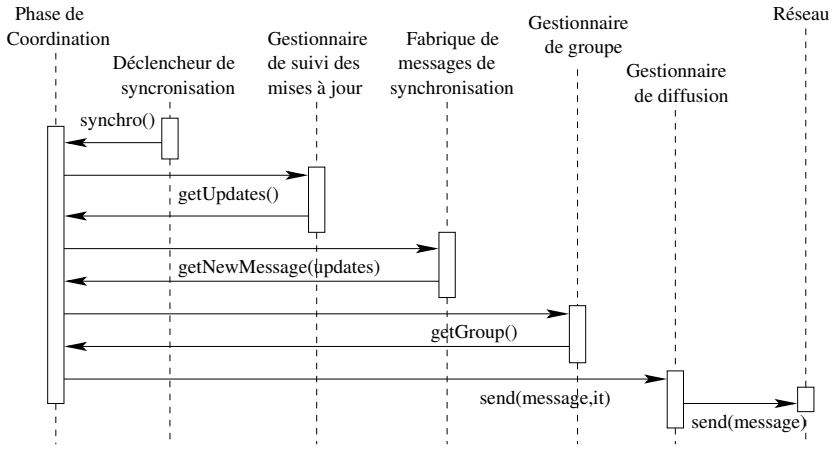


FIG. 8.10 – Processus de synchronisation sur le maître

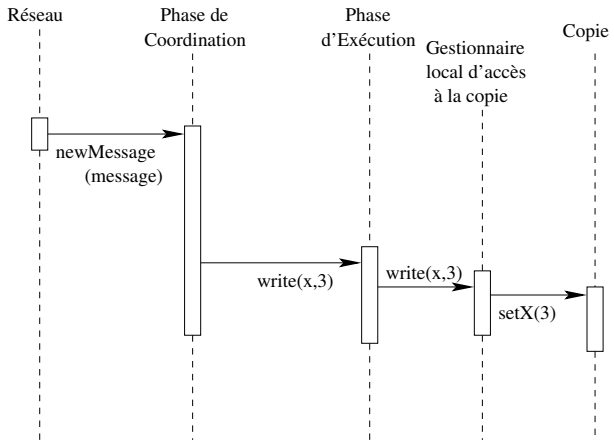


FIG. 8.11 – Processus de synchronisation sur un esclave

Les représentants de phase. On retrouve les mêmes phases dans tous les représentants de liaison : la phase d'accès, la phase de coordination, la phase d'exécution et la phase de réponse.

La phase d'accès est constituée du *gestionnnaire de rôle* décidant si le représentant de liaison a le droit de traiter une requête externe. La phase d'exécution est constituée d'un *gestionnaire local d'accès à la copie* permettant d'accéder à la copie. La phase de réponse se réduit également à un seul composant, le *gestionnaire de réponse*.

La phase de coordination est différente sur le maître et sur les esclaves. Le maître se charge de décider du moment de synchronisation, du suivi des mises à jour et de la construction des messages de synchronisation. Ainsi, sa phase de coordination est constituée d'un *déclencheur*

de synchronisation, d'un *gestionnaire de suivi des mises à jour* et d'une *fabrique de messages de synchronisation*. Les esclaves se contentent de recevoir les messages de synchronisation provenant du maître. La phase de coordination ne contient donc pas de composants fonctionnels.

Les composants fonctionnels. Pour la mise en œuvre des composants fonctionnels certains choix sont pris :

- Le gestionnaire de rôles du maître et celui des esclaves sont triviaux. Pour le maître les requêtes externes en écriture sont autorisées et pour les esclaves elles sont refusées. Les défaillances potentielles des copies ne sont pas prises en compte. Ainsi, il n'existe pas de processus d'élection d'un nouveau maître. De plus, le maître est statique.
- Le gestionnaire local d'accès à la copie est généré à partir de rsc (section 8.1.3).
- Le déclencheur de synchronisation demande la mise en cohérence des esclaves lorsque quatre écritures sont faites sur le maître depuis la dernière synchronisation.
- Le gestionnaire de suivi des mises à jour marque les champs modifiés sur l'objet.
- La fabrique des messages de synchronisation construit un message de synchronisation contenant l'état des champs modifiés depuis la dernière synchronisation et qui doivent être répercutés sur les copies.

Interactions entre les composants. Les diagrammes de séquence des figures 8.9, 8.8, 8.10 et 8.11 montrent le processus mis en œuvre lors de l'exécution d'opérations d'écriture ou de lecture. Nous avons omis les représentants de la phase de réponse et ceux de l'ordonnanceur pour simplifier la présentation :

- Lors d'une opération de lecture sur le maître ou l'esclave (figure 8.8), une phase d'accès est mise en œuvre au cours de laquelle le *gestionnaire de rôle* autorise l'opération. La phase d'exécution qui s'ensuit utilise le *gestionnaire local d'accès à la copie*. Enfin, le *gestionnaire de réponse* retourne le résultat à l'appelant au cours de la phase du même nom. Les représentants d'ordonnanceur appellent donc la phase d'accès, la phase d'exécution et la phase de réponse (protocole de type (AER)).
- Lors d'une opération d'écriture, un esclave exécute une phase d'accès au cours de laquelle le *gestionnaire de rôle* refuse la lecture, sachant qu'un message d'erreur est ensuite généré en *phase de réponse*.
- Du côté du maître, la même opération d'écriture déroule les phases d'accès puis de coordination (figure 8.9). L'opération est enregistrée dans le *gestionnaire de suivi des mises à jour*, avant d'exécuter l'accès lui-même à travers le *gestionnaire local d'accès à la copie*. Une réponse est ensuite renvoyée à l'appelant. Le représentant d'ordonnanceur appelle donc la phase d'accès, la phase de coordination, la phase d'exécution et la phase de réponse (protocole de type (ACER)).
- Le *déclencheur de synchronisation* (figure 8.10) active le processus de synchronisation de façon asynchrone lorsque quatre écritures sont faites sur le maître sans synchronisation des esclaves. La phase de coordination construit alors le message idoine en utilisant la *fabrique de messages de synchronisation* ainsi que le *gestionnaire de suivi des mises à jour*. Le message est alors envoyé à toutes les copies. Chaque esclave reçoit le message

et met à jour sa copie associée au cours de la phase d'exécution (figure 8.11). Les représentants d'ordonnanceur appellent donc la phase de coordination puis la phase d'exécution (protocole de type (ACER) sur le maître et sur les esclaves).

8.2.1.2 Adaptation du protocole

Afin d'adapter le protocole maître/esclaves paresseux selon les besoins, il est possible de modifier un (des) composant(s) fonctionnel(s), un (des) représentant(s) de phase ou un (des) représentant(s) d'ordonnanceur.

Changement d'un composant fonctionnel. Le premier moyen pour adapter le canevas est lié au remplacement de la mise en œuvre d'un composant fonctionnel par une autre.

Le *gestionnaire de suivi des mises à jour* peut être mis en œuvre par un journal physique, un fichier ou en accédant tout simplement à l'état de la copie maîtresse (en utilisant la phase d'exécution qui utilise le *gestionnaire local d'accès à la copie*). Le *gestionnaire local d'accès à la copie* peut également être remplacé si la nature de l'objet à dupliquer change (par exemple la copie peut être un objet Java, une page HTML, etc).

Il est également possible de remplacer le *gestionnaire de rôle* par une mise en œuvre permettant d'obtenir un maître dynamique. Parmi les politiques possibles, citons une politique d'élection entre copies ou une politique de type "token ring". Cependant dans ce cas, il est nécessaire de modifier la phase de coordination des esclaves ; ceux-ci en devenant maître doivent être capables de construire des messages de synchronisation.

Changement d'un représentant de phase. Ainsi, il est possible d'adapter un protocole en modifiant une phase (ajouter, retirer des composants, ou encore changer leur composition). A partir du protocole maître esclaves paresseux présenté il est possible de modifier certaines phases afin d'obtenir de nouveaux protocoles :

- Pour prendre en compte un protocole avec maître dynamique, il est possible de donner aux esclaves la même phase de coordination que celle du maître présentée ci-dessus. Cependant, afin d'assurer le modèle de cohérence locale à copies convergentes avec écriture sur les copies divergentes, il faut s'assurer que chaque nouveau maître a bien reçu les écritures du maître précédent.
- Afin d'envoyer les mises à jour aux différents esclaves le plus vite possible (protocole de type *as soon as possible*), le *déclencheur de synchronisation* peut être désactivé, le processus de synchronisation étant démarré lors d'une opération d'écriture sur le maître et non toutes les quatre écritures.
- Afin d'éviter qu'un esclave renvoie une exception lorsqu'on lui soumet une écriture on peut rajouter un *gestionnaire de redistribution* dans la phase d'accès de chacun des esclaves. Ainsi, un esclave, reçevant une requête externe demandant une écriture, la renvoie au maître.

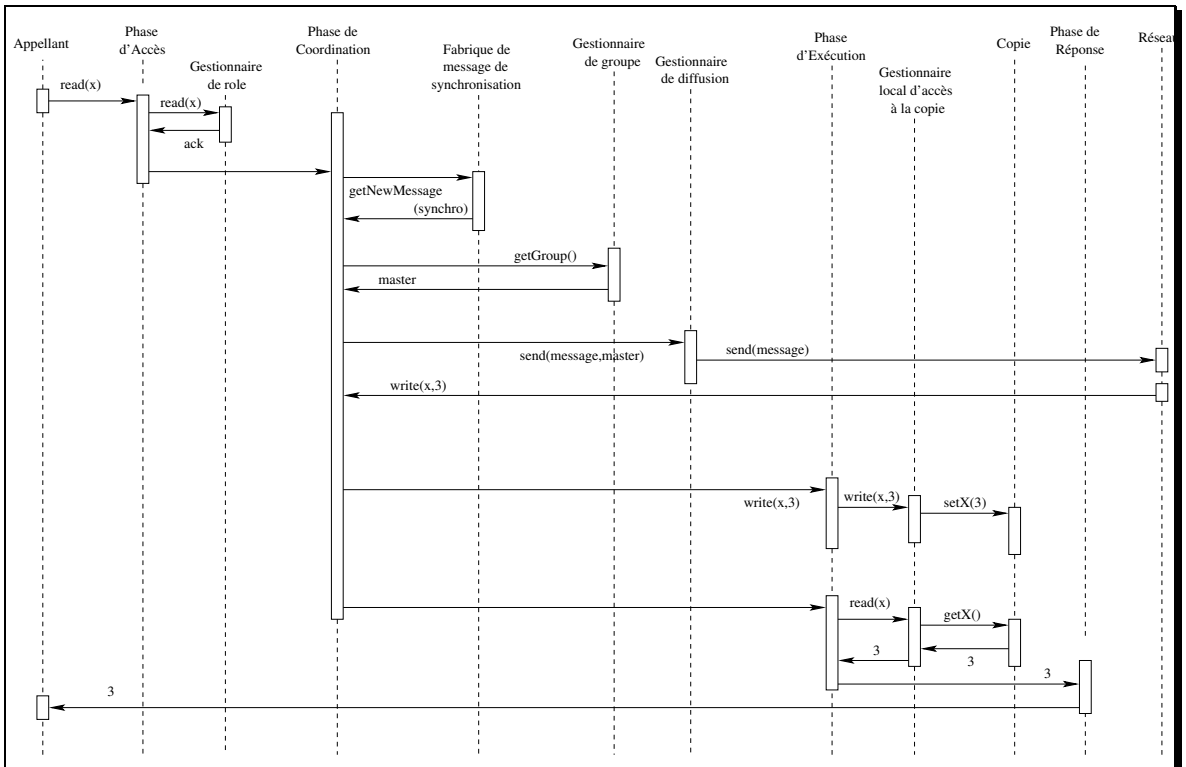


FIG. 8.12 – Lecture sur un esclave dans un protocole paresseux maître/eslaves de type demande/diffusion

Changement d'un représentant d'ordonnanceur. Il est également possible de modifier un représentant d'ordonnanceur afin d'inclure une nouvelle phase, de supprimer une phase ou de modifier l'ordonnancement des appels entre les phases.

Supposons que l'on souhaite obtenir un protocole paresseux maître/esclaves de type demande/diffusion. Pour cela, on peut transformer le protocole d'ordonnancement des représentants d'ordonnanceur des esclaves en protocole de type (ACER) lorsque ceux-ci reçoivent une requête de lecture. Dans ce cas, la copie locale n'est pas directement lue (phase d'exécution). Une phase de coordination est d'abord appelée. Lors de celle-ci une mise à jour est demandée au maître. Pour ce processus de synchronisation le représentant d'ordonnanceur met en œuvre un protocole de type (CE). Le représentant de la phase de coordination d'un esclave déclenche celle du maitre qui renvoie la mise à jour à faire à l'esclave. Celui-ci l'installe lors d'une phase d'exécution. La figure 8.12 illustre une lecture sur un esclave avec ce nouveau protocole. La mise à jour envoyée par le maître est `write(x,3)`.

Le premier protocole mettait en œuvre une politique de synchronisation par diffusion, alors que dans le cas présent, la politique utilisée est de type demande/diffusion : les esclaves demandent les modifications avant les lectures. Le maître diffuse toujours suivant la politique

du *déclencheur de synchronisation*. Si la phase de coordination du maître n'est pas modifiée, il envoie les modifications à toutes les copies lorsque l'une d'elles cherche à effectuer une lecture. On peut modifier la phase de coordination du maître afin d'envoyer les mises à jour uniquement à la copie esclave qui cherche à lire.

8.2.2 Le protocole de cohérence locale ROWA

Nous considérons maintenant un protocole de type ROWA (chapitre 2, section 2.3.2.1) mettant en œuvre le modèle de cohérence locale atomique. Dans un protocole de ce type une lecture est faite sur une copie et une écriture sur l'ensemble des copies de manière atomique et synchrone.

8.2.2.1 Mise en œuvre

Afin de mettre en œuvre ce protocole, nous définissons les représentants d'ordonnanceur, les représentants de phase et les composants fonctionnels.

Les représentants d'ordonnanceur. Les représentants d'ordonnanceur de toutes les copies sont identiques. Ils mettent en œuvre un protocole du type (AER) suite à une lecture. La requête est traitée de manière séquentielle par la phase d'accès, la phase d'exécution et enfin par la phase de réponse. Dans le cas d'une opération d'écriture le protocole est de type (ACER) : la requête est traitée par la phase d'accès, la phase de coordination, la phase d'exécution et enfin par la phase de réponse.

Les représentants de phase. Lors d'une écriture, le représentant soumet la requête à la phase d'accès. Celle-ci renvoie l'opération au représentant de l'ordonnanceur. L'opération est ensuite traitée par la phase de coordination. Le représentant de la phase de coordination envoie la requête en écriture à tous les représentants de la phase de coordination après avoir invoqué le service de contrôle de la concurrence. Lors de la phase d'exécution la mise à jour est appliquée sur la copie locale.

Lors d'une lecture, le représentant de la phase de coordination invoque également le service de contrôle de concurrence avant de transférer la requête au représentant de la phase d'exécution. La lecture est alors faite sur la copie locale.

Les composants fonctionnels. On trouve la fabrique de messages de synchronisation pour les représentants de la phase de coordination, le gestionnaire local d'accès à la copie pour les représentants de la phase d'exécution et le gestionnaire de réponse pour la phase de réponse. Le gestionnaire local d'accès à la copie est généré par le compilateur rsc (section 8.1.3).

Le contrôleur de concurrence Le contrôleur de concurrence assure qu'une opération d'écriture est en accès exclusif sur l'objet logique.

8.2.2.2 Adaptation du protocole

Cette section présente comment obtenir un protocole ROWAA (chapitre 2, section 2.3.2.1) à partir du protocole ROWA. Ce protocole met également en œuvre un modèle de cohérence locale à copie unique. Nous présentons également comment obtenir un protocole ROWAA mettant en œuvre un modèle de cohérence locale à copies convergentes avec lecture sur les copies divergentes.

De ROWA à ROWAA. Pour transformer le protocole ROWA en protocole ROWAA, il nous faut introduire :

- (1) une phase de validation composée d'un gestionnaire de copies défaillantes,
- (2) rajouter un gestionnaire de groupe de synchronisation dans la phase de coordination et
- (3) faire intéragir la phase de validation avec la phase de coordination.

Les représentants d'ordonnanceur sont donc du type (A(CEV)*R).

Si le gestionnaire de copies défaillantes de la phase de validation détecte une ou des copies défaillantes, il informe le représentant de l'ordonnanceur qui va alors déclencher de nouveau la phase de coordination. Il va également donner au gestionnaire du groupe de synchronisation de la phase de coordination la ou les copies défaillantes.

Changement de modèle de cohérence locale. Considérons maintenant un protocole ROWAA mettant en œuvre le protocole de cohérence locale à copies convergentes avec lectures sur les copies divergentes.

Afin de mettre en œuvre ce protocole, nous pouvons reprendre le protocole ROWAA présenté ci-dessus. Dans ce cas, le contrôleur de concurrence n'est plus nécessaire. De plus, il faut rajouter un gestionnaire de rôle dans la phase d'accès. En fait, il est possible de reprendre la phase d'accès du protocole maître/esclaves paresseux présentée en section 8.2.1. Les représentants d'ordonnanceur et les représentants des autres phases restent identiques à ceux présentés initialement pour le protocole ROWAA.

8.3 Adaptabilité au contexte non fonctionnel

Cette section se décompose en deux parties. Nous présentons tout d'abord PING, projet dans lequel nous avons utilisé RS2.7. Nous avons développé un prototype Java de RS2.7 dans

ce contexte et nous avons utilisé sa capacité d'adaptabilité au contexte non fonctionnel (section 8.3.1). Nous présentons ensuite quelques discussions portant sur l'adaptabilité au contexte non fonctionnel de RS2.7 (section 8.3.2).

8.3.1 Utilisation de RS2.7 dans le contexte des plateformes pour mondes virtuels

Une utilisation du canevas a été faite dans le cadre du projet européen PING (Platform for Interactive Networked Games) [CDD+00, CDD+01].

Principal objectif du projet PING

Ce projet visait à specifier, développer et démontrer une architecture souple et extensible pour des applications Internet interactives et massivement multi-participants. Ces applications Internet sont des jeux multi-joueurs se déroulant dans des mondes virtuels. Afin de permettre le passage à l'echelle, chaque joueur participant au monde virtuel possède sur sa machine une copie de la partie du monde où il se trouve.

La principale difficulté que visait à resoudre ce projet était de fournir une vue cohérente du monde virtuel aux utilisateurs. Malheureusement, dans un système réparti avec un grand nombre de sites il est difficile d'obtenir cette cohérence du monde en temps réel tout en offrant le passage à l'echelle. Ainsi, il est nécessaire de relâcher la cohérence entre les objets et entre les copies d'un même objet. Cependant, afin de préserver la sémantique et la cohérence des jeux il n'est pas possible de relâcher la cohérence de la même façon pour tous les objets composant le monde virtuel. Par exemple, les avatars des utilisateurs (les representants des utilisateurs) doivent refléter le même état et la même position et cela pour tous les utilisateurs partout dans le monde. Par contre, il est acceptable de dégrader la cohérence entre les objets du décors et entre leurs copies ou bien encore entre les objets représentant des personnages non cruciaux dans le déroulement du jeu. Le contexte de ce projet n'est pas transactionnel.

Mise en œuvre de RS2.7 pour PING

Notre expérimentation s'est focalisée sur la séparation de la cohérence locale et de la cohérence globale tels que nous l'avons décrite au chapitre 6. Plusieurs protocoles de cohérence locale ont été implantés pour les modèles suivants :

- Modèle à copies convergentes avec lecture sur les copies divergentes,
- Modèle à copies convergentes avec écriture sur les copies divergentes et
- Modèle à copies divergentes.

Ces protocoles de cohérence locale sont utilisés pour implanter plusieurs modèles de cohérence globale. Ces modèles de cohérence globale sont des variantes des modèles de cohérence globale séquentiel et causale prenant en compte les problèmes liés au temps réél [CDD+00].

Afin d'obtenir de bonnes performances, nous avons optimisé la composition des différents composants constituant un protocole de cohérence locale (composants fonctionnels, représentants de phase et d'ordonnanceur). Pour cela nous avons fusionné manuellement les différents composants constituant un représentant de liaison afin de n'avoir qu'un seul objet comme représentant d'objet de liaison. En effet, l'objectif de ce projet étant d'obtenir une plateforme supportant de nombreux joueurs, chaque joueur devait possèder une copie de tout ou d'une partie du monde. Ceci implique un représentant d'objet de liaison pour chaque copie d'objet qu'il possède. Il devient contraignant de garder la décomposition à l'intérieur de ces représentants lorsque le nombre d'objets dupliqués est important. De plus, cette décomposition nous permet d'obtenir l'adaptabilité dans tout ou partie des protocoles de cohérence locale pour faciliter la construction des protocoles et/ou pour modifier un protocole à l'exécution. Nous avons profité de l'adaptabilité lors de la conception des protocoles en réutilisant certains composants dans des protocoles implantant les différents modèles. Mais lors de l'exécution, cette adaptabilité ne nous servait plus. Le but du projet PING n'était pas de pouvoir modifier un protocole lors de l'exécution. Ainsi, une fois le choix fait des différents composants constituant les représentants de liaison, nous avons fusionné manuellement ces composants et optimisé la composition.

Tout cela ne remet pas en cause les principes proposés par RS2.7. Notre canevas est une analyse de l'aspect duplication. Suivant l'objectif recherché (l'adaptabilité au contexte non fonctionnel et/ou dans tout ou partie des protocoles de cohérence locale), le concepteur d'un service de duplication reprend ce qui l'intéresse.

Ainsi, dans le contexte du projet PING, nous avons gardé les interfaces externes du protocole de cohérence locale, l'interface avec l'application et les interfaces avec le niveau de cohérence globale, afin d'obtenir l'adaptabilité au contexte non fonctionnel.

Apports de RS2.7 pour PING

L'interêt de l'approche prise par RS2.7 a été double dans le contexte du projet PING. D'une part, la complexité de la construction de plateformes pour mondes virtuels par des programmeurs répartis géographiquement necessite de séparer le rôle de chacun. L'abstraction de l'aspect duplication, a facilité la définition de ces rôles. Les programmeurs du niveau global pouvaient clairement définir le service qu'ils attendaient de la part du protocole de cohérence locale et nous, nous pouvions nous consacrer à developper le protocole de cohérence locale adéquate suivant les objets et respectant le modèle de cohérence locale.

D'autre part, nous avons utilisé différents protocoles de cohérence locale implantant le même modèle de cohérence locale. L'abstraction de l'aspect duplication était essentiel. En effet, dans le projet PING il était nécessaire d'avoir le même modèle de cohérence globale et le même modèle de cohérence locale pour un ensemble d'objets. Cependant, le protocole de cohérence locale devait être différent suivant les objets. Notamment, certaines informations de synchronisation des copies d'un même objet proviennent de la couche applicative et dépendent donc de la nature de l'objet.

Notre validation de l'adaptabilité au contexte non fonctionnel a consisté à démontrer qu'il est possible de séparer l'information d'ordonnancement des opérations entre les objets de celle d'ordonnancement des opérations sur les copies d'un même objet logique. De plus, nous avons voulu montrer qu'il est possible d'utiliser les mêmes protocoles de cohérence locale pour mettre en œuvre des protocoles de cohérence globale utilisés dans des domaines différents. Ce dernier point a été démontré dans le chapitre 6 par une présentation formelle.

8.3.2 Discussion autours de la construction de modèles de cohérence globale

Cette section présente briévement des mises en œuvre de modèles de cohérence globale et comment grâce à l'adaptabilité du protocole de cohérence locale au contexte non fonctionnel, nous pouvons facilement utiliser un même protocole de cohérence locale dans différents contextes non fonctionnels.

Afin de simplifier le discours nous n'introduisons pas de nouveaux protocoles de cohérence locale et globale propres aux mondes virtuels, nous reprenons les modèles de cohérence globale sans synchronisation et avec synchronisations des MVP présentés au chapitre 6. Nous commencons par les modèles de cohérence globale atomique (chapitre 2, section 2.4.2.1) et PRAM (chapitre 2, section 2.4.2.4). Ensuite, nous présentons comment les protocoles de cohérence locale utilisés pour mettre en œuvre ces deux modèles de cohérence globale peuvent être adaptés pour mettre en œuvre le modèle de cohérence au relâchement (chapitre 2, section 2.4.3.2).

Modèle de cohérence globale atomique : Afin d'assurer le bon ordonnancement des opérations entre les différents objets, le protocole de cohérence globale doit garantir que toute modification est immédiatement perçue par tous les processus et dans le même ordre. Pour cela, il s'appuie sur un gestionnaire de concurrence et sur un protocole de cohérence locale.

Le gestionnaire de contrôle de la concurrence assure que les accès en écriture sont exclusifs et que plusieurs accès en lecture sont permis. Le gestionnaire de contrôle de la concurrence n'a pas conscience de la duplication et ne s'occupe donc pas de la gestion de la concurrence entre deux copies. Cela est du ressort du protocole de cohérence locale.

Dans le chapitre 6 nous avons vu que le protocole de cohérence locale doit mettre en œuvre le modèle de cohérence locale à copie unique atomique. Le protocole de cohérence locale doit donc assurer l'atomicité entre n'importe quelle paire d'opérations de lecture et d'écriture, et n'importe quelle paire d'opérations d'écriture. Par contre, les paires d'opérations lecture/lecture peuvent être concurrentes. Nous reprenons le protocole présenté en section 2.4.2.1. Quand un processus veut lire une copie, le protocole de cohérence locale envoie une requête au maître de la copie. Le maître synchronise alors la copie devant être lue et annule son droit en écriture. Lorsqu'un processus veut écrire une copie, le protocole de cohérence locale envoie, cette opération au maître courant. Le maître invalide alors toutes les copies (sauf la sienne), et synchronise ensuite la copie demandeuse. La copie demandeuse est alors le nouveau maître, et aucun autre processus n'a accès aux autres copies.

Ainsi, le protocole de cohérence locale bloque l'accès aux copies en cas d'écriture et le protocole de cohérence globale fait de même en cas d'écriture simultanée de différents processus sur le même objet.

Modèle de cohérence globale PRAM : Pour implanter ce modèle, il faut que le protocole de cohérence locale assure que chaque copie perçoit les écritures faites sur chacune des autres copies dans l'ordre dans lequelle elles ont été faites sur chacune d'elles. Le protocole de cohérence globale doit quant à lui garantir que les opérations faites sur un processus particulier sont perçues dans le même ordre par les autres processus.

Une mise en œuvre possible de ce modèle de cohérence globale est la suivante. A chaque processus P_i est associé un tableau d'entiers ayant pour indices les objets O_j utilisés par ce processus. A chaque opération de P_i sur un objet O_j, l'entier du tableau correspondant à O_j est incrémenté.

Le protocole de cohérence locale met en œuvre le modèle de cohérence locale à copies divergentes FIFO. Il assigne à chaque opération sur une copie particulière une estampille composée du numéro de la copie et du numéro d'ordre de l'opération sur la copie. Le numéro d'ordre de l'opération est incrémenté à chaque nouvelle opération.

Lors d'une opération d'écriture par un processus P_i sur un objet O_j, le protocole de cohérence globale incrémente l'élément correspondant dans le tableau. Ensuite, l'opération est envoyée au protocole de cohérence locale. Celui-ci incrémente la valeur de l'estampille puis synchronise les différentes copies. Un message est envoyé contenant le tableau et la valeur de l'estampille donnée par le protocole de cohérence locale. Le protocole de cohérence globale des processus $P_{i'}$ reçoit ce message (voir chapitre 6, section 6.2.5). Il vérifie s'il a déjà reçu les opérations de syncronisation des autres objets en fonction des valeurs contenues dans le tableau de P_i. Si c'est le cas il envoie le message de synchronisation au protocole de cohérence locale. Celui-ci décide en fonction de la valeur de l'estampille s'il peut installer la mise à jour.

Discussion : Le point le plus intéressant à retenir est qu'un même modèle de cohérence globale peut être implanté par des protocoles de cohérence locale mettant en œuvre différents modèles de cohérence locale (si ces derniers gardent les relations définies dans le modèle de cohérence globale). Par exemple au chapitre 6, nous avons vu que le modèle de cohérence globale PRAM peut être implanté en utilisant un modèle de cohérence locale à copie unique atomique ou à copies divergentes FIFO. Ainsi pour un ensemble d'objets dupliqués devant respecter le modèle de cohérence globale PRAM, il est possible d'associer pour certains objets dupliqués un modèle de cohérence à copie unique atomique et pour les autres un protocole à copies divergentes FIFO. Il est possible de raisonner au niveau global et au niveau local.

Modèle de cohérence globale au relâchement. Dans ce paragraphe, nous présentons comment adapter les protocoles de cohérence locale présentés ci-dessus pour mettre en œuvre un modèle de cohérence globale avec synchronisations des MVP.

Une implantation simpliste de ce modèle peut être la suivante :

- Un protocole de cohérence locale met en œuvre le modèle de cohérence locale à copies divergentes FIFO entre les copies d'un même objet.
- Un protocole de cohérence globale met en œuvre le modèle de cohérence PRAM entre les objets.
- Un protocole de cohérence globale met en œuvre un modèle de cohérence PRAM sur les variables de synchronisation.
- Les opérations de lecture et d'écriture ne peuvent se faire que si toutes les opérations d'acquisition (acquire) précédentes sont terminées sur tous les sites ;
- L'opération de relâchement (release) ne peut se terminer que si toutes les opérations d'écriture et de lecture sont terminées sur tous les sites.

La partie concernant uniquement la duplication est gérée par le protocole de cohérence locale à copies divergentes FIFO. Tout le reste est à la charge du protocole de cohérence globale. Ainsi, il est possible de reprendre le protocole de cohérence locale FIFO présenté ci-dessus afin de gérer l'aspect duplication.

De même le protocole de cohérence globale reprend le protocole présenté précédemment. De plus, celui doit gérer les variables de synchronisation et garantir les propriétés des opérations `acquire` et `release`.

Si l'on garde le protocole de cohérence locale FIFO présenté précedemment, les copies seront synchronisées à chaque écriture. Afin de s'adapter à ce modèle de cohérence globale, il est plus intéressant de modifier lègérement le protocole de cohérence locale afin que les copies se synchronisent lors de l'appel à l'interface "de mise à jour des copies" par l'intermédiaire de la méthode `synchronize()` (chapitre 6, section 6.3.2).

8.4 Conclusion

Dans ce chapitre, nous nous sommes attachés à montrer la pertinence de notre approche. Notre validation porte sur la présentation de mises en œuvre et non sur l'étude de performances. Nous ne désirons pas montrer que notre approche est efficace. Au premier abord cela serait étonnant vu la grande décomposition des protocoles de cohérence locale et la séparation cohérence locale/globale. Au contraire, nous désirons montrer que grâce à la définition du canevas RS2.7, il est possible d'adapter les protocoles de cohérence locale.

L'intérêt de notre approche est double. D'une part, lors de la conception des protocoles de cohérence locale, il est possible de réutiliser les mêmes composants structurels et fonctionnels afin de mettre en œuvre différents protocoles de cohérence locale. Il est également possible de réutiliser les protocoles construits en rajoutant de nouveaux composants afin qu'ils s'adaptent à des contextes non fonctionnels particuliers pour lesquels ils n'avaient pas forcément été conçus. Cette approche facilite donc la construction de protocoles de cohérence locale adéquats.

D'autre part, dans une approche dynamique, il devient possible de faire évoluer un proto-

cole de cohérence locale, au moment de l'exécution, afin que celui s'adapte à un changement de l'environnement ou des besoins de l'application. Cela se révèle nécessaire pour certains contextes : informatique mobile, système à grande échelle.

L'expérimentation nous a montré qu'il est assez facile de construire de nouveaux protocoles ou d'adapter un protocole particulier. Cependant, cela demande une très bonne connaissance des principes proposés par le canevas. Son utilisation par des personnes ne connaissant pas parfaitement le canevas nous a montré que la construction de protocoles peut se révéler être une tâche longue et ardue. Cependant, nous voyons nos travaux comme le point de départ de travaux sur la composition d'aspects, travaux proposant des outils de composition. Ceux-ci pourrons exploiter le fait que nous "ouvrons" l'aspect duplication afin de construire des protocoles statiques et optimisés (par exemple par réécriture de code en réduisant le nombre d'objet) ou des protocoles dynamiques où il est possible d'agir sur la composition lors de l'exécution. Il serait intéressant également que ces outils permettent d'obtenir des protocoles faisant une composition statique et optimisée sur une partie du protocole et offrant la possibilité d'adapter dynamiquement les autres parties du protocole.

Nous remarquons également que des outils permettant une composition statique et optimisée des composants constituant les protocoles pourront se révèler être très utiles. De nombreuses applications n'ont pas besoin que le protocole de cohérence locale puisse s'adapter lors de l'exécution. La propriété d'adaptabilité est alors surtout utile lors de la conception d'un protocole afin d'obtenir un protocole parfaitement adapté aux besoins. Dans ce cas, il est impératif d'optimiser la composition. A titre d'exemple, la différence de la taille du code pour un protocole de cohérence locale avec mise à jour sur n'importe quel site assez simple est de l'ordre d'un facteur dix. Dans le cadre du projet PING un tel protocole représente 77 lignes, alors que dans le prototype avec décomposition à l'intérieur du protocole de cohérence locale le même protocole fait un peu plus de 800 lignes. Cela est principalement dû au nombre important d'objets qui composent alors le protocole.

J'ai tendu des cordes de clocher à clocher ; des guirlandes de fenêtre à fenêtre ; des chaînes d'or d'étoile à étoile, et je danse.
Arthur Rimbaud

Chapitre 9

Conclusion et perspectives de recherche

Sommaire

e chapitre dresse un bilan de cette thèse (section 9.1) et en présente des perspectives que nous considérons importantes (section 9.2).

9.1 Bilan du travail effectué

A l'heure actuelle, la tendance est à des applications réparties, hétérogènes et construites à partir de composants (systèmes de gestion de bases de données, serveurs web, etc) autonomes, hétérogènes et faiblement couplés. Cette tendance entraine un besoin grandissant pour des infrastructures garantissant la gestion d'un certain nombre de fonctionnalités communes à n'importe quelle application (par exemple le support des transactions, des requêtes, de la persistance ou encore de la duplication). Ainsi, au cours des dix dernières années, des travaux académiques et industriels ont proposé des infrastructures pour la construction d'applications réparties.

Les intergiciels sont des exemples de telles infrastructures et constituent un élément essentiel dans de nombreuses applications. Malheureusement, la rigidité de leurs spécifications contredit la volonté de souplesse. En effet, la plupart des infrastructures sont des boîtes noires, offrant des modèles figés et fermés, qui ne facilitent pas leur adaptation à des besoins applicatifs différents. Les services de duplication héritent de leur rigidité et de leurs limitations.

Des architectures plus flexibles et plus ou moins adaptables ont alors été proposées. Ces propositions n'ont néanmoins pas ou peu considéré l'adaptabilité des services de duplication par rapport au contexte non fonctionnel et dans tout ou partie des protocoles de duplication.

9.1.1 Taxonomie des protocoles de duplication

Il existe de nombreux protocoles de duplication utilisés dans différents domaines de l'informatique (SGC, SGBDR, MPR, les systèmes de fichiers répartis, les plate-formes à objets, etc.). Malheureusement, il y a très peu de rapprochement entre les travaux menés dans ces différents domaines. Il en résulte un grand nombre de protocoles où il n'est pas toujours facile de comprendre ce qui est propre à chaque domaine et les concepts qu'ils partagent. De plus chaque domaine apporte son propre vocabulaire rendant la comparaison plus délicate.

Dans cette thèse nous avons présenté une étude approfondie des concepts et techniques de duplication. Nous avons choisi trois domaines pour notre étude : les SGC, les SGBDR et les MPR. Ces trois domaines apportent chacun des concepts différents. De plus, on y trouve pour chacun d'eux des protocoles représentatifs.

Cette étude nous a permis de construire une grille de classification des protocoles de duplication permettant de les comparer. Nous avons dégagé onze dimensions que l'on retrouve dans les protocoles de duplication utilisés dans ces différents domaines. En fait, cette grille nous a permis de montrer que les différents protocoles mis en œuvre dans ces domaines partagent de nombreux points. Elle nous a permis aussi de proposer un vocabulaire commun pour décrire les protocoles de ces domaines. Nous avons également présenté les spécificités apportées par chacun des domaines étudiés.

Cependant, le point le plus important, pour l'objectif que nous nous sommes fixé, est que cette étude nous a permis de dégager ce qui est propre à la duplication de ce qui ne l'est pas :

- Sur les onze dimensions, certaines nous paraissent spécifiques à l'aspect duplication : le nombre de copies à consulter lors d'une écriture ou d'une lecture, les droits de mises à jour, le moment de la synchronisation, l'initiative de la mise à jour, la nature des mises à jour, la topologie de la propagation des mises à jour, la détection des mises à jour et la gestion des conflits . Comme ces dimensions peuvent être mises en œuvre de différentes façons, il nous a paru intéressant qu'un protocole de duplication soit adaptable sur ce point.
- Les dimensions de gestion de la concurrence, des fautes et des communications par contre ne sont pas spécifiques à l'aspect duplication. C'est pour cela que nous ne les considérons pas comme partie intégrante d'un protocole de duplication. Cependant, le protocole de duplication est fortement lié à ces aspects afin d'assurer la cohérence entre les copies.
- Il ressort également de l'état de l'art que chaque domaine apporte ses spécificités en ce qui concerne la gestion de la cohérence des objets du système. Certains gèrent une cohérence portant sur chaque objet, d'autres sur des groupes d'objets et enfin certains

gèrent la cohérence à certains moments. Cette cohérence n'étant pas spécifique à l'aspect duplication, nous avons choisi d'offrir l'adaptabilité du protocole de duplication par rapport à la gestion de cette cohérence.

9.1.2 Définition d'un canevas adaptable de services de duplication

L'approche par service/aspect nous parait très prometteuse étant donné qu'il est assez difficile de mettre en œuvre un protocole de duplication pour le programmeur d'applications. Cependant, si dans la littérature de nombreux travaux proposent un support de duplication, il n'y a pas consensus sur son rôle. Ainsi, afin de ne donner au service de duplication que ce qui est propre à cet aspect, nous lui donnons pour rôle de mettre en cohérence les copies (protocole de cohérence locale) d'un même objet et la gestion du cycle de vie des copies.

Cependant, même si l'approche par service nous paraît très intéressante, offrir un service couvrant l'ensemble des protocoles de cohérence locale semble utopique et inutilisable. Ainsi, nous pensons que l'approche consistant à définir un canevas semble plus adaptée.

Nous considérons que la définition d'un canevas apporte plus de puissance en permettant de décrire des services différents. Un canevas est une collection de patrons de description et d'utilisation de ressources qui représente une abstraction de ces ressources qui peut être projetée dans un modèle de programmation et déclinée en personnalités pour l'adapter aux standards du domaine considéré [Cou02].

Cependant sa prise en main et son utilisation sont complexes. Le canevas est un niveau d'abstraction supérieur par rapport au service. Il permet de décrire comment construire un service. Ainsi, nous ne pensons pas que notre travail doit être utilisé directement par le développeur d'applications. Il s'adresse aux développeurs d'intergiciels. Tout l'intérêt de notre travail, est d'avoir essayé "d'ouvrir" cet aspect. Pour l'utilisateur final cela ne présente que très peu d'intérêt. Au contraire, nous pensons que notre travail doit servir à des travaux considérant le problème de la composition d'aspects, problème se posant au niveau des intergiciels.

Ces travaux vont pouvoir tirer partie de la connaissance de la structure interne des services afin d'optimiser le processus de composition. Dans le cas d'une approche statique, ils pourront s'attacher à optimiser la composition, par exemple, en fusionnant des composants par réécriture de code. Dans le cas d'une approche dynamique, la composition peut se faire lors de l'exécution d'une application. Cela permet de rajouter une couche d'adaptation, qui en fonction de l'environnement (par exemple l'état des ressources matériels), décidera du protocole de cohérence locale adéquat.

De plus, notre approche permet de construire le service de duplication approprié. Le canevas est adaptable. Si le contexte non fonctionnel est transactionnel, alors il est possible de rajouter des composants gérant les interactions avec cet aspect (adaptabilité au contexte non fonctionnel). Si le service est utilisé sur un ensemble de serveurs ou sur un ensemble d'assistants personnels alors suivant le cas on peut construire un protocole répondant parfaitement aux contraintes de ces supports (adaptabilité dans tout ou partie des protocoles).

9.1.3 Définition formelle des services de duplication

Afin de caractériser les différents types de services de duplication qui peuvent être construits à partir de RS2.7, nous avons introduit la notion de modèle de cohérence locale. Un modèle de cohérence locale définit comment les utilisateurs d'un service perçoivent les différentes copies d'un même objet. Il est mis en œuvre par un protocole de cohérence locale.

Cette notion nous a permis de donner une vue formelle des services de duplication. A partir des modèles de cohérence locale, nous pouvons définir les interactions avec d'autres aspects non orthogonaux. Ainsi, nous dégageons deux sortes d'interactions. Nous avons défini les interactions avec d'autres aspects portant sur la gestion de l'objet logique (gestion de la concurrence sur l'objet logique, gestion des fautes), en fonction du modèle à mettre en œuvre et non pour chaque protocole existant. De plus, d'autres interactions avec des aspects non orthogonaux apparaisent dans la gestion de la cohérence entre les objets : ce sont les interactions entre la cohérence globale et la cohérence locale.

La liste des modèles proposés n'est pas exhaustive, mais nous paraît représentative. En construisant des modèles hybrides nous pouvons déjà décrire de nombreux protocoles. Nous considérons qu'il est tout à fait possible de définir de nouveaux modèles, proposant des ordonnancements "exotiques". Cela ne change en rien notre approche.

9.1.4 Adaptabilité de RS2.7 au contexte non fonctionnel

Un des principaux apports de cette thèse est l'adaptabilité de RS2.7 au contexte non fonctionnel. Cette adaptabilité permet à notre canevas d'obtenir des services de duplication utilisables dans différents contextes.

Pour notre cadre de travail, le contexte non fonctionnel est caractérisé par un modèle de cohérence globale spécifiant la garantie qu'a une application sur ses accès sur la mémoire ou les données. Un modèle est mis en œuvre par un protocole de cohérence globale prenant en charge la répartition, le contrôle de la concurrence, la duplication, etc. Un protocole de cohérence globale, gère un ensemble d'aspects non fonctionnels non orthogonaux. C'est pour cela que nous avons choisi d'offrir à RS2.7 la propriété d'adaptabilité par rapport à ces aspects.

L'analyse des modèles et des protocoles de cohérence globale nous a permis de dégager les interactions entre cohérence globale et cohérence locale. Cette analyse, nous montre que suivant le modèle de cohérence globale à mettre en œuvre, certains modèles de cohérence locale doivent être utilisés. Etant donné qu'un modèle de cohérence locale peut être mis en œuvre par différents protocoles de cohérence locale, il est donc possible d'utiliser différents protocoles de cohérence locale pour mettre en œuvre un même modèle de cohérence globale.

Ainsi, nous avons défini les composants nécessaires à rajouter afin qu'un protocole de cohérence locale puisse être utilisé avec un protocole de cohérence globale particulier. Ainsi, les mêmes protocoles de cohérence locale peuvent être réutilisés pour mettre en œuvre différents modèles de cohérence globale. A notre avis, cela est un point important. On voit là tout

l'intérêt de la séparation des considérations. Le développement d'un protocole de cohérence locale peut être capitalisé en le réutilisant dans un contexte non fonctionnel différent.

9.1.5 Adaptabilité dans tout ou partie des protocoles de cohérence locale

Un autre apport de cette thèse est l'obtention de l'adaptabilité dans tout ou partie des protocoles de cohérence locale. Cela permet de construire des protocoles de duplication parfaitement adaptés en fonction du matériel et du besoin de l'application en terme de duplication.

Pour obtenir cette adaptabilité nous proposons deux factorisations des protocoles de cohérence locale faisant ressortir leurs caractéristiques. En exhibant les différents composants d'un protocole de cohérence locale, nous permettons à d'autres d'agir sur cette structure en fonction de leurs besoins (section 9.1.2).

Le protocole abstrait de cohérence locale, factorisation de nature structurelle, définit cinq phases et leur ordonnancement. Nous définissons une phase comme un regroupement d'instructions ayant une certaine sémantique. Cette factorisation structurelle permet d'exhiber la structure des protocoles de cohérence locale afin d'introduire des points d'adaptabilité. Cette structure peut être manipulée et est modifiable selon les besoins.

Nous définissons également un ensemble de composants fonctionnels intervenant dans la construction des protocoles de cohérence locale. Ces composants reprennent les points remarquables des protocoles de duplication présentés au chapitre 2 et peuvent être implantés de diverses façons selon l'objectif recherché et le contexte.

Cette approche présente un intérêt majeur. Les protocoles de duplication peuvent être cons-truits par assemblage de composants et les protocoles existants peuvent être construits incrémentalement en ajoutant de nouveaux composants, structurels ou fonctionnels, aux assemblages existants. De plus, cela permet à notre canevas de s'adapter à diverses situations très différentes les unes des autres. Il est possible également de réutiliser des composants pour mettre en œuvre des protocoles différents.

Les différents composants présentés représentent certains choix que nous avons pu faire. Nous sommes conscient du fait que les composants fonctionnels et structurels ne permettent peut être pas de mettre en œuvre tous les protocoles de cohérence locale existants. Cependant, il est tout à fait possible d'en rajouter de nouveaux. Déjà, avec ceux qui sont présentés, il est possible d'en mettre en œuvre une quantité non négligeable. Notre objectif est de démontrer que cette approche présente un intérêt dans certaines situations où suivant le contexte on désire changer quelques composants d'un protocole afin de s'adapter à ce contexte.

9.1.6 Mise en œuvre et expérimentation

Nous avons décrit la mise en œuvre de différents services de duplication obtenus à partir de RS2.7. L'objectif de cette mise en œuvre a été de valider notre proposition et plus parti-

culièrement les propriétés d'adaptabilité au contexte non fonctionnel et dans tout ou partie des protocoles.

Les services que nous avons implantés ont été utilisés dans divers contextes non fonctionnels dans le cadre du projet européen PING. Ce projet visait à specifier, développer et démontrer une architecture pour des applications Internet interactives et massivement multi-participants qui soit souple et extensible. L'application retenue était un jeu en réseau devant supporter de nombreux joueurs. Afin d'obtenir de bonnes performances et du fait du nombre important d'objets, nous avons expérimenté des modèles de cohérence locale et globale relâchant la cohérence.

Une constatation ressort de nos expérimentations. Pour le développeur de services de duplication, nous, l'utilisation de RS2.7 permet d'obtenir très rapidement de nouveaux protocoles. Cependant, cela demande une certaine compréhension des concepts sous jacent ne simplifiant pas forcément la construction de services pour le non initié. De plus, aucune contrainte n'est donnée sur la composition des différents éléments, conduisant parfois à la construction de protocoles de duplication extravaguants. Cela rejoint ce que nous disions dans la section 9.1.2, nous voyons plutôt notre travail comme le point de départ de travaux sur la composition d'aspects. RS2.7 est une définition de l'aspect duplication. Ces travaux devront également proposer des outils pour la composition prenant en compte la sémantique de la composition.

9.2 Perspectives de recherche

Ce travail de thèse nous a permis d'analyser en profondeur l'aspect duplication. Cela nous a permis de dégager plusieurs axes de recherche, liés à la duplication, nous paraissant intéressants pour la suite :

Adaptation. RS2.7 offre la propriété d'adaptabilité, il peut donc être adapté par l'intermédiaire d'une couche d'adaptation. Ainsi, en fonction de l'environnement, cette couche se charge de construire le service de duplication adéquat. Cette approche parait intéressante dans les environnements mobiles sujets à de fréquentes déconnexions, à des ressources limitées, à une grande hétérogénéité du matériel, etc.

Dans le cas de la mise en œuvre de la tolérance aux fautes, il peut également être intéressant de choisir le protocole de duplication adéquat en fonction de l'environnement (comme dans [**?**]). Par exemple, si les coûts de communication varient au cours du temps, il est possible de passer d'un protocole de duplication active à un protocole de duplication passive. La duplication passive permet d'envoyer moins de messages que la duplication active.

Composition. L'approche que nous avons pris n'a pas pour objectif d'obtenir de bonnes performances. En décomposant les protocoles de cohérence locale et de cohérence globale, nous augmentons le nombre d'objets nécessaires à la mise en œuvre d'un protocole.

Il nous paraît important de considérer cet aspect par la suite. Ainsi, RS2.7 pourrait être

un cadre d'étude pour des travaux sur la composition d'aspects. Le canevas pour composants Fractal [1] [CLB+01] d'ObjectWeb permet de décrire des assemblages et de générer des implantations optimisées. Cette approche a l'avantage de tirer pleinement parti de notre canevas et de son architecture interne. L'assemblage peut être statique (à la compilation) ou dynamique (interprété à l'exécution). Le choix entre les deux approches dépend du compromis à faire entre performance et capacité d'adaptation dynamique (reconfiguration dynamique). Pour cela, il serait intéressant d'expérimenter des techniques de fusion/combinaison de code.

La composition dynamique soulève le problème de la cohérence lors du passage d'une composition à une autre : que devient la cohérence entre les copies, la cohérence du protocole lorsque l'on remplace un composant par un autre, etc. Les propriétés qu'il faudrait vérifier pour que la définition (c'est à dire l'assemblage de composants) et l'évolution (c'est à dire conservation/restauration de l'état des composants) des composants soient cohérentes concernent entre autre la cohérence structurelle, la cohérence comportementale et la cohérence sémantique

Expérimentation. Nous souhaiterions également, expérimenter notre canevas afin d'obtenir des services de duplication pour des contextes particuliers. Le contexte mobile nous semble être un bon cadre d'étude. Notamment par le fait que les protocoles que l'on rencontre sont assez complexes et que toutes les copies d'un même objet ne sont pas gérées de la même façon [GHPS96, PB99, WC99]. Cela serait un bon cadre d'étude pour mettre en œuvre des modèles de cohérence locale hybrides.

Nous vondrions également construire des personnalités EJB et CORBA de RS2.7. C'est à dire rajouter certains composants dans RS2.7 afin de prendre en compte leurs spécifications. Etant donné qu'il n'existe pas de spécification de l'aspect duplication pour CORBA et les EJB, RS2.7 est un premier pas dans ce sens. De plus nous n'offrons pas une structure figée et fermée. L'approche ouverte de RS2.7 pourrait être intéressante dans ces contextes.

Aides à la composition. Construire des services à partir de RS2.7 est une tâche demandant une grande connaissance du canevas. Par la suite, il serait intéressant d'étudier comment simplifier cette construction. Ainsi, il est possible de fournir des outils d'aide à la composition. Ces outils doivent prendre en compte la sémantique des composants afin de ne pas permettre de construire des services de duplication n'ayant aucun sens.

Chaque fois que nous entendrons dire : de deux choses l'une, empressons-nous de penser que, de deux choses, c'est vraisemblablement une troisième.

Rostand, Jean - Esquisse d'une histoire de la biologie, Conclusion

[1] http ://www.objectweb.org/architecture/component/index.html

Annexes

Carrera RS 2.7

Il était une fois, tout là-bas entre deux océans, une course folle qui s'appelait "Carrera Panamericana". De 1950 à 1954 cinq éditions seulement suffirent à faire entrer cette épreuve dans la grande légende de l'histoire du sport automobile. Il faut dire qu'au pays des Mayas ou des Aztèques, de Cuidad Jaurez à Tuxtla Guttierez (ou inversement) tout était toujours réuni pour faire de ces 1000 Milles à répétition, la course pas comme les autres. Le soleil, la poussière, les montagnes ou la plaine, les cols escarpés autant que les interminables et dangereux "bouts droits", l'ambiance, la vitesse, l'enthousiasme, l'exploit, les drames aussi, hélas, constituaient autant de piment à ces sprints échevelés. Ajoutons-y la proximité de marchés occidentaux (Etats-Unis, bien sûr, mais aussi Etats de l'Amérique centrale), la chaleur de l'accueil mexicain, et le niveau des prix offerts, et ne nous étonnons pas si cette course, dure pour les mécaniques et souvent périlleuse pour les hommes, finit par "envoûter" les plus grandes marques et les plus grand noms.

En 1973 Porsche ressucita en l'honneur de la 911 la glorieuse appellation Carrera, déjà rendue célèbre quinze ans plus tôt par la 356. La nouvelle Carrera était en fait une 911 2.4 S ayant subi un certain nombre de modifications. La cylindrée du moteur a été portée à 2,7 litres, ce qui permet d'obtenir une puissance de 210 ch à 6300 tr/mn pour un couple de 26 mkg à 5100 tr/mn

La Carrera RS 2.7 (RS signifiant Renn Sport) a initialement été présentée en version RSH, pour Homologation, en 17 exemplaires. Mais globalement, on compte deux versions de Carrera RS 2.7 : la Sport, ou ligthweight (M471), et la Touring (M472). Il était prévu de la construire à 500 exemplaires pour obtenir son homologation en Groupe 4 (GT modifié) mais elle eut un tel succés auprés des amateurs qu'elle fut produite à 1590 exemplaires (dont une cinquantaine de RSR 2.8) et homologuée en groupe 3 (GT normal).

Tous ces modèles se distinguaient par leur bouclier aérodynamique à l'avant et leur béquet arrière en "queue de canard", avec le nom de Carrera" s'étalant largement sur les cotés. Ces caractéristiques, devenues banales aujourd'hui, étaient alors dans leur grande nouveauté. Toutes les versions étaient équipés de disques ventilés comme ceux de la 917, et d'une carrosserie comportant un maximum d'éléments de plastique ou d'alliage léger (la sport pèse 975 kg, la Touring, 1075 kg). Mythe absolu, considéré encore aujourd'hui comme la meilleure 911 de l'histoire.

La RS 2.7 sera ensuite remplacée par la RS 3.0, produite à seulement 109 exemplaires, puis aura pour descendance la RS 92, la RS 3.8 et la GT3, mais ceci est une autre histoire.

Bibliographie

[AA90a] D. Agrawal and A. El Abbadi. Efficient techniques for replicated data management. In *Proceedings of the WorkShop on Management of Replicated Data*, pages 48–52, Houston, TX, USA, Nov. 1990.

[AA90b] D. Agrawal and A. El Abbadi. The tree quorum protocol : an efficient approach for managing replicated data. In *Proceedings of the 16th Int. Conf. on VLDB*, pages 243–254, Brisbane, Aug. 1990.

[AAAS97] D. Agrawal, G. Alonso, A. El Abbadi, and I. Stanoi. Exploiting atomic broadcast in replicated databases. In *In Proc. of Euro-Par*, 1997.

[ABHN91] M. Ahamad, J. Burns, P. Hutto, and G. Neiger. Causal memory. In *Proceeding of the 5th International Workshop in Distributed Algorithms*, pages 9–30. Delphi, Greece, springer-verlag edition, 1991.

[ABKW97] T. Anderson, Y. Breitbart, H. Korth, and A. Wool. Replication, consistency, and practicality : Are these mutually exclusive ? In *In Proc. of the SIGMODConf.*, 1997.

[ABMA88] R. Alonso, D. Barbara, H. Garcia Molina, and S. Abad. Quasi-copies : efficient data sharing for information retrieval systems. In *Proccedings of the International Conference on Extending Data Base Technology, EDBT'88*, 1988.

[ACD+96] C. Amza, A. Cox, S. Dwarkadas, P. Keeleher, H. Lu, R. Rajamony, W. yu, and W. Zwaenepoel. TreadMarks : Shared Memory Computing on Networks of Workstations. In *IEEE Computer*, volume 29, pages 118–128. february 1996.

[AD76] P. Alsberg and J. Day. A principle for resilient sharing of distributed resources. In *In Proc. of the Int. Conf. on Software Engineering*, pages 562–570, San Francisco, California, October 1976.

[AGM93] Y. Afek, G.Brown, and M. Merritt. Lazy caching. In *ACM Transaction on Programming Languages and Systems*, volume 15, pages 182–205. 1993.

[AH90] S. Adve and M. Hill. Weak ordering : A new definition. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, 1990.

[ARC01] Projet RNTL ARCAD. D1.1 - etat de l'art sur l'adaptabilité. Technical report, Ecole des Mines de Nantes, 2001.

[AS94] G. Agha and D. C. Sturman. A methodology for adapting patterns of faults. In Kluwer Academic Publishers, editor, *Foundations of Dependable Computing : Models and Frameworks for Dependable Systems*, volume 1, pages 23–60, 1994.

[ASC85] A. El Abbadi, D. Skeen, and F. Christian. An efficient fault-tolerant protocol for replicated data management. In *Proceedings of the 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 215–228, Portland, March 1985.

[ASP01] Aspectj, 2001. http ://aspectj.org/servlets/AJSite.

[AT86] A. El Abbadi and S. Toueg. Availability in partitioned replicated databases. In *Proceedings of the 5th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 240–251, Cambridge, March 1986.

[AW94] H. Attiya and J. Welch. Sequential consistency versus linearisability. In *ACM Transaction on Computer Systems*, volume 12, pages 91–122. 1994.

[BC98] G. Brun-Cottan. *Cohérence de données répliquées partagées par un groupe de processus coopérant à distance.* PhD thesis, Université Pierre et Marie Curie, Paris VI, september 1998.

[BCM95] G. Brun-Cottan and M. Makpangou. Adaptable Replicated Objects in Distributed Environments. Technical Report 2593, INRIA, may 1995.

[BDD+00] R. Basset, P. Déchamboux, S. Drapeau, L. Garcia-Banuelos, and A. Lefebvre. Jorm : a java object repository mapping system, internal interface specification. Technical report, Internal deliverable, RNRT CORSICA Project, France Télécom R&D, Editor Pascal Déchamboux, 24 may 2000.

[BG84] P.A. Bernstein and N. Goodman. An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. In *ACM TODS*, volume 9, pages 596–615. Dec 1984.

[BGM90] D. Barbara and H. Garcia-Molina. The case for controlled inconsistency in replicated data. In *Proceedings of the Workshop on Management of Replicated Data*, Houston, TX, USA, Nov. 1990.

[BGM91] D. Barbara and H. Garcia-Molina. The demarcation protocol : a technique for maintaining arithmetic constraints in distributed database systems. Technical Report CS-TR-320-91, Princeton University, April 1991.

[Bir85] K. Birman. Replication and Fault-Tolerancein the ISIS system. In *10th ACM Symposium on Operating Systems Principles*, pages 79–86, december 1985.

[Bir93] K. Birman. The process group approach to reliable distributed computing. *Communication of the ACM*, 36(12), December 1993.

[BJRA85] K. Birman, T. Joseph, T. Raeuchle, and A. Abadi. Implementing fault-tolerant distributed objects. In *IEEE Transactions on Software Engineering*, volume 11, pages 502–508. 1985.

[BKR+99] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, and A. Silberschatz. Update propagation protocls for replicated databases. In *In Proc. of the SIGMOD Conf.*, 1999.

[BLMS82] A. Birell, R. Levin, R. M.Needham, and M. D. Schroeder. Grapevine : An exercise in distributed computing. In *Communications of the ACM*, volume 25, pages 260–274, avril 1982.

[BM91] B. Bershad and M.Zekauskas. Midway : Shared memory parallel programming with entry consistency for distributed memory multiprocessors. Technical Report CMU-CS-91-170, Carnegie Mellon University, Department of Computer Science, Pittsburgh, PA,USA, 1991.

[BMST93] N. Budhijara, K. Marzullo, F.B. Schneider, and S. Toueg. The primary-backup approach. In *Distributed Systems*, chapter 8, pages 199–216. Addison-Wesley, 1993.

[BPR97] P. Beaugendre, T. Priol, and C. René. Cobra : a corba-compliant programming environment for high-performance computing. Technical Report PI-1141, IRISA, Rennes, France, 1997.

[BR94] K. Birman and R. Van Renesse. Reliable distributed computing with the isis toolkit. In *IEEE Computer Society Press.* 1994.

[BR994] *Reliable Distributed Computing with the Isis Toolkit.* IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

[BS97] S. Bobrowski and G. Smith. *Oracle8 Server Replication.* Oracle Corporation, June 1997.

[BS99] M. N. Bouraqadi-Saâdani. Un cadre réflexif pour la programmation par aspects. In *Langages et Modèles à Objets (LMO'99)*, Villefranche sur Mer - France, January 1999. Hermes.

[BVG87] P.A. Bernstein, V.Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[CBZ91] J. Carter, J. Benett, and W. Zwaenepoel. Implementation and Performance of Munin. In *Proceeding of the 13th Symposium on Operating Systems Principles*, volume 25, pages 152–164, Pacific Grove, California, USA, 1991. Operating Systems Review.

[CDD+00] C. Collet, F. Dang Tran, S. Drapeau, L. Garcia-Banuelos, A. Gérodolle, D. Houatra, T. Nedelec, L. Parmentier, and C.L. Roncancio. Persistency, replication and real time consistency : State of the art. Technical Report RR-1031-I-LSR-13, Rapport de contrat projet européen PING IST-1999-11488, IMAG-LSR, ENST et France Télécom R&D, november 2000.

[CDD+01] C. Collet, F. Dang Tran, S. Drapeau, L. Garcia-Banuelos, A. Gérodolle, T. Nedelec, L. Parmentier, and C.L. Roncancio. Object and event management : First specification. Technical Report PING-IMAG-07-D22-R1-1, Rapport de contrat projet européen PING IST-1999-11488, IMAG-LSR, ENST et France Télécom R&D, june 2001.

[CE99] K. Czarnecki and U. Eisenecker. *Generative Programming : Methods, Techniques, and Applications*, chapter Aspect-Oriented Decomposition and Composition. Addison-Wesley, 1999.

[CF78] L. Censier and P. Feautrier. A new solution to coherence problems in multicache systems. In *IEEE Transactionson Computers*, volume 27, pages 1112–1118. 1978.

[CLB+01] T. Coupaye, R. Lenglet, M. Beauvois, E. Bruneton, and P. Déchamboux. Composants et Composition dans l'Architecture de Systèmes Répartis. In *Journèes Composants : flexibilité du système au langage, ASF (ACM SIGOPS France)*, Besancon, France, october 2001.

[Col00] C. Collet. The NODS project : Networked Open Database Services. In *ECOOP*, 2000.

[Cou02] T. Coupaye. Séminaire interne France Telecom R&D, septembre 2002.

[CRR96] P. Chundi, D. Rosenkrantz, and S. Ravi. Deferred updates and data placement in distributed databases. In *In Proc. of the Int. Conf. on Data Engineering*, 1996.

[CS86] A. Chan and D. Skeen. The reliability subsystem of a distributed database manager. Technical Report CCA-85-02, Computer Corporation of America, 1986.

[CT96] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. In *Journal of the ACM*, volume 43, pages 225–267, mars 1996.

[DE89] W. Du and A. Elmagarmid. Quasi serializability : a correctness criterion for global concurrency control in interbase. In *Proceedings of the fifteenth international conference on Very large data bases*, pages 347–355, Amsterdam, The Netherlands, 1989.

[DGMS85] S. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *Computing Surveys*, 17(3) :341–370, September 1985.

[DR01] S. Drapeau and C. Roncancio. Concepts et Techniques de Duplication. Technical report, Laboratoire LSR IMAG, 2001.

[DRD02a] S. Drapeau, C. Roncancio, and P. Déchamboux. Overview of an adaptable replication framework. In *Poster in International Symposium on Distributed Objects and Applications (DOA02)*. Extended abstract published as a technical report of the University of California at Irvine, October 2002.

[DRD02b] S. Drapeau, C. Roncancio, and P. Déchamboux. RS2.7 : an Adaptable Replication Framework. In *18émes Journées Bases de Données Avancées (BDA02)*, October 2002.

[DRD03] S. Drapeau, C.L. Roncancio, and P. Déchamboux. RS2.7 : un Canevas Adaptable de Duplication. In *à paraitre dans TSI*, 2003.

[DSB86] M. Dubois, C. Scheurich, and F. Briggs. Memory access buffering in multiprocessors. In *Proceedings the 13th Annual International Symposium on Computer Architecture*, pages 434–442. IEEE Computer Society, 1986.

[DYK01] L. DeMichel, L. Yalçinalp, and S. Krishnan. *Enterprise Java Beans Specification Version 2.0 Proposed Final Draft 2.* Sun Microsystems Inc., april 24 2001.

[Eag81] D.L. Eager. Robust concurrency control in distributed databases. Technical Report CSRG-135, Computer System Research Group, University of Toronto, Oct. 1981.

[EK89] S. J. Eggers and R. H. Katz. The effect of sharing on the cache and bus performance of parallel programs. In *Proccedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 1989.

[ES83] D.L. Eager and K.C. Sevcik. Achieving robustness in distributed database systems. In *ACM-TODS*, volume 8, pages 354–381. Sept. 1983.

[FGS97] P. Felber, R. Guerraoui, and A. Schiper. A corba object group service. In *ECOOP'97 Workshop on CORBA*, 1997.

[FGS98] P. Felber, R. Guerraoui, and A. Schiper. The implementation of a corba object group service. *Theory and Practice of Object Systems*, 4(2) :93–105, 1998.

[FGS00] P. Felber, R. Guerraoui, and A. Schiper. Replication of CORBA Objects. volume LNCS 1752, pages 254–276. Springer-Verlag, Berlin, Heidelberg, 2000.

[FP98] J. C. Fabre and T. Perennou. A metaobject architecture for fault-tolerant distributed systems : The friends approach. *IEEE Transactions on Computers*, 47(1) :78–95, 1998.

[GGM95] B. Garbinato, R. Guerraoui, and K. R. Mazouni. Implementation of the GARF Replicated Object Plateform. *Distributed Systems Engineering Journal*, 2 :14–27, 1995.

[GHOS96] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The Danger of Replication and a Solution. In *ACM SIGMOD International Conference on Management of Data*, Montreal, June 1996.

[GHPS96] J. N. Gray, P. Helland, P.O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *ACM SIGMOD Conf. on Management of Data*, Canada, 1996.

[Gif79] D. Gifford. Weighted voting for replicated data. In *Proceeding of the 7th Symposium on Operating Systems Principles*, pages 150–159, Pacific Grove, California, USA, 1979.

[GL93] R. Golding and D. Long. Modeling replica divergence in a weak-consistency protocol for global-scale distributed data bases. Technical Report UCSC-CRL-93-09, University of California, Santa Cruz, California, USA, 1993.

[GLM+90] K. Gharachorloo, D. Lenoski, J. Maudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 15–26, Seattle, Washington, USA, 1990. IEEE Computer Society.

[GN95] M. Gallersdörder and M. Nicola. Improving Performance in Replicated Databases Through Relaxed Coherency. In *In Proc. of the VLDB Conf.*, Zürich, Switzerland, 1995.

[Gol92] R. Golding. *Weak-Consistency Group Communication and Membership*. PhD thesis, University of California, Santa Cruz, California, USA, december 1992.

[Gra79] J. Gray. Notes on database operating systems. In *Lecture Notes in Computer Science*. 1979.

[GSC+83] N. Goodman, D. Skeen, A. Chan, U. Dayal, S. Fox, and D. Ries. A Recovery Algorithm for a Distributed Database System. In *Proceedings of the 2nd ACM SIGACT-SIGMOD*, pages 8–15, Atlanta, GA, march 1983.

[HA90] P. Hutto and M. Ahamad. Slow memory : Weakening consistency to enhance concurrency in distributed shared memories. In *Proceeding of the 10th International Conference on Distributed Computing Systems*, pages 302–311, Paris, France, 1990. IEEE Computer Society.

[HAA99] J. Holliday, D. Agrawal, and A. El Abbadi. The performance of database replication with group multicast. In *In Proc. of the Int. Symp. on Fault-Tolerant Computing (FTCS)*, 1999.

[HOT97] W. Harrison, H. Ossher, and P. Tarr. Using Delegation for Software and Subject Composition. Technical Report RC 20946, IBM Thomas J. Watson Research Center, 1997.

[HT93] V. Hadzilacos and S. Toueg. *Fault-Tolerant Broadcasts and Related Problems*, chapter 5, pages 97–145. [Mul93] second edition, 1993.

[II94] IONA Technologies Ltd. and Isis Distributed Systems Inc. An introduction to orbix + isis. Technical report, 1994.

[Inf98] Informix. *Enterprise Replication : A High-Performance Solution for Distributing and Sharing Information*, 1998.

[ION95] IONA Technologies Ltd. The orbix architecture. Technical report, Dublin ,Ireland, 1995.

[Joh97] R.E. Johnson. Framework = Components + Patterns. In *Communication of the ACM*, volume 40, pages 39–42. October 1997.

[KA88] A. Kumar and A.Segev. Optimizing voting-type algorithms for replicated data. In *Advances in Database Technology EDBT'88*, LNCS 303, pages 428–442. 1988.

[KA98] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *In Proc. of the Int. Conf. on Distributed Computing Systems (ICDCS*, 1998.

[KA00a] B Kemme and G. Alonso. Don't be lazy, be consistent : Postgres-R, a new way to implement Database Replication. In *Proc. of 26th International Conference on Very Large Databases (VLDB), Cairo, Egypt*, 2000.

[KA00b] B. Kemme and G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Transactions on Database Systems*, 2000.

[KB91] N. Krishnakumar and A. Bernstein. Bounded ignorance in replicated systems. In *In Proc. of PODS*, 1991.

[Kel95] P. Keleher. *Lazy Release Consistency for Distributed Shared Memory.* PhD thesis, Rice University, Department of Computer Science, Houston, Texas, USA, 1995.

[KG96] J. Kleinöder and M. Golm. Transparent and Adaptable Object Replication Using a Reflective Java. Technical Report TR-I4-96-07, Universität Erlangen-Nürnberg, september 1996.

[Kin99] A. Kindler. A Classification of Consistency Models. Technical Report B99-14, Humboldt-Universitat zu Berlin, Institut fur Informatik, D-10099 Berlin, Germany, october 1999.

[KKvST97] A.M. Kermarrec, I. Kuz, M. van Steen, and A. Tanenbaum. A Framework for Consistent, Replicated Web Objects. Technical Report IR-431, Vrije Universiteit, Faculty of Mathematics and Computer Science, september 1997.

[KKvST98] A.-M. Kermarrec, T. Kuz, M. van Steen, and A. Tanenbaum. A Framework for Consistent, Replicated Web Objects. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, pages 276–284, Amsterdam, The Netherlands, 1998. IEEE Computer Society.

[KLM+97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maedaand, C. Videira Lopes, J.M. Loingtier, and J. Irwin. In *ECOOP'97 - Object-Oriented Programming, 11th European Conference, Jyväskylä, Finland, June 9-13, 1997, Proceedings*, 1997.

[Lam78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978.

[Lam79] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. In *IEEE Transactions on Computers*, volume 28, pages 690–691. 1979.

[Lar89] Larousse. *petit Larousse illustré*. Librairie Larousse, 1989.

[LBS01] T. Ledoux and M. N. Bouraqadi-Saâdani. Le point sur la programmation par aspects. *TSI*, 2001.

[LH89] K. Li and P. Hudak. Memory coherence in shared virtual memory systems. In *ACM Transaction on Computer Systems*, volume 7, pages 321–359. 1989.

[LLM99] K. Lieberherr, D. Lorenz, and M. Mezini. Programming with aspectual components. Technical report, College of Computer Science, Northeastern University, Boston, Massachusetts, April 1999.

[LLS90] R. Ladin, B. Liskov, and L. Shrira. Lazy Replication : Exploiting the Semantics of Distributed Services. In *9th Symposium on Principles of Distributed Computing*, pages 43–57, New-York, USA, 1990.

[LLSG92] R. Ladin, B. Liskov, L. Shrira, and S. Ghemawat. Providing High Availability Using Lazy Replication. In *ACM Transaction on Computer Systems*, number 10(4), pages 360–391, 1992.

[Lob00] O. Lobry. *Support mémoire adaptable pour serveurs de données répartis*. PhD thesis, Université Joseph Fourier - Grenoble 1, october 2000.

[LOML01] K. Lieberherr, J. Ovlinger, M. Mezini, and D. Lorenz. Modular programming with aspectual collaborations. Technical report, College of Computer Science, Northeastern University, Boston, Massachusetts, March 2001.

[Lor95] M. Lorenz. *Rapid Software Development with Smaltalk.* SIGS Books, 1995.

[LS88] L. Lipton and J. Sandberg. Pram : A scalable shared memory. Technical Report CS-TR-180-88, Princeton University,Department of Computer Science, 1988.

[Maf95] S. Maffeis. Adding Group Communication and Fault-Tolerance to CORBA. In *USENIX Conference on Object-Oriented Technologies*, june 1995.

[MBBP89] B. Martin, C. Bergan, W. Burkhard, and J. Paris. Experiencewith parpc. In *Proceeding of the 1989 USENIX Winter Conference*, pages 1–12. USENIX Association, 1989.

[MHO96] Hafedh Mili, William Harrison, and Harold Ossher. Supporting Subject-Oriented Programming in Smalltalk. In *TOOLS*, August 1996.

[Mic95] Microsoft Corporation. *The Component Object Model Specification*, march 1995.

[Mic02] Microsoft Corporation. *.net. http ://www.microsoft.com*, 2002.

[MMSA+96] L. Moser, P. Melliar-Smith, D. Agarwal, R. Budhia, and C. Lingley-Papadopoulos. Totem : A fault-tolerant multicast group communication system. In *Communications of the ACM*, 1996.

[MMSN98] L. Moser, P. Melliar-Smith, and P. Narasimhan. Consistent object replication in the eternal system. *Theory and Practice of Object Systems*, 4(2) :81–92, 1998.

[MMVB00] C. Marchetti, M. Mecella, A. Virgillito, and R. Baldoni. An interoperable replication logic for corba systems. In *In Proceedings of the International Symposium on Distributed Objects and Applications*, pages 7–16, Antwerp, Belgium, September 2000.

[MPS89] S. Mishra, L. Peterson, and R. Schlichting. Implementing fault-tolerant replicated ojects using psync. In *Eigth Symposium on Reliable Distributed Systems*, pages 42–52, 1989.

[MRZ94] M. Mizuno, M. Raynal, and J. Z. Zhou. Sequential consistency in distributed systems. In F. Mattern K. Birman and A. Schiper, editors, *Proc. Int. Workshop "Theory and Practice in Distributed Systems"*, number 938 in LNCS, pages 227–241, Dagstuhl, Germany, 1994. Springer-Verlag.

[MSEL99] G. Morgan, S. Shrivastava, P. Ezhilchelvan, and M. Little. Design and implementation of a corba fault-tolerant object group service. In *In Proceedings of the Second IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems*, Helsinki, Finland, June 1999.

[MT00] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. In *IEEE Transactions on Software Engineering*, volume 26, january 2000.

[Mul93] S. Mullender. *Distributed Systems.* Addison-Wesley - ACM Press, second edition, 1993.

[Nel81] B. Nelson. *Remote Procedure Call.* PhD thesis, Carnegie Mellon University, Department of Computer Science, Pittsburgh, PA, USA, 1981.

[Obj03] ObjectWeb. *Jonathan White paper*, http ://www.objectweb.org/jonathan/current/doc/src/white-paper/white_paper_tex.html, 2003.

[OD83] D. Oppen and Y. Dalal. The Clearinghouse : A Decentralized Agent for Locating Named Objects in a Distributed Environment. In *ACM Transactions on Office Information Systems*, volume 1, pages 230–253, july 1983.

[OKK+96] H. Ossher, M. Kaplan, A. Katz, W. Harrison, and V. Kruskal. Specifying Subject-Oriented Composition. *Theory and Practice of Object Systems*, 2(3), 1996.

[OMG97] OMG. Corba services : Common object services specifications. Technical report, Object Management Group, Framingham, MA, USA, 1997.

[OMG98] OMG. *The Common Object Request Broker : Architecture and Specififcation.* Object Management Group, Framingham, MA, USA, february 1998.

[OMG99] OMG. The common object request broker : Architecture and specification, 2.3 edition. Technical report, Object Management Group Technical Committee Document formal/98-12-01, June 1999.

[OMG01] OMG. Fault tolerant corba. Technical report, Object Management Group Technical Committee Document formal/2001-09-29, 2001.

[OV98] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 2nd edition, 1998.

[PB99] E. Pitoura and B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. In *Transactions on Knowledge and Data Engineering*, November 1999.

[Pet87] L. Peterson. Preserving context information in an ipc abstraction. In *Sixth Symposium on Reliability in Distributed Software and Database Systems*, pages 22–31, 1987.

[PGS97] F. Pedone, R. Guerraoui, and A. Schiper. Transaction reordering in replicated databases. In *In Proc. of the Symp. on Reliable Distributed Systems*, 1997.

[PL91] C. Pu and A. Leff. Replica control in distributed systems : an asynchronous approach. In ACM SIGMOD Records, editor, *Proceedings of the International Conference on the Management of Data*, pages 377–386, Denver, CO, USA, mai 1991. ACM Press.

[PMS99] E. Pacitti, P. Minet, and E. Simon. Fast algorithms for maintaining replica consistency in lazy master replicated databases. In *In Proc. of the VLDB Conf.*, 1999.

[PNMS97] L. E. Moser P. Narasimhan and P. M. Melliar-Smith. Replica Consistency of CORBA Objects in Partitionable Distributed Systems. *Distributed Systems Engineering*, 4(3) :139–150, september 1997.

[PNMS02] L. E. Moser P. Narasimhan and P. M. Melliar-Smith. Strong Replica Consistency for Fault-Tolerant CORBA Applications. *Journal of Computer System Science and Engineering*, 2002.

[PV91] D. Powell and P. Verissimo. *Delta4 : A Generic Architecture for Dependable Computing*, chapter 6, pages 89–123. Distributed Fault-Tolerance. Springer-Verlag, 1991.

[RB94] R. Van Renesse and K. Birman. *Fault Tolerant Programming Using Process Groups.* in **??**, 1994.

[RBG+95] R. Van Renesse, K. Birman, B. Glade, K. Guo, M. Hayden, T. Hickey, D. Malki, A. Vaysburd, and W. Vogels. Horus : a flexible group communications system. Technical Report TR95-1500, Cornell University, Department of Computer Science Research, Ithaca, New York, USA, 1995.

[RKC93] M. Rusinkiewicz, P. Krychniak, and A. Cichocki. Towards a model for multidatabase transactions. In *International Journal of Intelligent and Cooperative Information Systems*, volume 1, pages 579–517, 1993.

[Rog97] G.F. Rogers. *Framework Based Software Development in C++.* Prentice Hall, 1997.

[RP95] K. Ramamritham and C. Pu. A formal characterization of epsilon serialisability. In *IEEE Transactions on Knowledge and Data Engineering*, volume 7, pages 997–1007, december 1995.

[Sch90] F.B. Schneider. Implementing fault-tolerant services using the state machine approach : a tutorial. In *ACM Computing Surveys*, volume 22, pages 299–319. December 1990.

[Sim95] C. Simonyi. The Death Of Computer Languages, The Birth of Intentional Programming. Technical Report MSR-TR-95-52, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399 USA, september 1995.

[SKK+90] M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegel, and D. Steere. Coda : A Highly Available File System for a Distributed Workstation Environment. In *IEEE Transactions on Computers*, volume 39, pages 447–459, 1990.

[SL76] D.G. Severance and G. Lohman. Differential files : their application to the maintenance of large databases. In *ACM-TODS*, volume 1. Sept. 1976.

[SS90] M. Satyanarayanan and E. Siegel. Parallel communication in a large distributed environment. *IEEE Transactions on Computers*, 39(3) :328–348, 1990.

[Sto79] M. Stonebraker. Concurrency control and consistency of multiple copies of data in distributed ingres. *IEEE Transactions on Software Engineering 3*, 3 :188–194, 1979.

[Syb00] Sybase, Inc. *Replication Server Version 12.0 Administration Guide*, January 2000.

[TDP+94] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch. Session guarantees for weakly consistent replicated data. In *Proceedings of the third International Conference on Parallel and Distributed Information Systems*, pages 140–149, Austin, Texas, USA, 1994. IEEE Computer Society.

[TK97] G. Thia-Kime. *Critères de cohérence pour données partagées à support réparti.* PhD thesis, Université de Rennes 1, Octobre 1997.

[Ver99a] *Chapter 22, Asynchronous replication*, 1999.

[Ver99b] *Chapter 22, Fault Tolerant Server*, 1999.

[vSHT97] M. van Steen, P. Homburg, and A. Tanenbaum. The Architectural Design of Globe : a Wide-Area Distributed System. Technical Report IR-422, Vrije Universiteit, Faculty of Math and Computer Science, Amsterdam, The Netherlands, mars 1997.

[vSHT99] M. van Steen, P. Homburg, and A. Tanenbaum. Globe : A Wide-Area Distributed System. *IEEE Concurrency*, pages 70–78, janvier-mars 1999.

[WC99] G. D. Walborn and P. K. Chrysanthis. Transaction Processing in PROMOTION. In *14th ACM Annual Symposium on Applied Computing*, San Antonio Tx, 1999.

[WNT98] S. Weber, P. Nixon, and B. Tangney. A Flexible Framework for Consistency Management in Object Oriented Distributed Shared Memory. Technical report, Distributed System Group, Department of Computer Science, Trinity College, Dublin, Ireland, october 1998.

[WQ87] G. Wiederhold and X. Qian. Modeling asynchrony in distributed databases. In *Proceedings of the 3rd International Conference on Data Engineering*, pages 246–250, 1987.

[WQ90] G. Wiederhold and X. Qian. Consistency control of replicated data in federated databases. In *Proceeedings of the 1st Workshop on the Management of Replicated Data*, pages 130–132, Houston, November 1990.

[WYP97] K.L. Wu, P. Yu, and C. Pu. Divergence control algorithms for epsilon serializability. *IEEE Transactions on Knowledge and Data Engineering*, 9(2) :262–274, mars-avril 1997.

[ZAL96] A. Zeroual, A. Abdellatif, and M. Lassadi. *CA-OPEN INGRES*. June 1996.

...

J'ai l'impression d'en avoir terminé finalement avec ce qui commença en Cours Préparatoire : les devoirs le soir... peut être pas...

RS2.7 : un Canevas Adaptable de Services de Duplication

Résumé : Notre objectif dans cette thèse est de donner la propriété d'adaptabilité à l'aspect duplication. La séparation des préoccupations et l'approche par services permettent au développeur d'applications de s'abstraire de l'aspect duplication lors de ses développements. Cependant, ces approches souffrent d'une limitation majeure : il semble très difficile, voir impossible, de fournir un service/aspect générique de duplication pouvant être paramétré afin d'être utilisé dans différents contextes d'exécution et couvrant l'ensemble des protocoles existants.

Ces constatations nous ont conduit à la définition d'un canevas de services de duplication, nommé RS2.7. RS2.7 est le squelette d'un service de duplication définissant sa structure. Il permet d'obtenir des services de duplication indépendants de tout code propre à l'application, pouvant être utilisés dans différents contextes non fonctionnels (transactionnel, mémoires partagées, etc.) et prenant en compte les contraintes et les protocoles spécifiques à chaque domaine.

Nos contributions portent sur trois axes : (1) la modélisation des services de duplication pouvant être obtenus à partir de RS2.7, (2) l'adaptabilité du canevas par rapport au contexte non fonctionnel et (3) l'adaptabilité dans tout ou partie des protocoles de duplication.

RS2.7 a été mis en œuvre et notre validation porte sur la démonstration des caractéristiques d'adaptabilité offertes. Nous cherchons à montrer que notre canevas permet d'obtenir des services très variés et convenant pour divers contextes non fonctionnels.

Mots-clefs : duplication, canevas adaptable, cohérence locale, cohérence globale, décomposition.

RS2.7 : an Adaptable Framework for Replication Services

Abstract : The objective of our thesis is to make the replication aspect adaptable. Separation of concerns and services-oriented approaches allow the application developer to disregard the replication aspect during development. However, these approaches suffer from a major limitation : it seems very difficult - even impossible - to provide a generic replication service/aspect, which can be parameterised in order to be used in various execution contexts and which cover the whole set of the existing protocols.

These considerations have lead us to the definition of a framework of replication services named RS2.7. RS2.7 is the skeleton of a replication service defining its structure. It makes possible to obtain replication services independent of any application code, which can be parametrised to be used in various non-functional contexts (transactional, distributed shared memories, etc.) and which take into account constraints and protocols specific to each field.

Our contributions focus on three main issues : (1) modelisation of replication services obtained from RS2.7, (2) adaptability of the framework to the non-functional context and (3) adaptability in whole or part of the replication protocols.

Several implementations of RS2.7 were developed. The validation has shown the adaptability characteristic offered by RS2.7. The framework makes it possible to obtain different services suited to various non-functional contexts.

Keywords : replication, adaptable framework, coherency, consistency, decomposition.