Transactions Adaptables pour les Environnements Mobiles

Patricia Serrano Alvarado

UNIVERSITÉ JOSEPH FOURIER - GRENOBLE I

Thèse

pour obtenir le grade de

Docteur de l'Université Joseph Fourier

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

Discipline : Informatique

Présentée et soutenue publiquement par

Patricia SERRANO ALVARADO

Le 6 février 2004

Transactions Adaptables pour les Environnements Mobiles

Composition du jury

Président :	M Farid OUABDESSELAM
Rapporteurs :	M Patrick VALDURIEZ
	M Mads NYGÅRD
Examinateurs :	M Jean FERRIÉ
Directeurs de thèse :	M Michel ADIBA
	Mme Claudia Lucía RONCANCIO

Thèse préparée au sein du laboratoire Logiciels, Systèmes, Réseaux - IMAG

A mis padres,
María Teresa y Francisco,
por su infinito amor,
porque han vigilado y acompañado
cada uno de mis pasos,
por creer en mi.
Pero sobre todo, porque el amor que se profesan,
es el mejor ejemplo que han podido darme.

A mes parents,
María Teresa et Francisco,
pour leur amour infini,
parce qu'ils ont veillé sur moi
et ont accompagné chacun de mes pas,
parce qu'ils croient en moi.
Mais surtout, car l'amour qu'ils partagent
est le meilleur exemple qu'ils ont pu me donner.

Remerciements

Je tiens à remercier :

Farid OUABDESSELAM, Professeur à l'Université Joseph Fourier (UJF) pour m'avoir fait l'honneur de bien vouloir présider le jury de cette thèse.

Patrick VALDURIEZ, Directeur de recherche à l'Institut National de Recherche en Informatique et en Automatique (INRIA) et Mads NYGARD, Professeur à l'Université Norvégienne de Science et Technologie (NTNU) pour avoir accepté de juger ce travail et pour avoir consacré une partie de leur précieux temps à la lecture de ce manuscrit.

Jean FERRIÉ, Professeur à l'Université Montpellier II, pour avoir accepté de faire parti du jury et pour ses commentaires qui ont aidé à améliorer la qualité de ce manuscrit.

Claudia RONCANCIO, Maître de conférence à l'Institut National Polytechnique de Grenoble (INPG), qui a suivi de très prés ce travail. Pour nos innombrables discussions et ses commentaires toujours très constructifs et intéressants qui ont contribué à la compréhension et la qualité de ce travail. Claudia, ta motivation, ton enthousiasme et tes encouragements ne pouvaient que m'amener à bon terme dans mon travail.

Michel ADIBA, Professeur à l'Université Joseph Fourier (UJF) pour l'intérêt qu'il a toujours porté sur mon travail et pour ses commentaires toujours intéressants qui ont aidé à la qualité de cette thèse.

Christine COLLET, Professeur à l'Institut National Polytechnique de Grenoble (INPG) pour m'avoir accueillie dans son équipe où l'aspect multiculturel m'a permis d'apprendre de nouvelles choses chaque jour.

Luciano GARCÍA BAÑUELOS, tout d'abord pour son amitié et les agréables moments qu'il nous a fait passer grâce à sa bonne humeur et à ses qualités d'artiste. Pour les nombreuses discussions qui m'ont beaucoup appris. Mais surtout, je te remercie de tout mon cœur de m'avoir présenté l'homme de ma vie.

Lizbeth et Caty, mes amies et confidentes, car malgré le manque de temps, nous avons sû cultiver une belle amitié.

Les membres du projet NODS : Quynh, Trinh, Giang, Pili, Trini, Khalid, Gennaro, Tanguy, Denis, Mourad, Levent, Laurent, Fabrice, qui par leurs différentes origines et cultures m'ont appris que le monde est très grand mais que les cœurs sont tous identiques. Particulièrement, Cyril LABBÉ qui a contribué à la valorisation de mon travail par une étude analytique très prometteuse. Noha IBRAHIM et Noelle MERLE qui à travers leurs projets d'étude ont contribué à la validation expérimentale d'une partie de cette thèse. Christophe BOBINEAU pour son intérêt pour mon travail, pour avoir dédié une partie de son temps à la lecture de ce manuscrit et pour toutes ses remarques qui ont beaucoup aidé à améliorer ce travail. Genoveva, qui par sa force de travail m'a motivé à plusieurs reprises. Ceux qui sont rentrés avant moi : Rafael, José Luis, Edgar, Elizabeth et Marlon.

Les membres du laboratoire Logiciels, Systèmes, Réseaux (LSR) qui font vivre cet établissement avec leur bonne humeur et leur travail.

Les membres de l'Action Spécifique CNRS Mobilité et Accès aux Données : Jalel BEN OTHMAN, Guy BERNARD, Christophe BOBINEAU, Gérôme CANALS, Sophie CHABRIDON, Bruno DEFUDE, Jean FERRIÉ, Stéphane GANÇARSKI, Rachid GUERRAOUI, Corine HARI, Ab-

delkrim LAHLOU, Cécile LE PAPE, Pascal MOLI, Gérald OSTER, Philippe PUCHERAL, Alain RINGAPAIN, Claudia RONCANCIO, Patrick VALDURIEZ et Nicolas VIDOT.

Le Consejo Nacional de Ciencia y Tecnologia (CONACyT) qui par son soutient économique a rendu possible ces études. J'espère que j'aurais l'occasion de contribuer au développement scientifique du Mexique afin de remercier de la confiance qui m'a été donnée.

La Société Française d'Exportation de Ressources Educatives (SFERE) pour son suivi et son soutient tout au long de mes études en France. Je pense en particulier à Anna MANETA et Cécile CHAUMIER.

La bande de copains mexicains : Tony, Manuel, Alex, Paloma, Luis Calvillo, Luis Rocha, Rosemberg, José, Héctor, Gerardo, Alejandra, Samantha, Cyril, Carmen, Bety, Raúl, Laura, Aaron, Azucena, Felipe, Toño, Humberto, Gaby et tous les autres, pour les innombrables et très agréables réunions que nous avons partagé. L'éloignement de notre pays nous a rapproché ici, de l'autre coté de l'Atlantique.

Achraf, pour avoir ouvert les portes de sa maison et de son cœur. Tu peux compter sur un chez toi au Mexique.

Joseph SIFAKIS, pour m'avoir fait l'honneur de m'accueillir au laboratoire VERIMAG les cinq premiers mois de mon séjour en France.

Raúl JACINTO MONTES, mon directeur de thèse de *master*, qui m'a motivé le premier à entreprendre cette aventure.

Israel, Lydia, Luis Fernando, Diana, Lorena et Elizabeth, mes amis de toujours.

Andrée, ma belle-mère, pour m'avoir accueillie dans sa maison, dans sa famille et dans son cœur.

Thierry, *mi cuñado* (mon beau-frère), pour sa patience pendant les longues parties d'échecs que nous pûmes avoir pendant les vacances, pour ses encouragements et sa tendresse.

Mon frère José Francisco et sa femme Avelina qui m'ont témoigné leur confiance en m'offrant le rôle de marraine de leur premier enfant, José Salvador.

Ma sœur Brenda, qui est plus que ma sœur, tu es mon amie, ma confidente, mon étoile la plus précieuse.

Stéphane, pour nos interminables discussions où l'on voulait juste comprendre et où l'on finissait par tout remettre en cause. Pour les heures que tu as passé à lire avec soin et patience ce document. Pour tes commentaires avisés qui ont beaucoup aidé à la clarté et la qualité de ce travail. Pour ta compagnie et ton soutient tout au long de cette thèse, sans toi il aurait était tellement difficile d'y arriver. Merci pour m'avoir appris à connaître et à aimer ce beau pays qu'est la France. Merci pour ta tendresse mais surtout pour ton amour.

Table des matières

Table des figures

Liste des tableaux

CHAPITRE 1

Introduction

Ce premier chapitre introduit notre travail. Nous commençons par caractériser l'informatique mobile (section 1.1), puis la problématique posée ainsi que les objectifs visés dans cette thèse (section 1.2). Nous continuons par la présentation des contributions (section 1.3) et l'organisation de ce manuscrit (section 1.4).

1.1 Caractérisation de l'informatique mobile

Depuis déjà quelques années l'informatique mobile émerge comme domaine de recherche. Les avancées de la technologie des télécommunications sans fil et l'évolution des caractéristiques des unités mobiles comme les ordinateurs portables, les PDA (*Personal Digital Assistants*), les téléphones mobiles ou les cartes à puce, contribuent à la popularité croissante et à la diversité des applications mobiles.

Un des objectifs de l'informatique mobile, en ce qui concerne la gestion de données, est de permettre aux utilisateurs mobiles d'accéder et de manipuler des données à partir de n'importe quel emplacement, à n'importe quel moment et à partir de n'importe quel type de poste (n'importe où, n'importe quand). Les données peuvent être découvertes au fur et à mesure des déplacements. L'accès se fait sur des bases de données installées sur des unités fixes ou mobiles. Nous pouvons citer, à titre d'exemple, différents cas de figure :

- dans un aéroport, un voyageur peut avoir accès à l'information concernant les arrivés/départs des vols et toute autre information liée à l'aéroport (services, boutiques, plans) à partir de son PDA ;
- un voyageur a besoin de savoir quels sont les hôtels/restaurants/musés les plus proches de son emplacement actuel ou de l'endroit dans lequel il sera prochainement ;
- un homme d'affaires, qui passe la plupart de son temps à voyager, a besoin de manipuler les données de son entreprise lors de ses déplacements à partir de son ordinateur portable ;
- un médecin a besoin d'avoir accès aux dossiers de ses patients aussi bien à l'intérieur d'un hôpital que depuis l'extérieur (chez les patients ou lors d'une intervention) ;

- les personnes faisant des travaux sur le terrain (ingénieurs, architectes, archéologues, biologistes, etc.) ont besoin eux aussi d'accéder à leurs bases de données depuis différents endroits ;
- un internaute peut vouloir accéder aux services de magasins virtuels (électroniques) à partir d'une terrasse de café, d'un parc ou lors d'une balade.

Ces différentes applications posent des challenges en matière d'accès et de gestion de bases de données. Les approches traditionnelles ne sont pas toujours directement applicables dans le contexte mobile. C'est pour cette raison que la recherche dans le domaine de l'informatique mobile connaît aujourd'hui une activité très importante.

La suite de cette section est organisée de la manière suivante. Nous commençons par montrer les caractéristiques de l'environnement mobile considéré (section 1.1.1). Ensuite, nous introduisons une classification des Systèmes de Gestion de Bases de Données (SGBD) ainsi que certains axes de recherche sur les SGBD dans l'environnement mobile (section 1.1.2).

1.1.1 Environnements mobiles

Un environnement mobile se compose d'Unités Mobiles (UM) et d'Unités Fixes (UF) (cf. figure 1.1). Lors de leurs déplacements, les UM peuvent conserver leur connexion au réseau au travers d'une interface fournie par des UF particuliers appelées Station Base (SB) ou Station de Support Mobile (SSM). La région géographique couverte par une station base est appelée cellule. Les UM communiquent avec la station base de la cellule dans laquelle elles se trouvent. Le processus consistant à passer d'une cellule à une autre en gardant la connexion est appelé *hand-off* ou *hand-over*.

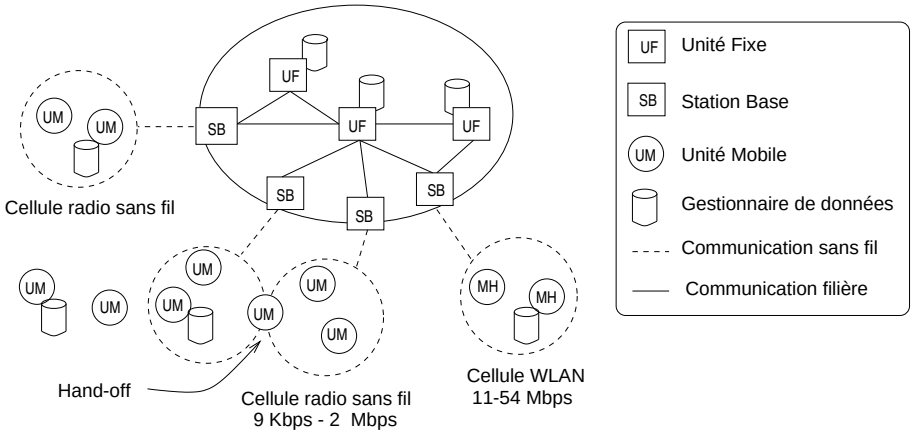

FIG. 1.1 – *L'environnement mobile*

Les UM sont généralement limitées en mémoire, en capacité de calcul, d'affichage et principalement en autonomie de batterie. Bien que l'évolution de leur caractéristiques suit

celle des UF, les UM seront toujours moins puissantes. Ces limitations sont principalement dues aux contraintes introduites pour satisfaire aux critères de portabilité (réduction du poids et de la taille). Afin de pallier les contraintes des UM, les constructeurs proposent des dispositifs faciles à ajouter/enlever (dispositifs *plug-in*) selon les besoins et le confort des utilisateurs.

Les technologies de communication sans fil sont conçues pour couvrir différents besoins. Elles varient en amplitude de couverture, débit de communications, coût d'installation/utilisation, etc. Par exemple, le prix d'une communication via un réseaux à commutation par paquets est facturé selon la quantité de données transmises, alors que pour une communication via un réseaux à commutation par circuits le nombre d'unités de temps utilisées est considéré pour la facturation. Les réseaux de type WLAN (Wireless Local Area Network) n'ont pas un prix unique de communication, les fournisseurs facturent le temps de connexion de manière arbitraire (minute, heure, journée, forfait, etc.). En ce qui concerne la bande passante, elle est très limitée en comparaison de celle des réseaux fixes qui peuvent atteindre 1 Gbps :

- Les réseaux GSM (*Global System for Mobile communication*) offrent un débit de communication théorique de 9 Kbps. Comme il s'agit d'un réseau à commutation par circuits, une fois la communication établie, le débit de communication est garanti.

- Les réseaux dits de génération 2,5 comme GPRS (*General Packet Radio Services*) et EDGE (*Enhanced Data GSM Environment*) étendent le GSM. La bande passante des réseaux de type GPRS peut varier de 56 à 114 Kbps, tandis que celui d'EDGE peut atteindre 384 Kbps.

- Pour les réseaux à commutation par paquets, le débit de communication des réseaux de troisième génération comme l'UMTS (*Universal Mobile Telephone Service*) peut varier théoriquement de 144 Kbps (lors d'un déplacement en voiture), 384 Kbps (lors d'un déplacement à pied) et jusqu'à 2 Mbps à l'intérieur d'un bâtiment.

- Avec les WLAN, le débit entre l'UM et la SB peut atteindre théoriquement 11 Mbps (norme 802.11b) lorsque l'UM est proche de la SB. Lorsque l'UM s'éloigne le débit de communication passe à 5,5 Mbps, 2 Mbps et finalement à 1 Mbps au fur et à mesure que le signal s'affaiblit et se dégrade. Le cas des réseaux WLAN de la norme 802.11g est similaire, leur bande passante peut atteindre théoriquement 54 Mbps.

- La communication par satellite peut offrir un débit théorique de 2.4 Kbps à la sortie de l'UM (*uplink*) et de 4.8 Kbps à la réception de l'UM (*downlink*).

Par ailleurs, la capacité de communication entre les UM et les SB est asymétrique. En effet, les SB n'ont pas de contraintes d'énergie et bénéficient généralement de canaux de communication à haut débit, capables de disséminer d'information à un large nombre d'UM.

Afin d'offrir aux utilisateurs une connexion continue (depuis le bureau, à l'aéroport, en centre ville, en train, en pleine mer, etc.), il est indispensable de faire cohabiter les technologies de communication. Cette cohabitation contribue à augmenter la variabilité des capacités de communication. Ainsi, passer d'un WLAN au GSM fait baisser de façon importante la bande passante. La situation est similaire pour le prix des communications.

Malgré la cohabitation des réseaux, il existe encore des zones où la communication n'est pas possible. Entrer dans ces zones entraîne la coupure de la communication (déconnexion).

Dans un contexte mobile, une déconnexion n'est pas considérée nécessairement comme une défaillance mais peut être vue comme un état normal du système. En effet, dans l'informatique mobile, de nouveaux modes d'opération doivent être supportés afin de prendre en compte la variabilité des capacités de la communication sans fil et les limitations en source d'énergie : connexion "forte" et "faible", mode "déconnecté" et mode "veille". Passer d'une connexion forte à une connexion faible, dépend de la disponibilité de la bande passante. Puisque la dégradation de la connexion peut être fréquente, travailler avec une connexion faible doit faire partie des états normaux du système mobile. Un utilisateur mobile peut décider de passer en mode déconnecté pour minimiser les dépenses de communication, de batterie ou parce que la capacité de communication est limitée. Il peut également passer à un mode veille pour économiser de l'énergie. Dans ce mode d'opération, la vitesse de l'horloge du processeur est réduite et aucun calcul n'est réalisé. L'UM revient à une opération normale lors de la réception d'un message.

Les applications mobiles doivent prendre en compte les particularités de l'environnement mobile. Ainsi, l'un des principaux défis de l'informatique mobile, en général, est de gérer au mieux les variations du contexte sans perdre pour autant le contrôle sur la qualité des services.

1.1.2 SGBD dans les environnements mobiles

Au cours des années, les systèmes de gestion de bases de données ont évolué du système centralisé au système réparti et de nos jours, du réparti au mobile. Nous considérons qu'un système mobile comprend une application avec des utilisateurs mobiles ou fixes qui accèdent aux données localisées sur des sites également mobiles ou fixes. Un système mobile peut être réparti à l'échelle d'une zone géographique comme un bâtiment, un quartier, une ville, un département, une région, un pays, etc. Un tel système peut être composé d'un nombre important de SGBD qui diffèrent par leur niveau d'autonomie et d'hétérogénéité.

Les systèmes mobiles peuvent être considérés comme une extension des systèmes répartis et multibases. En effet, la classification proposée dans [OV99] peut être étendue pour intégrer les systèmes mobiles [DH95]. L'extension se base sur les caractéristiques de l'environnement mobile. Passer du niveau réparti au niveau mobile signifie que le système comporte des unités mobiles.

Nous avons légèrement modifié les classifications de [OV99, DH95] et nous proposons la classification de la figure 1.2. Les axes d'autonomie, de distribution et d'hétérogénéité sont définis comme suit :

Autonomie. Elle indique le degré d'indépendance des composants du système. Ceci concerne, entre autres, la liberté de choisir le type de données utilisées, la manière de les manipuler, la liberté de décider le type d'information à partager avec le système global, ainsi que l'indépendance d'exécution.

Hétérogénéité. Elle fait référence à l'hétérogénéité du matériel informatique mais aussi des gestionnaires de données : langages de requêtes, gestionnaires de requêtes, gestionnaires de transactions, etc.

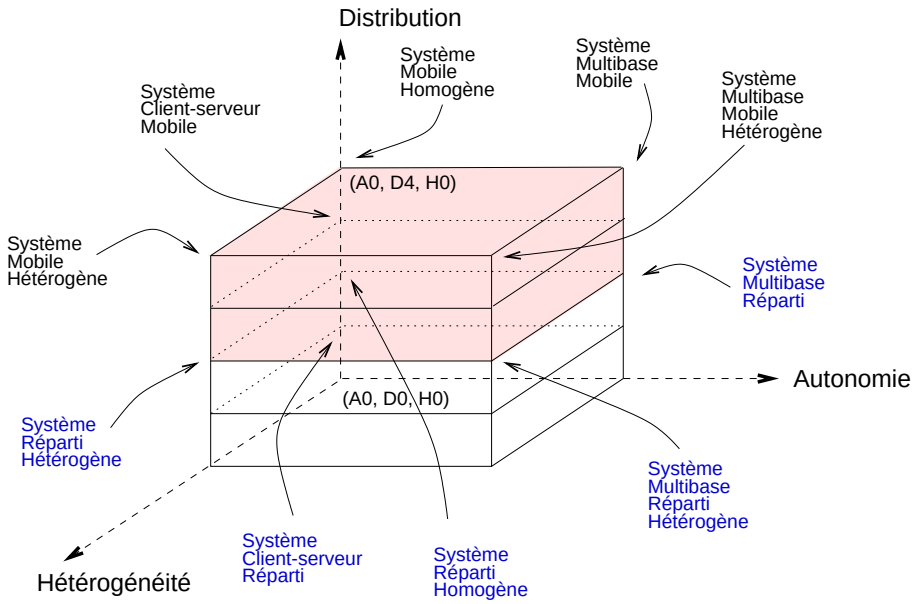

FIG. 1.2 – *Classification des systèmes bases de données*

Distribution. Elle fait référence à la distribution physique des données mais aussi aux fonctions de gestion de données. En général, trois types de distribution sont considérés : centralisé, client-serveur et pair à pair (*peer to peer*).

Dans la figure 1.2, nous présentons 4 niveaux, les deux niveaux inférieurs correspondent aux systèmes répartis tandis que les deux niveaux supérieurs (en gris) correspondent au contexte mobile.[1] L'avant dernier niveau représente les systèmes mobiles avec une distribution des données/fonctionnalités de type client-serveur (A, D3, H).[2] Pour ce type de configuration, les serveurs sont localisés sur des unités fixes et les UM sont des clients. Le dernier niveau de la classification représente une distribution plus égalitaire des sites de type pair à pair (A, D4, H). Un serveur peut être "léger" ou "fort". Les serveurs légers sont des UM comportant certaines capacités de gestion afin de pouvoir travailler de façon autonome lors des déconnexions ou des faibles capacités de communication. Les serveurs forts sont des UM ou fixes pouvant offrir leurs services à d'autres serveurs.

Plusieurs axes de recherche ont vu le jour dans le domaine des systèmes mobiles [IB94, PS98, BBOB+03]. Nous citons ci-dessous quelques uns d'entre eux :

- *La dissémination d'information à un large nombre d'utilisateurs mobiles.* Le but est de profiter de la capacité de communication des SB (ou des stations fixes particuliers) pour disséminer de l'information [AAFZ96, FZ96, PC99, DVCK99]. Dans ce contexte, nous trouvons des applications concernant la météo, le trafic routier, etc.

1. Afin de simplifier la classification, nous prenons en compte deux niveaux d'autonomie (0,1) et non pas trois (0,1,2), comme dans le schéma proposé dans [OV99].

2. (A, D, H) identifie les trois axes de la classification, A=Autonomie, D=distribution et H=Hétérogénéité.

- *La gestion de données spatio-temporelles.* Le déplacement des UM rend nécessaire leur localisation. Les données représentant la localisation des UM sont dites spatio-temporelles. L'objectif de ces données est de localiser efficacement des UM [WXCJ98, GBE+00]. Les applications bénéficiant de la localisation des UM sont celles qui utilisent des requêtes nommées "dépendantes de la localité" : retrouver l'hôtel/restaurant "le plus proche" ou retrouver les camions qui sont sur l'autoroute A41, etc.

- *L'optimisation de la gestion des données embarquées.* L'objectif est de minimiser la consommation de la mémoire et des processeurs. Ces ressources sont généralement très limitées sur les UM comme les cartes à puce [PBVB01, ABP03]. Les applications visées sont de type personnel : dossiers médicaux, d'assurance, etc.

- *La duplication/synchronisation des données.* Dans les applications de type client-serveur (professionnelles, personnelles, etc.), les données peuvent être dupliquées sur les clients mobiles à partir de serveurs fixes. Les clients peuvent manipuler les données localement (lors des déconnexions) pour ensuite les synchroniser avec les serveurs (lors des reconnexions) [VCFS00, BP98].

- *La gestion de transactions mobiles.* L'objectif est de contrôler les accès concurrents aux données sans affecter leur cohérence (propriétés ACID) en présence des contraintes de l'environnement mobile (déconnexions, bande passante faible, etc.). Les données peuvent être embarquées sur des UF et/ou sur des UM [GHPS96, PB99, LS95, WC99].

Dans notre recherche, nous nous intéressons au dernier point : les transactions dans l'environnement mobile.

1.2 Transactions mobiles : problématique et objectifs

Les transactions permettent l'accès aux bases de données d'une manière contrôlée. Elles transforment une base de données d'un état cohérent à un autre état cohérent. Dans les systèmes centralisés et répartis, l'utilisation de transactions est la façon la plus sûre de gérer les bases de données. Ceci ne fait pas exception dans les systèmes mobiles étant donné que le besoin d'accès aux données d'une manière correcte reste d'actualité.

Le but de cette section est de montrer notre problématique de recherche. La section 1.2.1 montre comment la variation de l'environnement peut affecter le déroulement des applications, plus précisément, l'exécution des transactions. Ensuite, la section 1.2.2 met en évidence la nécessité du support de différents modèles d'exécution des transactions mobiles.

1.2.1 Impact de l'environnement mobile sur les transactions

Les caractéristiques de l'environnement mobile peuvent avoir un impact négatif sur l'exécution des transactions :

- Le temps d'exécution augmente et varie à cause des ressources limitées des UM ou du débit de communication faible et variable des réseaux sans fil.

- La faiblesse de la bande passante affecte également la consommation d'énergie car l'augmentation du temps d'exécution, entraîne une augmentation de la dépense d'énergie.

- Le prix de communication est généralement élevé et peut être très variable selon la technologie de réseau sans fil utilisée et selon la politique économique du fournisseur.
- Les coupures de communication peuvent survenir à cause d'une défaillance de batterie ou lors de la sortie de la zone de couverture du réseau.

En résumé, la variabilité et les contraintes de l'environnement mobile peuvent affecter considérablement l'exécution des transactions entraînant : (1) des annulations des transactions et (2) des coûts d'exécution imprévus et variables (dépense d'énergie, prix communication, temps d'exécution, etc.).

Les annulations sont dues au fait que les gestionnaires actuels, qui assurent les propriétés ACID des transactions (*atomicité, cohérence, isolation et durabilité*) considèrent la déconnexion des UM comme une défaillance. La variation du coût d'exécution est due au fait que les gestionnaires exécutent les transactions quel que soit l'état de l'environnement mobile.

Les contributions dans le domaine des transactions mobiles se sont essentiellement concentrées sur le support des déconnexions des UM. Nous pouvons ainsi citer : [PB99, GHPS96, LH02, WC95]. La prise en compte des variations de l'environnement mobile reste un problème ouvert.

1.2.2 Support de différents modèles d'exécution des transactions mobiles

Nous considérons qu'une transaction est mobile si *au moins une UM participe à l'exécution de la transaction*. Nous définissons cinq modèles d'exécution. Les trois premiers impliquent une unique UM, les deux derniers en impliquent plusieurs :

1. exécution complète sur des unités fixes mais demandée par une UM ;
2. exécution complète sur une UM ;
3. exécution répartie entre une UM et des unités fixes ;
4. exécution répartie entre plusieurs UM ;
5. exécution répartie entre plusieurs UM et fixes.

Ces cinq modèles d'exécution couvrent toutes les possibilités qui incluent des UM/UF. Selon le modèle d'exécution, la complexité et la difficulté de mise en œuvre de la gestion des transactions mobiles varient. Lorsque la transaction est exécutée sur le réseau fixe (modèle 1), des techniques traditionnelles peuvent être utilisées [DG00, KK00, YZ94, DHB97]. Lorsqu'il s'agit d'une exécution complète sur l'UM (modèle 2), les techniques de gestion doivent chercher à optimiser l'utilisation des ressources locales de l'UM [PBVB01, Vin02] ou à assurer la cohérence des données embarquées qui sont souvent considérées comme des copies [LH02, LS95, WC99, WC95, MB01]. Lorsqu'une exécution répartie mobile doit être gérée (modèles 3, 4 et 5), en plus d'optimiser les ressources locales de l'UM, les caractéristiques de la communication sans fil doivent être prises en compte.

A l'heure actuelle, les propositions dans le domaine des transactions mobiles sont souvent définies pour des contextes particuliers (duplication, objets fragmentables, données d'agrégation, etc.). Elles s'orientent surtout vers le support des modèles d'exécution sur l'UM ou

sur le réseau fixe (modèles d'exécution 1 et 2). Le support de transactions réparties mobiles n'a pas été suffisamment développé malgré les avantages qu'il peut offrir. Une exécution répartie entre une unité mobile et des unités fixes peut être avantageuse pour l'unité mobile lorsqu'elle désire économiser ses ressources ou simplement interagir avec d'autres sites. Une exécution répartie entre plusieurs unités mobiles peut être désirée lorsqu'on veut profiter, par exemple, des services offerts par des "voisins" également mobiles. Une exécution répartie entre plusieurs unités mobiles et fixes peut être nécessaire, par exemple, dans une exécution coopérative.

Chacun des modèles d'exécution a besoin d'un "état particulier" de l'environnement mobile. Par exemple, pour exécuter une transaction sur une UM, la communication n'est pas obligatoire. Par contre, pour une exécution répartie elle est indispensable, au moins temporellement. Nous avons vu qu'il est fréquent que l'environnement mobile passe d'un état à un autre. Nous considérons, qu'il serait intéressant de supporter plus d'un modèle d'exécution afin de ne pas interrompre le déroulement des applications. A l'heure actuelle, les gestionnaires de transactions, généralement, n'utilisent pas les différents modèles d'exécution selon le contexte. Seulement, certaines propositions prévoient une exécution sur l'unité mobile ou sur le réseau fixe selon l'état de la connexion (connécté, déconnecté). Les autres modèles d'exécution et les possibles variations de l'environnement mobile (bande passante, prix de communication, ressources sur l'unité mobile, etc.) ne sont pas prises en compte.

1.2.3 Objectifs de la thèse

L'objectif de ce travail de thèse est la conception et mise en œuvre d'une approche transactionnelle adéquate pour les environnements variables comme le contexte mobile. Nous visons une approche générale qui ne cible pas d'application particulière et les composants du système sont faiblement couplés.

Les variations de l'environnement mobile peuvent affecter fortement l'exécution des transactions. Le défit dans un tel contexte est de limiter le coût d'exécution et de réduire le nombre d'annulations de transactions dues aux variations de l'environnement. L'adaptation des transactions au contexte d'exécution peut être présentée comme une solution. Nous traduisons l'adaptation par une "perception" des variations du contexte d'exécution et une "réaction" convenable du point de vue transactionnel.

Nous considérons que la capacité d'adaptation est étroitement liée à la capacité de supporter différents modèles d'exécution pour une même transaction. De même, la facilité de définir différents modèles d'exécution dépend des critères de qualité acceptés par les applications. Le critère de qualité est défini comme un compromis entre un coût d'exécution et la qualité des résultats des transactions. Par exemple, lorsque la bande passante est basse ou coûteuse, une transaction sur l'UM sera préférable, plutôt qu'une transaction répartie mobile. Ceci afin d'économiser le temps de communication, éventuellement aux dépends de la qualité des résultats. Au contraire, si le coût d'exécution n'a pas d'importance, mais qu'une bonne qualité des réponses est nécessaire, alors l'exécution de la transaction répartie mobile sera préférée.

Les cinq modèles d'exécution présentés ci-dessus sont considérés, en particulier l'exécution de transactions réparties mobiles car nous trouvons très avantageuse son utilisation dans le contexte mobile et ce modèle n'a pas été suffisamment abordé.

Par ailleurs, la gestion des transactions mobiles adaptables doit être réalisée d'une manière adéquate. En effet, les techniques utilisées doivent prendre en compte la particularité des sites composant les systèmes mobiles : autonomie, hétérogénéité, déconnexion, déplacement, faible capacité de communication, prix de communication, entre autres.

1.3 Contributions de notre travail

Notre travail a été réalisé au sein du projet NODS [Col00] dont l'objectif est de construire des systèmes de gestion de bases de données à partir de services ouverts [VS00, GB03, Dra03, Duo03]. Ce projet vise à offrir des services de granularités différentes pouvant coopérer au niveau intergiciel. Au lieu de proposer un ensemble fixe de composants logiciels offrant des services de persistance, de duplication, de requêtes, de tolérance aux pannes, etc., NODS vise à offrir une structure logicielle ouverte, adaptable, et évolutive pouvant être configurée selon les besoins et les caractéristiques des applications. Un système de bases de données est ainsi vu comme une infrastructure constituée de services coopératifs, adaptables et extensibles permettant aux applications de construire sur mesure leurs propres composants de gestion de base de données.

Dans ce cadre, notre objectif est d'offrir une approche transactionnelle qui apporte l'aspect mobile aux applications et qui puisse être intégrée dans un système NODS.

Dans la suite, nous présentons nos principales contributions. Nous commençons par l'analyse de notre état de l'art (section 1.3.1) dont les conclusions justifient nos propositions. Nous considérons important de séparer les aspects modèle des techniques de gestion transactionnelle. Ainsi, nous proposons, d'une part, un modèle de transactions mobiles adaptables (section 1.3.2) et d'autre part, les techniques de gestion transactionnelle adéquates pour implanter le modèle proposé (sections 1.3.3 et 1.3.4).

1.3.1 Analyse de l'état de l'art

L'étude de l'état de l'art se divise en deux parties. D'une part nous proposons une analyse des concepts et des techniques utilisées dans les systèmes multibases. D'autre part nous analysons les travaux proposant de nouveaux modèles ou techniques de gestion de transactions mobiles.

Nous faisons un rapprochement des systèmes mobiles avec les systèmes multibases, du fait des caractéristiques des sites mobiles : couplage faible, autonomie et souvent hétérogénéité. [3] L'autonomie est indispensable à cause des restrictions de communication. L'hétérogénéité est présente de fait de la diversité des unités mobiles et de la découverte de nouveaux serveurs pendant les déplacements.

L'objectif de la première partie de notre étude est de montrer si les techniques utilisées dans les multibases peuvent être réutilisées directement, adaptées ou s'il est nécessaire de proposer de nouvelles approches. L'analyse aborde la sérialisabilité globale ainsi que d'autres critères de correction plus souples, comme la "sérialisabilité locale" [MRKS91a] ou la "sérialisabilité à deux niveaux" [MRKS98]. De même, l'atomicité globale et les techniques de récupération sont analysées. Par ailleurs, les modèles transactionnels étendus, comme les Sagas [GMS87] ou les Flexibles [ELR90], sont également analysés.

3. Ce rapprochement n'impose pas de fonctionnalités multibases à notre contexte.

Cette partie de notre état de l'art nous permet de montrer que les modèles souples et les techniques flexibles sont applicables aux environnements mobiles. Néanmoins, ils ne peuvent pas être utilisés directement. Il est nécessaire de les adapter pour supporter, par exemple, les déconnexions de sites mobiles.

Dans la deuxième partie de notre étude, nous proposons une analyse approfondie d'un grand nombre de travaux sur les transactions mobiles [SARA04, SARA01c]. Fréquemment, ces travaux adaptent certaines techniques et certains modèles des systèmes répartis et multi-bases. L'analyse est faite selon les propriétés ACID des transactions mobiles, mais également, selon les modèles d'exécution supportés et leur façon de gérer les variations de l'environnement mobile.

Bien que le domaine de recherche des transactions mobiles présente une activité très importante depuis une dizaine d'années et que plusieurs résultats intéressants aient vu le jour, certains points restent encore ouvertes :

- malgré l'importance de la variabilité de l'état de l'environnement mobile, très souvent, seule la déconnexion est prise en compte ;
- les propositions s'orientent vers des transactions exécutées complètement sur l'UM ou sur le réseau fixe (modèles d'exécution 1 et 2), les transactions réparties mobiles étant mises à l'écart (modèles d'exécution 3, 4 et 5) ;
- l'adaptation des transactions selon le contexte n'est pas suffisamment abordé ;
- les propositions sont en général orientées vers des contextes applicatifs très particuliers ;

1.3.2 Le modèle de transactions mobiles adaptables AMT

Au vu de l'état de l'art, nous proposons un modèle de transactions mobiles adaptables nommé AMT (*Adaptable Mobile Transactions*) [SARAL03, SA02]. Le modèle AMT est suffisamment flexible pour prendre en compte les contraintes de l'environnement mobile afin de limiter les annulations et le coût d'exécution.

Nous proposons d'ajouter de l'information concernant l'infrastructure d'exécution dans la définition des transactions. En effet, le modèle AMT permet de définir des transactions mobiles composées d'un ensemble d'**alternatives d'exécution**, chacune étant associée à un état particulier du contexte. Elles peuvent utiliser l'un des cinq modèles d'exécution. Le fait d'avoir plusieurs alternatives pour l'exécution d'une transaction rend possible l'adaptation à l'environnement mobile.

Afin de mieux gérer les caractéristiques de l'environnement mobile, le modèle proposé relâche les propriétés ACID, en particulier, l'atomicité et la cohérence.

Nous spécifions formellement le modèle AMT en utilisant ACTA [CR94] qui est un formalisme basé sur la logique du premier ordre. Cette spécification formelle permet de caractériser la structure du modèle ainsi que ses propriétés. L'utilisation de ce formalisme, accepté par la communauté de recherche dans le domaine des transactions, permet de bien situer et de comparer le modèle AMT par rapport aux autres modèles proposés.

Par ailleurs, une étude analytique des transactions de type AMT en utilisant un modèle probabiliste a été réalisé [SARAL03]. Cette étude permet de donner certaines estimations de performances qui montrent le gain qu'il est possible d'obtenir avec les transactions de type AMT. Du fait de la complexité de la définition des transactions de type AMT, l'un des

objectifs de cette étude est de proposer des bases pour un outil d'aide à la conception de transactions adaptables.

Notre approche est générale mais elle s'adresse principalement aux applications qui : (1) veulent optimiser la consommation des ressources, (2) peuvent éventuellement exécuter une tâche de plusieurs manières et (3) préfèrent éventuellement attendre un état convenable de l'environnement plutôt que d'obtenir des coûts imprévus ou des annulations transactionnelles.

1.3.3 L'intergiciel TransMobi

Afin de mettre en œuvre le modèle AMT, nous avons conçu et réalisé l'intergiciel TransMobi. Il fournit les fonctionnalités et les protocoles nécessaires pour gérer les transactions définies à partir du modèle AMT. Son but est d'apporter l'aspect mobilité à la gestion transactionnelle de façon transparente pour les SGBD sous-jacents.

TransMobi se localise entre le code applicatif et les SGBD composant le système mobile [SARAL03, SA02]. Il joue un rôle d'intégrateur qui s'occupe de certaines fonctions propres aux moniteurs transactionnels. Il assure, entre autres, l'atomicité sémantique des T_{AMT} avec le protocole CO2PC (cf. section 1.3.4) et la sérialisabilité avec une adaptation de la technique OTM [GRS94].

TransMobi gère la perception de l'environnement mobile et adapte l'exécution des transactions selon les changements du contexte. Il prend en compte les caractéristiques des SGBD dans les environnements mobiles et permet leur hétérogénéité, autonomie, déconnexion, etc.

Nous avons implanté un prototype de TransMobi. Le développement s'est réalisé en utilisant PersonalJava. Le système gestionnaire de bases de données utilisé est PointBase [Poi02], le réseau sans fil est un *Wireless Local Area Network* (WLAN) IEEE 802.11b et les UM utilisées sont un ordinateur portable et un PDA de type IPAQ H3850 de Compaq.

1.3.4 Le protocole de validation CO2PC

Nous proposons CO2PC, une Combinaison d'une approche Optimiste et le protocole 2PC [Gra78]. Son but est d'assurer l'atomicité sémantique des transactions réparties mobiles. CO2PC est un protocole flexible qui permet aux sous-transactions de valider de manière anticipée (optimiste) ou coordonnée avec la transaction globale (non-optimiste).

CO2PC offre plusieurs avantages pour le contexte mobile, il : permet l'autonomie des participants, autorise leur hétérogénéité, réduit le blocage des ressources, limite le nombre de messages nécessaires pour son exécution et permet la déconnexion des participants.

1.4 Organisation du document

Cette section montre l'organisation de ce document de thèse. La figure 1.3 donne une idée de la manière de l'aborder ce manuscrit.

- Le chapitre 2 rappelle les concepts transactionnels utilisés dans les systèmes multibases de données. Ce chapitre se focalise sur les techniques pour assurer les critères de correction (sérialisabilité, quasi-sérialisabilité, etc.) et l'atomicité des transactions. Il fait également un rappel de quelques modèles de transactions étendues. L'objectif est

d'analyser les avantages et les inconvénients de ces concepts lors de leur utilisation dans l'environnement mobile.

- Le chapitre 3 offre une analyse approfondie d'un nombre important de travaux de recherche qui proposent des solutions autour des transactions mobiles. L'analyse est faite : (1) en fonction des propriétés ACID et (2) en fonction de la manière de supporter la mobilité et la déconnexion des utilisateurs mobiles.

- Le chapitre 4 introduit le modèle de transactions mobiles adaptables (AMT). Tout d'abord, le modèle et ses propriétés sont présentés de façon intuitive, puis d'une façon formelle en utilisant ACTA.

- Le chapitre 5 présente une étude analytique du modèle AMT. Cette étude donne une idée du gain en performance qui peut être obtenu en utilisant le modèle de transactions AMT. Cette étude permet également d'aider à la conception des transactions mobiles de type AMT. En particulier, les alternatives d'exécution composant une transaction AMT peuvent être analysées et choisies d'après le calcul des coûts d'exécution.

- Le chapitre 6 présente l'intergiciel TransMobi. TransMobi met en œuvre les protocoles appropriés pour garantir les propriétés du modèle AMT. En particulier, ce chapitre introduit CO2PC, le protocole que nous proposons pour assurer l'atomicité sémantique des transactions réparties mobiles.

- Le chapitre 7 conclut cette thèse et présente quelques perspectives de recherche.

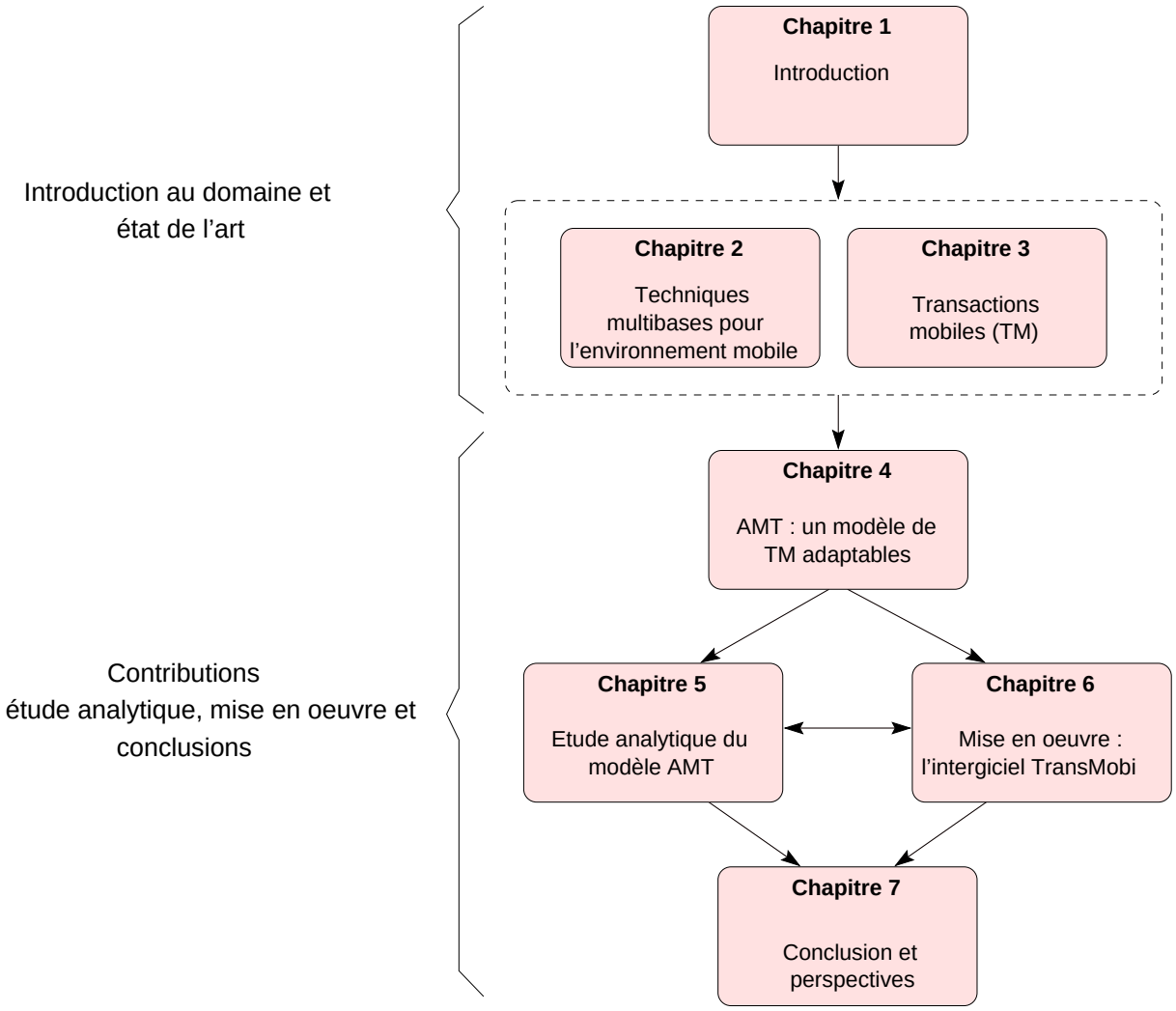

FIG. 1.3 – *Organisation du document.*

CHAPITRE 2

Techniques multibases et les environnements mobiles

L'objectif de ce chapitre est d'analyser la possibilité d'utilisation, dans l'environnement mobile, des techniques de gestion et des modèles transactionnels proposés pour les systèmes multibases.

Le chapitre est organisé de la manière suivante. Nous commençons par caractériser les SGBD dans l'environnement mobile (section 2.1). Ensuite, nous nous concentrons sur les techniques transactionnelles des systèmes multibases (section 2.2) ainsi que sur quelques modèles de transactions étendues afin d'analyser leur possible application aux systèmes mobiles (section 2.3). Nous terminons par quelques conclusions (section 2.4).

2.1 Caractérisation des systèmes mobiles

Dans cette section, nous montrons que les caractéristiques des systèmes mobiles permettent de faire un rapprochement avec les systèmes multibases de données (section 2.1.1). Ensuite, nous caractérisons les transactions mobiles (section 2.1.2).

2.1.1 Rapprochement des systèmes mobiles et multibases

Un système mobile peut être réparti à l'échelle d'une zone géographique comme un bâtiment, un quartier, une ville, un département, une région, un pays, etc. Il est composé de SGBD (serveurs) pouvant être localisés sur des UM. Les utilisateurs ou clients peuvent utiliser également des UM pour accéder aux applications offertes par le système.

Dans la figure 2.1, nous montrons une version simplifiée de la figure 1.2 où nous nous concentrons uniquement sur les systèmes mobiles. Dans la suite, nous analysons chacune des architectures illustrées.

SGBD client-serveur mobile homogène (A0, D3, H0) : ce type de système est composé de clients et serveurs. Ils sont homogènes et ne bénéficient d'aucune autonomie.

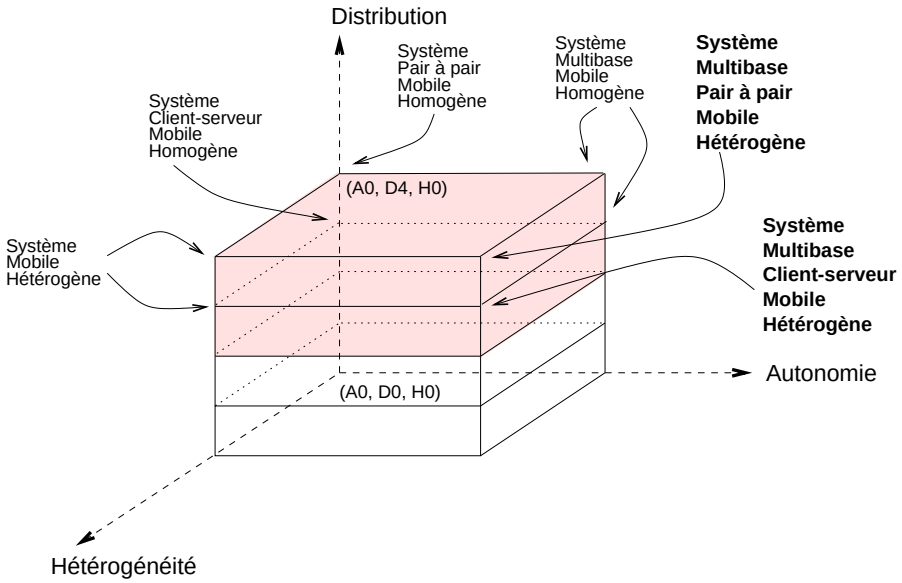

FIG. 2.1 – *Les systèmes de bases de données mobiles*

SGBD pair à pair mobile homogène (A0, D4, H0) : ce cas est similaire au dernier, sauf que les composants du système ont les mêmes capacités. La distinction entre client et serveur n'existe pas.

SGBD mobile hétérogène (A0, D3, H1) et (A0, D4, H1) : ici, le système peut être constitué de SGBD hétérogènes qui coopèrent ouvertement du fait du faible niveau d'autonomie. Ces deux types de systèmes sont plus complexes à cause de leur hétérogénéité.

SGBD multibase mobile homogène (A1, D3, H0) et (A1, D4, H0) : ces deux types de systèmes bénéficient d'autonomie, cependant leurs composants doivent être homogènes. Ils doivent fournir les mêmes interfaces, avoir le même type de données et utiliser le même type de gestionnaire.

SGBD multibase mobile hétérogène client-serveur (A1, D3, H1) : ce type de système est composé de clients et serveurs. Les serveurs bénéficient d'autonomie et ils sont hétérogènes.

SGBD multibase mobile hétérogène pair à pair (A1, D4, H1) : ce type de système est le plus complexe, les composants ont les mêmes capacités, ils préservent leur autonomie et ils sont hétérogènes.

Nous considérons qu'un système mobile correspond à l'un des deux derniers cas (en gras dans la figure 2.1). La nature dynamique du système suggère une diversité dans le niveau d'autonomie et d'hétérogénéité des SGBD. Un client mobile peut se connecter et se

déconnecter à partir de différents emplacements et les accès peuvent se faire sur des serveurs connus ou nouveaux, découverts lors des déplacements. En plus, les sites peuvent être ajoutés ou enlevés selon leur besoin. Ainsi, du fait des caractéristiques de l'environnement mobile :

- l'hétérogénéité concerne aussi bien le type d'UM (PDA, laptop,...) que le type de données et de gestionnaires.
- l'autonomie est nécessaire car, d'une part, lors des déconnexions, les UM doivent être capables de continuer leurs activités de manipulation de données, de manière indépendante. D'autre part, les limitations dans la capacité de communication amène à restreindre les échanges d'information, et ainsi le partage avec les autres composants du système.

Du point de vue architecture, un système mobile peut avoir une configuration qui peut varier de client-serveur à pair à pair. En effet, l'architecture peut présenter différentes configurations selon l'état du contexte. Dans les architectures classiques, le rôle des différentes parties est défini de façon statique [JHE99]. En l'absence de défaillances, la localité et l'état de la connexion des parties (des clients ou des serveurs) ne souffre pas de grands changements. Néanmoins, dans l'environnement mobile, la distinction entre les parties peut être floue et variable. En effet, selon le contexte, les UM peuvent être (dynamiquement) des clients légers ou des serveurs. Dans la figure 2.1 ceci se traduit par passer du niveau (A, D3, H) au niveau (A, D4, H). Dans la suite, nous présentons différentes configurations.

Client léger. Dans ce cas, les clients ont besoin des serveurs pour exécuter leurs opérations. Cette architecture est particulièrement appropriée pour les terminaux à ressources limitées. Le serveur est chargé de tous les calculs tandis que les clients affichent seulement du texte, des graphiques, jouent de l'audio, de la vidéo compressée, capturent quelques informations à l'aide d'un stylo, etc. Les clients légers sont souvent des PDA ou des téléphones portables.

Client fort. Dans l'environnement mobile les clients peuvent être forcés de travailler de manière autonome à cause des : déconnexions, des connexions faibles ou du prix de communication élevé. Les clients forts émulent les fonctions des serveurs pour exécuter les applications sans avoir à être fortement connectés aux serveurs. Généralement, les clients forts sont des ordinateurs portables avec des ressources suffisantes pour exécuter les applications.

Client-serveur flexible. Cette configuration généralise les architectures des clients légers et forts. En effet, dans cette architecture les rôles des clients et des serveurs peuvent être échangés dynamiquement. Afin d'augmenter les performances et la disponibilité des applications, la distinction entre les clients et les serveurs peut être temporairement floue. Dans cette architecture, les clients peuvent être des ordinateurs portables.

Pair à pair. Cette configuration ne fait pas la différence entre les rôles des parties. Les sites doivent posséder suffisamment de ressources pour pouvoir servir à d'autres sites.

Par ailleurs, dans le domaine de l'informatique mobile, il est courant de trouver des configuration à trois tiers qui incluent un agent ou *proxy* entre les serveurs ou entre les clients

et serveurs. Cette configuration introduit un agent localisé sur le réseau fixe. Les agents sont utilisés pour une grande variété d'activités. Ils peuvent, entre autres, jouer le rôle de représentant d'une ou de plusieurs UM (lorsqu'ils sont déconnectés) ou être attachés à un service ou à une application particulière. Les clients peuvent varier de léger à fort.

En résumé, nous considérons qu'un système mobile s'approche des multibases hétérogènes, du fait des caractéristiques qu'il présente : hétérogénéité, autonomie et distribution des données/fonctionnalités. Dans ce contexte, notre travail s'intéresse plus particulièrement à la gestion des transactions mobiles.

2.1.2 Caractérisation des transactions mobiles

Dans cette section, nous identifions les caractéristiques des transactions mobiles afin de savoir ce que les techniques (ou protocoles) gérant les transactions mobiles doivent prendre en compte. Nous identifions, également, les différents modèles d'exécution d'une transaction mobile.

Comme il n'existe pas de consensus sur la notion de transaction mobile, nous proposons la définition générale suivante :

Définition 2.1 *Une transaction mobile (TM) est une transaction où au moins une unité mobile participe à son exécution.*

La participation d'une UM dans une transaction entraîne des particularités qui nous appelons risques. Nous considérons que les risques doivent être limités ou maîtrisés. Le tableau 2.1 montre les mesures que nous considérons qu'il faut prendre selon le risque entraîné.

Risques	**Recommandations**
Annulations et coûts d'exécution imprévus : temps, dépense de batterie, prix de communication, etc.	Prendre en compte la variabilité de l'EM (en particulière la déconnexion), réduire le nombre et la taille des messages, prendre en compte les ressources limités des UM.
Le déplacement des UM ajoute de la complexité à la gestion des TM.	Contrôle de déplacement réparti ou centralisé, participation d'unités fixes (SB).
Potentiellement, transactions de longue durée et blocage du système.	Utilisation des techniques non-bloquantes comme les optimistes.

TAB. 2.1 – *Caractérisation des transactions mobiles.*

Du fait de la participation des UM dans l'exécution des TM, le risque d'annulation est important à cause des déconnexions. Afin de limiter le risque d'annulation, les déconnexions doivent être prises en compte pendant l'exécution d'une TM.

Egalement, l'utilisation des UM ainsi que la communication sans fil peuvent entraîner des coûts d'exécution imprévus (cf. section 1.2.1). Nous considérons qu'afin de faire face à ces inconvénients, il est nécessaire de prendre en compte les variations d'état de l'environnement

mobile (EM) et les limitations des ressources des UM. Dans le même sens, les protocoles assurant les propriétés des TM doivent réduire le nombre et la taille des messages nécessaires.

Le déplacement des UM ajoute de la complexité à la gestion des TM. En effet, il est nécessaire de "localiser" et "suivre" les UM. Ceci peut se faire de manière centralisé ou répartie, à partir d'une unité fixe qui peut être une SB. [1]

Par ailleurs, même si l'exécution des opérations d'une TM ne demande pas beaucoup de temps, les déconnexions rendent la TM de longue durée. Les transactions de longue durée peuvent entraîner le blocage du système. Afin d'éviter ceci, des techniques de gestion transactionnelle non-bloquantes, comme les techniques optimistes doivent être utilisées.

Modèles d'exécution des transactions mobiles

Selon la distribution des activités, nous identifions 5 modèles d'exécution différents pour une transaction mobile :

1. exécution complète sur des unités fixes mais demandée par une UM ;
2. exécution complète sur une UM ;
3. exécution répartie entre une UM et des unités fixes ;
4. exécution répartie entre plusieurs UM ;
5. exécution répartie entre plusieurs UM et unités fixes.

Dans la suite, nous introduisons ces modèles d'exécution plus en détail.

1. Exécution complète sur le réseau fixe. La transaction est déclenchée par une UM mais complètement exécutée sur des UF. C'est l'approche classique de *query shipping* où le serveur exécute des mises à jour ou des requêtes et envoie les résultats aux clients. Ce scénario est approprié, par exemple, pour des requêtes dépendantes de la localité (demande des hôtels localisés à 5 Km de l'emplacement actuel de l'UM) et des mises à jour (réservation dans l'un de ces hôtels [DK98b]). Ce type d'exécution convient aussi à l'exécution de transactions sur de grands ensembles de données.

2. Exécution complète sur une UM. Dans ce cas, la transaction est déclenchée et exécutée sur l'UM. Les données nécessaires doivent être embarquées et l'UM doit avoir les capacités suffisantes pour exécuter la transaction. L'autonomie de l'UM lui permet de continuer ses activités même si la connexion avec le serveur n'est pas possible. Des techniques de réconciliation sont indispensables pour intégrer sur le serveur, le travail fait sur l'UM. Par exemple, un vendeur possède dans son UM l'information sur les produits qu'il vend (stock, prix, caractéristiques, etc.). Il fait des ventes localement et les intègre sur le serveur de l'entreprise périodiquement.

1. Généralement, les fournisseurs des réseaux sans fil n'autorisent pas l'installation de logiciel sur des SB, cependant, nous pensons que ceci peut changer dans le future.

3. Exécution répartie entre une UM et le réseau fixe. Ce modèle est très flexible car il permet une exécution répartie entre l'UM et les serveurs du réseau fixe. Cette répartition peut être motivée par des raisons de disponibilité de ressources (données, énergie, etc.) ou de performances. Afin de limiter la communication avec le serveur pendant l'exécution de transactions, l'UM exécute certaines fonctions. Dans le scénario du vendeur présenté dans le modèle d'exécution précédent, le vendeur peut faire une vente sans avoir besoin d'établir une communication avec le serveur. Cependant, la procédure de paiement peut requérir une communication avec le serveur d'une banque pour vérifier le crédit du client. Ainsi, la vente est faite par une transaction répartie ayant une sous-transaction sur l'UM et une autre sur le serveur de la banque.

4. Exécution répartie entre plusieurs UM. Ce cas est très ambitieux et difficile, mais intéressant. L'objectif est de fournir une approche pair à pair. Une UM joue le rôle de serveur pour d'autres UM. L'idée est que suivant la localisation des UM, il peut être intéressant de demander à un "voisin" de remplir le rôle de serveur de données ou de services. Ici, voisin signifie "proche" en terme de communication. Comme exemple, nous pouvons considérer deux vendeurs qui travaillent dans la même zone géographique et qui ont besoin de partager des données d'une manière coopérative sans avoir à communiquer avec le serveur.

Pour supporter ce modèle d'exécution, des fonctions particulières sont nécessaires afin que les UM se connaissent. La station base peut jouer un rôle important. Elle peut garder des catalogues qui permettent aux UM de connaître les services disponibles dans leur zone géographique.

Dans les réseaux *ad hoc* [FJL00, IET], les UM interagissent avec une connexion point à point sans utiliser une infrastructure centrale comme une station base. Le modèle d'exécution répartie entre plusieurs UM est particulièrement approprié aux configurations dynamiques des réseaux.

5. Exécution répartie entre plusieurs UM et UF. Ce scénario est complètement réparti, l'exécution de la transaction est répartie entre plusieurs UM et plusieurs UF. Cette approche est la généralisation du modèle précèdent. Il peut être utilisé pour le travail coopératif et les multibases de données.

Le commerce électronique est un domaine d'application prometteur où des UM peuvent faire des transactions entre elles (modèle d'exécution 4) et/ou avec des UF (modèle d'exécution 5) [PA02]. Nous pouvons imaginer une foire d'affaires où des fournisseurs, manufacturiers, détaillants, clients, etc. se donnent rendez-vous. Chaque participant possède une ou plusieurs UM ou UF. Ainsi, les clients peuvent consulter des catalogues mis à disposition par les fournisseurs/manufacturiers/détaillants et acheter sur place. Pour satisfaire la demande, les fournisseurs peuvent communiquer entre eux et faire des échanges. Avec le support transactionnel approprié toutes les opérations peuvent être faites sur place et répercutées plus tard sur les serveurs des entreprises.

Selon le modèle d'exécution, la gestion des transactions mobiles varie en complexité. Lorsque la transaction est complètement exécutée sur le réseau fixe (modèle 1), des techniques traditionnelles peuvent être utilisées. Lorsqu'il s'agit d'une exécution complète sur l'UM (modèle 2), les techniques de gestion doivent chercher à optimiser les ressources locales (mémoire, batterie, capacité de calcul, etc.) des UM. Lorsqu'une exécution répartie mobile

(modèles 3, 4 et 5) doit être gérée, en plus d'optimiser les ressources locales des UM, les caractéristiques de la communication sans fil doivent être prises en compte.

Dans les sections qui suivent, nous analysons les techniques traditionnelles utilisées pour la gestion de transactions multibases, qui par définition sont réparties. Nous présentons les avantages et inconvénients de leur utilisation dans l'exécution de transactions réparties mobiles. Nous nous concentrons sur les approches multibases de données, du fait que nous les considérons plus appropriées pour l'environnement mobile (cf. section 2.1.1).

2.2 Techniques des transactions multibases

Un système multibases gère des "transactions globales" (TG) qui ont accès aux données localisées sur des SGBD généralement autonomes. Dans le contexte mobile, les SGBD sont localisés sur des UM ou des unités fixes. Chaque SGBD utilise un gestionnaire de transactions indépendant qui exécute des transactions locales (tl) et participe à l'exécution des transactions globales. Les SGBD composant le système multibases sont appelés "SGBD locaux". Le gestionnaire des transactions globales est appelé "gestionnaire global".

- Une transaction globale TG_i est composée d'un ensemble de "sous-transactions" tg_{ij}. Chacune des sous-transaction est exécutée sur un SGBD différent.
- Les transactions locales tl_i sont exécutées sur un seul SGBD, hors de la portée du système multibases.

L'ordonnancement des transactions sur un SGBD (appelé ordonnancement local) est composé des transactions locales et des sous-transactions qui y sont exécutées. Deux transactions t et t' sont en conflit direct si toutes deux accèdent à une même donnée et que l'un des accès est une écriture. Les transactions en conflit doivent suivre une **relation d'ordre**. Ainsi, si $<$ est une relation d'ordre, $t < t'$ signifie que t s'exécute avant t'.

Dans la suite, nous allons analyser certaines techniques pour assurer un ordre d'exécution globale (section 2.2.1) ainsi que pour assurer la validation et la reprise après pannes des transactions globales (section 2.2.2). Le chapitre 6 du livre [BCF+97] ainsi que [BGMS92] nous ont aidé à la réalisation de cette étude.

2.2.1 Sérialisabilite globale

La sérialisabilité est la théorie de correction de référence pour l'exécution concurrente de transactions dans les bases de données [BHG87]. Dans les systèmes multibases, la sérialisabilité est étendue à la **sérialisabilité globale**. Ainsi, un ordonnancement est globalement sérialisable si l'ordre d'exécution des sous-transactions suit le même ordre sérialisable dans chaque SGBD local ainsi que globalement. En effet, l'union des ordonnancements des SGBD locaux compose l'ordonnancement global. Un ordonnancement global est sérialisable si son graphe de sérialisabilité global est acyclique [BS88].

A cause des caractéristiques des systèmes multibases, assurer la sérialisabilité globale peut s'avérer très compliqué. L'un des problèmes des systèmes autonomes est le fait que les SGBD participant dans le système, en général, ne partagent pas l'information concernant leur gestion locale. Les ordonnancements locaux sont hors de la portée du contrôle global. Seule la demande d'exécution des sous-transactions est sous contrôle global.

L'une des conséquences de ces caractéristiques est la probabilité d'avoir des conflits indirects (inconnus du gestionnaire global). Un conflit indirect entre deux transactions globales peut avoir lieu lorsqu'il existe un ordonnancement local où une sous-transaction tg_{ii} a une relation d'ordre avec une transaction locale tl_j, laquelle a une relation d'ordre avec une sous-transaction tg_{ji}. Puisque tg_{ii} et tg_{ji} ne sont pas en conflit direct, l'ordonnanceur global ne peut pas s'apercevoir de la relation d'ordre entraînée par tl_j.

Par exemple, considérons deux transactions globales TG_i et TG_j exécutés de manière séquentielle où TG_i a accès aux données a et b, TG_j aux données a et c:

$$TG_i = lecture_{tg_{ii}}(a), ecriture_{tg_{ij}}(b)$$

$$TG_j = ecriture_{tg_{ji}}(a), lecture_{tg_{jj}}(c)$$

Considérons également deux sites, site1 et site2, dont le site1 exécute la transaction locale:

$$tl_k = lecture_{tl_{ki}}(b), ecriture_{tl_{kj}}(c)$$

L'ordonnancement sur les deux sites peut être:

$$site1 = lecture_{tg_{ii}}(a), validation_{tg_{ii}}, ecriture_{tg_{ji}}(a), validation_{tg_{ji}}$$

$$site2 = lecture_{tl_{ki}}(b), ecriture_{tg_{ij}}(b), validation_{tg_{ij}}, lecture_{tg_{jj}}(c), validation_{tg_{jj}},$$
$$ecriture_{tl_{kj}}(c), validation_{tl_k}$$

Puisque les sites sont autonomes, ils peuvent utiliser différents protocoles de contrôle de concurrence. Considérons que le site1 exécute les transactions sequentiellement et le site2 utilise l'approche par certification. Dans la certification, les dépendances de conflit écriture-écriture et lecture-écriture suivent naturellement cet ordre puisque les écritures dans la base de données sont différées au moment de la validation.

Ainsi, le site1 génère l'ordonnancement $TG_i < TG_j$. Par contre, malgré le fait que les transactions globales sont exécutées et validées de manière séquentielle, le site2 génère l'ordonnancement contraire. En effet, la transaction locale tl_k génère un conflit indirect ce qui donne l'ordonnancement $TG_j < tl_k < TG_i$.

Les conflits indirects peuvent introduire des cycles dans la relation d'ordre des transactions globales, ce qui affecte la sérialisabilité globale. Afin de faire face à ce problème, de nouveaux critères de correction et techniques de gestion ont été proposés dans les systèmes multibases [BGMS92]. Dans cette section, nous analysons plusieurs de ces techniques. Toute d'abord celles qui assurent la sérialisabilité globale et ansuite de techniques qui la relâchent.

Techniques pour assurer la sérialisabilité globale

Afin d'assurer la sérialisabilité globale, deux conditions doivent être remplies: (1) les SGBD locaux doivent produire des ordonnancements sérialisables et (2) chaque SGBD maintient l'ordre relatif des sous-transactions déterminé au niveau global.

Dans cette partie, nous introduisons différentes techniques qui assurent la sérialisabilité globale. Elles sont organisées selon le type d'ordonnancement qu'elles fournissent. Les ordonnancements vont du plus général au plus restrictif: sérialisable, (section 2.2.1.1), fortement sérialisable, (section 2.2.1.2), à point de sérialisation, (section 2.2.1.3) et rigoureux (section

2.2.1.4). Notre but est de savoir quels sont les avantages/inconvénients des techniques qui fournissent les différents ordonnancements, lorsqu'elles sont appliquées au contexte mobile.

2.2.1.1 Ordonnancements sérialisables

Un SGBD est opaque lorsqu'il ne laisse pas voir l'information sur les protocoles transactionnels qu'il utilise. La seule hypothèse qui peut être faite est que ces SGBD doivent garantir un ordonnancement local sérialisable et sans interblocage.

La méthode OTM (*Optimistic Ticket Method*) [GRS94, GRS91] propose une solution pour garantir la sérialisabilité globale dans les systèmes à SGBD opaques, qui permet l'hétérogénéité dans l'utilisation des protocoles de concurrence dans chaque SGBD.

OTM introduit des conflits directs entre les transactions globales : chacun des sites accédés contient une donnée qui doit être mise à jour (incrémentée) par toutes les sous-transactions. Ainsi, l'accès au ticket dans chaque site indique l'ordre relatif entre les sous-transactions. Cela permet au gestionnaire global d'assurer un ordonnancement globalement sérialisable. Le gestionnaire global gère un graphe qui représente la sérialisabilité globale. Les transactions globales sont les nœuds du graphe. Les arcs représentent l'ordre d'accès au ticket sur chaque site. Lorsqu'il existe un cycle dans le graphe, le gestionnaire demande l'annulation d'une transaction globale faisant partie du cycle.

Afin de fournir la sérialisabilité globale, cette méthode demande à ce que les SGBD participants génèrent des ordonnancements locaux sérialisables et recouvrables. De plus, la validation des sous-transactions doit attendre celle de la transaction globale. Pour ce faire, les SGBD doivent fournir (ou simuler) l'état de préparation du protocole 2PC.

Selon le protocole de concurrence utilisé, l'accès au ticket peut être optimisé. Par exemple, si un SGBD utilise 2PL, le plus convenable est que l'opération pour incrémenter le ticket soit la dernière de la sous-transaction. Cela, afin de réduire le temps d'attente d'autres sous-transactions concurrentes. Si l'approche utilisée par le SGBD sous-jacent est à base d'estampilles, le meilleur moment pour accéder le ticket est lorsque la sous-transaction démarre. Enfin, si le contrôle de concurrence est par certification (*Optimistic Concurrency Control*) [KR81], il n'y a pas un meilleur moment pour incrémenter le ticket cela peut être fait à n'importe quel moment.

Nous considérons que OTM est une technique adéquate pour l'environnement mobile, elle préserve l'hétérogénéité des SGBD locaux et la gestion des tickets semble être une technique facile à implanter. Le fait de demander aux SGBD d'introduire et de gérer les tickets peut être perçu comme une violation de leur autonomie. Cependant, nous pensons que cette gestion est une charge minimale pour les SGBD.

L'inconvénient de cette approche est que la génération de conflits peut entraîner des cycles dans le graphe de sérialisabilité globale, et en conséquence, des abandons là où il n'y a pas forcement de problème. De plus, les ressources des transactions sont bloquées jusqu'à la validation globale.

2.2.1.2 Ordonnancements fortement sérialisables

Un ordonnancement est "fortement sérialisable" si il assure que pour toute paire de transactions t et t', si la dernière opération de t est exécutée avant la première opération de t', alors t précède t' dans l'ordonnancement.

La plupart des protocoles de contrôle de concurrence génèrent des ordonnancements fortement sérialisables. Si tous les sites participant dans un système multibases sont des SGBD fortement sérialisables, alors, une exécution séquentielle des transactions globales est suffisante pour garantir la sérialisabilité globale.

Cette approche limite fortement les performances et appliquée à l'environnement mobile elle peut entraîner des périodes d'attente indéfinies dans le système. L'exécution séquentielle des transactions réparties mobiles n'est pas envisageable. Tous les sites devraient être disponibles en permanence, autrement le système peut rester bloqué indéfiniment.

Afin d'obtenir de meilleures performances, des modifications à cette approche sont proposées. Par exemple, dans [AGMS87] est proposé le verrouillage des sites au niveau global. En effet, avant le démarrage d'une transaction globale, celle-ci verrouille les sites qu'elle va accéder. Après la validation globale, les verrous sont libérés. Cette technique évite la création de cycle et permet l'exécution concurrente de transactions globales qui n'accèdent pas les mêmes sites.

Une approche similaire, l'algorithme *site graph* [BS88], propose la construction d'un graphe biparti. Il existe un nœud par transaction active et par site. Lorsqu'une transaction veut démarrer, son nœud est créé. Puis, on insère un arc entre le nœud crée et les nœuds site que la transaction va accéder. Si un cycle est généré, la transaction qui veut démarrer est suspendue jusqu'à la disparition du cycle.

Une autre approche propose le verrouillage altruiste [AGMS87, KS87]. L'idée est d'exécuter les sous-transactions séquentiellement sur tous les sites. Par exemple, si TG_i accède les sites $s1, s2, s3$, dans cet ordre, et si une TG_j accède aux sites $s2, s3$, alors TG_j accède au site $s2$ après TG_i, ensuite le site $s3$ également après TG_i. Ainsi, même si les deux transactions s'exécutent de manière concurrente, elles n'introduisent pas de cycle dans le graphe de sérialisabilité globale.

Toutes ces optimisations de l'exécution séquentielle des transactions globales ne conviennent pas pour l'environnement mobile. Ce sont des approches pessimistes qui préviennent la création des cycles dans le graphe de sérialisabilité globale. Pour les mettre en œuvre, il est nécessaire que tous les sites soient disponibles plus ou moins en permanence. Ceci est une contrainte que l'environnement mobile ne peut garantir.

2.2.1.3 Ordonnancements à point de sérialisation

Un point de sérialisation ps_i est une opération particulière d'une transaction t_i [Pu88]. Un ordonnancement est à point de sérialisation si il assure que pour toute paire de transactions t_i, t_j, si ps_i est exécuté avant ps_j, alors t_i précède obligatoirement t_j dans l'ordre de sérialisabilité.

Les ordonnancements à point de sérialisation sont un sous ensemble des ordonnancements fortement sérialisables. La différence est que les premiers permettent une meilleure concurrence que les deuxièmes. En effet, les transactions globales peuvent s'exécuter en parallèle, seul les points à sérialisation doivent être exécutés dans le même ordre sur tous les sites.

Un cas particulier du point de sérialisation est la validation d'une transaction. Dans ce cas, l'ordonnancement doit correspondre à l'ordre de validation des transactions. Les ordonnancements basés sur la validation sont appelés "ordonnancements fortement recouvrables". Ces ordonnancements sont un sous ensemble des ordonnancements recouvrables [BHG87]. La différence entre les ordonnancements recouvrables et les fortement recouvrables est que dans les premiers, l'ordre de validation est défini par les conflits d'écriture suivi de lecture (si t_j lit à partir de t_i, alors t_i valide avant t_j). Par contre, dans les ordonnancements fortement recouvrables, tous les conflits directs établissent l'ordre de validation. C'est-à-dire, aussi bien les conflits (1) d'écriture suivi de lecture, (2) d'écriture suivi d'écriture, que (3) de lecture suivi d'écriture

Dans les ordonnancements fortement recouvrables, seul la validation des sous-transactions d'une transaction globale doit être faite séquentiellement.

Le point de sérialisation peut être différent dans chaque protocole de contrôle de concurrence [MRB+92]. L'hétérogénéité concernant les techniques de contrôle de concurrence utilisées par les SGBD est permise. Néanmoins, le gestionnaire global doit avoir connaissance des points de sérialisation afin de pouvoir les identifier. Ceci affecte l'autonomie des sites.

Malgré l'optimisation du point de vue de la concurrence, le problème pour obtenir ces ordonnancements dans un environnement mobile reste le même que dans la section précédente. Le gestionnaire global doit assurer l'exécution séquentielle des points de sérialisation des transactions globales sur chacun des sites. Dans une exécution séquentielle, tous les sites doivent être disponibles à tout moment, autrement, le système risque d'être bloqué indéfiniment. En effet, l'attente au niveau global des transactions dépend de la disponibilité des sites mobiles et fixes. Cette attente peut devenir importante lors de la déconnexion des sites mobiles.

2.2.1.4 Ordonnancements rigoureux

Un ordonnancement est rigoureux si la dernière opération de t_i précède la première opération de t_j alors t_i précède t_j dans l'ordonnancement.

Le protocole 2PL rigoureux[2] [BHG87, EGLT76] fait partie des protocoles pouvant générer des ordonnancements rigoureux. D'autres protocoles comme l'estampillage peuvent produire des ordonnancements rigoureux avec quelques modifications.

Avoir des ordonnancements rigoureux sur tous les sites ne garantit pas un ordonnancement global rigoureux. Il faut que le gestionnaire global coordonne la validation des sous-transactions. L'idée est que les sous-transactions d'une transaction globale valident seulement si toutes leurs opérations sont terminées. Le retardement de la validation augmente la probabilité d'interblocage dans cette technique.

Le blocage des ressources utilisées par les TG jusqu'à la validation globale, limite fortement la disponibilité des données surtout lorsqu'il s'agit de transactions réparties mobiles. La présence de possibles déconnexions pour un temps non-borné rend cette approche peu intéressante pour l'environnement mobile.

De plus, l'hétérogénéité est remise en cause si l'on impose aux SGBD d'utiliser le même protocole de control de concurrence, par exemple 2PL rigoureux. Comme la majorité des SGBD commerciaux implantent 2PL rigoureux (lorsqu'ils offrent le niveau d'isolation "sé-

2. Dans 2PL rigoureux tous les verrous sont libérés après la terminaison de la transaction.

rialisable"), ceci n'est pas une vraie contrainte. Cependant, ceci n'est pas le cas des SGBD commerciaux légers [BBPV02, Vin02].

Techniques pour relâcher la sérialisabilité globale

D'autres techniques moins contraignantes ont été proposées. Elles relâchent la sérialisabilité globale tout en préservant une cohérence acceptable pour certaines applications.

Dans cette partie, nous analysons tout d'abord la sérialisabilité locale (section 2.2.1.5), puis la sérialisabilité à deux niveaux (section 2.2.1.6) et l'epsilon-sérialisabilité (section 2.2.1.7).

2.2.1.5 Sérialisabilité locale

S'il n'existe pas de contrôle global dans le système et si tous les SGBD assurent la sérialisabilité, alors une "sérialisabilité locale" est garantie. Autrement dit, un ordonnancement global est localement sérialisable (LSR) si tous les sites garantissent un ordonnancement local sérialisable.

Une façon pour garantir des ordonnancements globaux LSR est d'interdire les contraintes d'intégrité globales.[3] Dans [MRKS91a], il est montré que si toutes les transactions globales préservent les contraintes d'intégrité locales, quelque soit l'état des données qu'elles lisent sur les autres sites, alors une exécution LSR est considérée correcte.

L'approche appelée quasi-sérialisabilité proposée dans [DE89], en plus d'interdire de contraintes d'intégrité globales, interdit les dépendances de valeur.[4] En effet, les sous-transactions d'une transaction globale ne doivent pas avoir de dépendance de valeur entre elles. Les considérations faites dans quasi-sérialisabilité permettent d'ignorer les conflits indirects. Ainsi, lorsqu'il n'y a aucune relation entre les SGBD, alors le système multibases est LSR.

Ces techniques flexibles semblent être adaptées pour l'environnement mobile. Elles préservent l'autonomie (pas besoin de transmettre l'information concernant les ordonnancements locaux) et permettent l'hétérogénéité des SGBD composant le système multibases. La flexibilité de la gestion globale permet une autonomie considérable pour les sites mobiles.

2.2.1.6 Sérialisabilité à deux niveaux

La sérialisabilité à deux niveaux (2LSR) [MRKS91b, MRKS98] étend la notion des ordonnancements globaux LSR. Un ordonnancement global est 2LSR si : (1) il est LSR et (2) sa projection sur les transactions globales est sérialisable (indépendamment des conflits directs).

Deux types de données sont considérés : "locales" et "globales". Les données locales concernent chaque SGBD et existaient avant la définition du système multibases. L'ensemble des SGBD et leurs données composent le niveau local du système. Les données globales sont insérées lors de la création du système multibases. L'ensemble des sites contenant des données globales est appelé niveau global.

3. Une contrainte d'intégrité globale est une contrainte qui lie des données provenant de plusieurs sites. La vérification de ces contraintes est de la responsabilité du système multibases.

4. Lorsque l'exécution d'une transaction t_i dépend d'une valeur produite/fournie par une transaction t_j, alors t_i a une dépendance de valeur sur t_j.

La répartition des données en local et global dans chaque site permet de définir une hiérarchie de modèles de systèmes multibases qui assurent la 2LSR. Les modèles se basent sur la restriction des opérations de lecture et écriture des transactions locales et globales. Les modèles varient du plus restrictif (où les transactions globales accèdent seulement aux données globales et les transactions locales aux données locales) au plus général (où des restrictions minimales sont imposées aux transactions).

Le modèle le plus restrictif, appelé "trivial", se base sur les restrictions suivantes sur les transactions :

1. les transactions locales lisent et écrivent seulement sur des données locales, et

2. les transactions globales lisent et écrivent seulement sur des données globales.

Dans ce modèle, le niveau global est complètement découplé du local. Les contraintes sur les données peuvent être locales et globales. Les **contraintes locales** concernent des données locales. Les **contraintes globales** peuvent concerner plus d'un site mais seulement des données globales.

Ce modèle est simple à mettre en place mais il est très restrictif. Le niveau local ne peut pas profiter du global et vice-versa. Afin de faire interagir les deux niveaux, d'autres modèles ont été proposés. Par exemple, si la restriction 1 du modèle trivial est remplacée par :

1'. les transactions locales lisent et écrivent sur des données locales et lisent les données globales.

La contrainte suivante doit être ajoutée : les transactions globales ne doivent pas avoir des dépendances de valeur.

En plus des contraintes globales et locales, des contraintes portant sur des données globales et locales peuvent être définies (contraintes localo-globales).

Si plus d'interaction est nécessaire entre le niveau global et local, alors les restrictions 1 et 2 du modèle trivial sont remplacées par :

1'. les transactions locales lisent et écrivent sur des données locales et lisent les données globales et

2'. les transactions globales lisent et écrivent sur les données globales et locales.

Avec ce troisième modèle, les contraintes d'intégrité globales et locales sont autorisées. Par contre, les contraintes localo-globales et les dépendances de valeur entre les sous-transactions d'une transaction ne sont pas autorisées.

Dans [MCFP96, Mor96], il est proposé une approche qui permet d'obtenir des ordonnancements 2LSR en présence de dépendances de valeur. Cette approche permet de répondre aux besoins d'un spectre plus large d'applications.

La sérialisabilité à deux niveaux paraît être bien adaptée à l'environnement mobile. Elle préserve l'autonomie des sites et permet leur hétérogénéité (similairement à LSR). Le seul problème est la complexité ajoutée à la définition des transactions, du fait des contraintes d'accès aux données (locales et globales) et des contraintes d'intégrité.

2.2.1.7 Epsilon-sérialisabilité

L'epsilon-sérialisabilité (ESR) [PL91, WYP92, PL92] est une généralisation de la sérialisabilité. ESR tire profit de l'incohérence que les applications peuvent tolérer afin d'augmenter les performances. L'objectif est de permettre d'une façon explicite une quantité limitée d'incohérence nommée "epsilon".

Les transactions de "interrogation" sont distinguées des transactions de "mise à jour". Le but est de permettre d'obtenir des résultats incohérents à condition qu'ils respectent certaines limites. Les limites d'incohérence sont représentées par un nombre de conflits acceptés entre les transactions. Les limites sont distinguées en limites "d'importation" et "d'exportation". En effet, chacune des transactions d'interrogation a une limite d'importation. Egalement, chacune des transactions de mise à jour a une limite d'exportation. Ainsi, un ordonnancement global incohérent est accepté, pourvu que le nombre de conflits non-sérialisables soit borné. Il est intéressant de souligner que si le nombre de conflits acceptés (limites d'importation et d'exportation) est égal à 0, alors les ordonnancements ESR sont sérialisables.

Plusieurs méthodes ont été proposées pour contrôler la divergence créé par les limites d'incohérence [WYP92]. L'objectif de ces méthodes est similaire à celui des protocoles de contrôle de concurrence. En effet, légèrement modifiées, les approches comme le 2PL, l'estampillage ou la validation optimiste peuvent être utilisées [WYP92].

Si la limite d'incohérence est dépassée, des méthodes de restauration doivent être employées. La restauration peut être basée sur des transactions de compensation ou sur des techniques pour défaire et refaire les transactions (cf. section 2.2.2).

ESR semble adéquat pour l'environnement mobile. Cependant, il faut trouver des techniques de contrôle de divergence adéquates, car à cause des déconnexions, les limites d'incohérence peuvent d'être dépassées. L'inconvénient de cette approche est qu'il faut savoir sur quelle donnée se produit le conflit et quel est le type de conflit (lecture-écriture ou écriture-lecture).

2.2.2 Validation globale

Dans les systèmes répartis, la notion d'atomicité est étendue à l'**atomicité globale**. L'atomicité d'une transaction globale doit garantir que soit toutes les sous-transactions valident soit toutes abandonnent.

Dans un système multibases, du fait de l'autonomie d'exécution, les sous-transactions d'une transaction globale peuvent être annulées unilatéralement. Afin d'assurer l'atomicité et la durabilité globale, le gestionnaire global doit employer des techniques particulières de récupération globale. En général, chaque SGBD assure la durabilité et l'atomicité des transactions locales et des sous-transactions globales. Ainsi, le gestionnaire global doit seulement s'assurer que les sous-transactions, d'une transaction globale, sont toutes validées ou toutes abandonnées.

Un facteur clé dans la récupération globale est l'interface fournie par les SGBD. S'ils supportent l'interface "préparation", assurer l'atomicité est relativement simple avec le protocole 2PC [Gra78]. Cependant, si cette interface n'est pas supportée, alors il est possible qu'une transaction globale valide sur certains sites et annule sur d'autres. Dans ce cas, les techniques pour "refaire" ou pour "compenser" doivent être employées. L'approche refaire

permet d'assurer l'atomicité globale, tandis que la compensation relâche cette propriété mais assure "l'atomicité sémantique".

Les sections qui suivent discutent des avantages et désavantages de l'approche 2PC (section 2.2.2.1), ainsi que des techniques pour refaire (section 2.2.2.2), et pour compenser (section 2.2.2.3).

2.2.2.1 Validation à deux phases

La manière la plus répandue de garantir l'atomicité globale est d'utiliser le protocole de validation à deux phases (2PC) [Gra78]. D'une manière générale, son fonctionnement est le suivant. Lors de la terminaison de l'exécution des opérations des sous-transactions, le gestionnaire global (qui joue le rôle de coordinateur) demande l'opération "prêt-à-valider" à chaque site. Lorsque les sites reçoivent cette opération, ils votent pour une validation ou une annulation. Si le site vote la validation, il entre dans l'état de "préparation". Dans cet état le site n'a plus le droit de valider ou d'annuler la sous-transaction. Lorsque le gestionnaire global reçoit tous les votes, il décide de valider (si tous les votes sont des validations) ou d'annuler globalement (si l'un des votes est d'annulation).

Puisqu'une défaillance de système peut arriver pendant qu'une sous-transaction est dans l'état préparation, chaque site doit stocker les modifications des transactions dans un support de stockage stable. De plus, lorsqu'une sous-transaction t_j est prête à valider, le SGBD doit assurer que les transactions t_i dont t_j a lu, ont validé. Autrement dit, pour que 2PC fonctionne efficacement, les SGBD doivent produire des ordonnancements recouvrables [5] et sérialisables [BHG87].

Fournir l'état de préparation peut être vu comme un relâchement de l'autonomie, néanmoins, grâce à son efficacité, l'interface de 2PC est devenue un standard pour la plupart des SGBD commerciaux. La quasi-totalité des SGBD commerciaux implémente nativement 2PC, néanmoins ceci n'est pas le cas des SGBD commerciaux légers [BBPV02, Vin02, IBM02a, Ora02a].

Malgré la standardisation de 2PC, il n'est pas convenable pour l'environnement mobile car :

- la déconnexion des participants n'est pas considérée ;
- l'état préparation peut entraîner le blocage des transactions indéfiniment, lors des déconnexions ;
- le nombre de messages nécessaires par participant est élevé (4 messages) ;
- les UM ne sont pas considérées comme des stockages stables. Afin de stocker les modifications des transactions sur un support stable, des transferts vers des sites fixes peuvent être envisagés avant de passer à l'état de préparation ;
- certains types d'UM légers n'ont pas de ressources suffisantes pour fournir l'état de préparation.

5. De manière intuitive, un ordonnancement est recouvrable, si chaque transaction valide après la validation des transactions à partir desquelles elle a lu, par exemple : $ecriture_i[x] < lecture_j[x] < validation_i < validation_j$.

2.2.2.2 Refaire les transactions annulées

Reprise des mises à jour

Dans le cas où les SGBD ne supportent pas l'état préparation, une simulation peut être faite. Pour ce faire, un représentant du gestionnaire global, appelé "agent" est installé sur chaque site du système multibases. Les agents jouent le rôle de participant et le gestionnaire global de coordinateur du protocole 2PC. Du fait que les SGBD ne supportent pas l'état préparation, ils peuvent annuler une sous-transaction globale même après que l'agent correspondant ait voté pour une validation globale. Si une sous-transaction est annulée après que le gestionnaire global ait décidé de valider globalement, l'agent du SGBD qui a annulé la sous-transaction demande une "transaction de reprise". Celle-ci est constituée par les opérations de mise à jour journalisées par l'agent lors de l'exécution de la sous-transaction annulée (technique *redo*).

Afin d'éviter qu'une fois que l'agent ait voté, le SGBD, auquel il est attaché, annule la sous-transaction correspondante à cause d'une lecture invalidée, les ordonnancements locaux doivent être sans annulations en cascade [BHG87]. [6]

L'approche pour refaire peut produire des ordonnancements non-sérialisables du point de vue multibases à cause des transactions de reprise. En effet, localement, la transaction de reprise est une nouvelle transaction qui ne pose pas de problème pour être sérialisée. Par contre, pour le gestionnaire global, cette transaction fait partie de la sous-transaction qu'elle reprend. Il faut rappeler que pour être correct, un ordonnancement local doit être sérialisable du point de vue du système multibases (cf. section 2.2.1). En exécutant des transactions de reprise, un ordonnancement local est "m-sérialisable" [MRKS92b, MRB+01], s'il est sérialisable du point de vue du système multibases.

Un ordonnancement est m-sérialisable si sa projection sur l'ensemble des transactions validées (y compris les transactions de reprise) et sur l'ensemble des opérations de lecture des sous-transactions annulées, est sérialisable. La m-sérialisabilité considère que les lectures faites par une sous-transaction (annulée) et les écritures faites par sa transaction de reprise, sont une seule transaction.

Les différents types d'ordonnancements introduits dans la section 2.2.1 peuvent être obtenus en présence de la m-sérialisabilité. Par exemple, afin d'obtenir des ordonnancements LSR (cf. section 2.2.1.5), la m-sérialisabilité peut être assurée en limitant la porté des transactions globales comme dans l'approche des données à deux niveaux (cf. section 2.2.1.6). Par ailleurs, afin d'obtenir la sérialisabilité globale, les transactions globales peuvent être exécutées séquentiellement ou l'algorithme *site graph* (cf. section 2.2.1.2) peut être utilisé.

Reprise totale

La simulation de 2PC peut être obtenue avec d'autres techniques moins contraignantes, sans imposer de restrictions sur les données qu'une transaction globale peut lire ou écrire. Par exemple, si la transaction de reprise exécute, non seulement les opérations d'écriture mais aussi de lecture (approche *retry*) [MR91]. Cette stratégie de reprise n'est possible que

6. De manière intuitive, un ordonnancement est sans annulation en cascade si une transaction peut lire seulement des valeurs écrites par des transactions validées ou par elle-même. Par exemple : $ecriture_j[x] < validation_j < lecture_i[x] < validation_i$.

si les sous-transactions annulées sont rejouables [MRKS92a]. C'est-à-dire, si la validation de la transaction de reprise est certaine, après un nombre borné d'essais, que lque soit l'état des données au moment de la reprise.

Dans cette stratégie de reprise, les dépendances de valeur entre les sous-transactions sont interdites. Ceci, afin que la sous-transaction annulée (qui sera rejouée) ne communique aucune information concernant ses opérations à d'autres sous-transactions.

Afin de générer des ordonnancements corrects, ce mécanisme de reprise peut être combiné avec les techniques introduites dans la section 2.2.1. Ainsi, si le critère de correction à assurer est LSR, le gestionnaire global ne fait pas de contrôle de concurrence global car il repose sur l'hypothèse suivante : après l'exécution de la transaction de reprise, les ordonnancements locaux sont sérialisables. Par ailleurs, si la correction à assurer est 2LSR, alors le gestionnaire global doit considérer la transaction de reprise de la sous-transaction annulée, comme une sous-transaction de la transaction globale originale. Si la sérialisabilité globale doit être assurée, la présence des transactions de reprise peut entraîner des conflits indirects entre les transactions globales à travers les transactions locales. Les techniques comme *site graph* ou le verrouillage altruiste (section 2.2.1.2) peuvent être utilisées afin de prévenir les cycles introduits par les conflits indirects.

En général, la technique de reprise totale peut être utilisée pour assurer l'atomicité globale si (1) chacune des sous-transaction est re-jouable et (2) les dépendances entre les sous-transactions d'une transaction globale sont interdites.

Les deux types de reprises (de mise à jour et totale) peuvent être appliquées à l'environnement mobile. Il faut, néanmoins, s'assurer de faire une bonne combinaison avec les techniques qui préservent le critère de correction. C'est-à-dire, ne pas perdre le gain obtenu avec la technique de reprise en utilisant des méthodes de correction contraignantes comme celles introduites dans les sections 2.2.1.2, 2.2.1.3 et 2.2.1.4.

2.2.2.3 Compenser les transactions annulées

A l'inverse des techniques pour refaire, la technique de compensation consiste à annuler sémantiquement les opérations de mise à jour faites par les sous-transactions validées (marche en arrière sémantique). Pour ce faire, l'agent démarre une nouvelle transaction appelée "transaction de compensation". A titre d'exemple, si une sous-transaction effectue la réservation d'une place sur un vol, la transaction de compensation doit annuler la réservation. Comme la sous-transaction de réservation a été visible (d'autres transactions ont pu la percevoir) l'état de la bases de données, après l'exécution de l'annulation de la réservation, n'est pas forcement le même que si la réservation n'avait jamais eu lieu.

Cette technique de récupération se base sur la richesse sémantique des applications. En effet, ce qu'une transaction de compensation doit faire dépend fortement de l'application en question.

Les transactions de compensation doivent avoir certaines propriétés, par exemple :

- Une transaction de compensation est exécutée sur un site comme n'importe quelle autre et elle est contrainte au contrôle de concurrence local. Elle doit egalement respecter les contraintes d'intégrité locales.

- Une transaction de compensation déclenchée, doit valider. Cette contrainte est appelée "persistance de compensation" [GM83, GMS87, KLS90]. Si cette contrainte est garan-

tie, il n'y a pas besoin d'utiliser un protocole de validation pour assurer l'atomicité des transactions de compensation.

L'utilisation de transactions de compensation, comme moyen de récupération, peut remettre en cause la cohérence de la base de données et la correction des ordonnancements locaux et globaux. Dans la suite, quelques explications sont introduites à ce sujet. Considérons que t_i est une sous-transaction compensable est que tc_i est sa transaction de compensation.

- Une transaction doit percevoir soit l'état de la base de données modifiée par la transaction compensable, soit l'état après la transaction de compensation mais pas les deux. Ceci est appelé "atomicité de compensation" [KLS90].

- Les transactions de compensation doivent concerner uniquement le site où la sous-transaction compensable a été exécutée. Si t_i a une influence sur d'autres sites, ceux-ci auront besoin d'exécuter en quelque sorte tc_i, ce qui complique considérablement la récupération. Afin d'éviter ce problème, deux restrictions peuvent être mises en œuvre :

  1. prévenir la visibilité des résultats partiaux entre transactions globales. Ceci peut être obtenu en interdisant l'exécution de transactions globales entre t_i et tc_i ;
  2. interdire les dépendances de valeur entre les sous-transactions d'une transaction globale.

- L'ensemble de transactions dépendantes de t_i ($\mathrm{dep}(t_i)$) sont les transactions qui lisent les modifications faites par t_i et qui sont ordonnancées après t_i. La transaction de compensation tc_i doit annuler les effets de t_i, sans entraîner une annulation en cascade des transactions de $\mathrm{dep}(t_i)$. En effet, les opérations faites par $\mathrm{dep}(t_i)$ doivent persister après l'exécution de tc_i. Ceci est obtenu si tc_i est commutable avec les transactions de $\mathrm{dep}(t_i)$ [KLS90]. La commutativité des transactions permet d'ignorer les conflits possibles entre tc_i et $\mathrm{dep}(t_i)$ et l'on peut considérer que tc_i a lieu avant $\mathrm{dep}(t_i)$.

- La validation partielle d'une transaction globale (valider sur un site et annuler sur un autre) peut entraîner la violation des contraintes d'intégrité globales. Ainsi, les transactions de compensation doivent assurer le rétablissement de la cohérence en préservant les contraintes d'intégrité globales. Cependant, les transactions exécutées entre t_i et tc_i peuvent percevoir les incohérences de la validation globale partielle. Il y a deux façons de prévenir un tel problème :

  1. interdire les contraintes d'intégrité globales ou
  2. interdire aux transactions globales de percevoir les effets des sous-transactions (celles qui valident et celles qui annulent) d'une transaction globale avant sa validation ou sa compensation. Ceci est appelé "isolation de récupération" [LKS91]. Cette contrainte peut être préservée s'il est interdit d'ordonnancer des transactions globales, entre une sous-transaction et sa transaction de compensation.

Les transactions mobiles étant de longue durée (à cause des déconnexions possibles), elles peuvent bénéficier de l'approche par compensation. En effet, cette approche permet, par exemple, de valider les sous-transactions avant la validation de la transaction globale correspondante. Cette forme de récupération peut permettre aux UM de se déconnecter sans entraîner l'annulation des transactions globales et sans bloquer le système.

2.3 Modèles de transactions étendues

Cette section introduit quelques modèles de transactions étendues. Le nombre de modèles étendus proposés dans la littérature étant très large, nous en avons choisi un sous-ensemble représentatif afin de montrer leurs avantages/limites par rapport à leur utilisation dans l'environnement mobile. Les modèles présentés sont les transactions emboîtées fermées (section 2.3.1), ouvertes (section 2.3.2), les Sagas (section 2.3.3), les DOM (section 2.3.4) et les Flexibles (section 2.3.5).

2.3.1 Transactions emboîtées fermées

Le modèle des transactions emboîtées (fermées) propose une structure multi-niveau [Mos81]. Une transaction emboîtée est composée d'un ensemble de sous-transactions, lesquelles peuvent récursivement avoir d'autres sous-transactions formant un arbre hiérarchique de transactions.

Une transaction fille peut être démarrée seulement après sa transaction mère. La transaction mère ne peut valider que si toutes ses transactions filles ont terminé. La validation d'une transaction mère implique la validation de toutes ses transactions filles. Si une transaction mère annule, toutes les filles doivent annuler à leur tour. Cependant, si une transaction fille annule, la transaction mère peut choisir une manière de récupération qui peut être, entre autres, la ré-exécution ou l'exécution d'une autre sous-transaction.

Les transactions emboîtées fournissent la propriété d'isolation au niveau global (transaction racine), c'est-à-dire, que les autres transactions ne peuvent pas percevoir ses effets avant sa validation. Par contre, à l'intérieur de la transaction emboîtée, l'isolation peut être relâchée entre les transactions filles et mères.

Les transactions emboîtées fournissent les propriétés ACID au niveau global, et les propriétés AI pour chacune des sous-transactions.

L'utilisation de transactions emboîtées permet, grâce à une décomposition adéquate des sous-transactions, d'augmenter le parallélisme d'exécution et de réduire l'annulation des transactions. En effet, les sous-transactions sœurs (ou mères et filles) peuvent être exécutées en parallèle. L'annulation d'une sous-transaction peut être gérée sans entraîner nécessairement l'annulation de la transaction globale.

La décomposition d'une transaction en sous-transactions peut être très avantageuse pour l'environnement mobile. En utilisant ce modèle, une transaction peut, par exemple, exécuter en parallèle une sous-transaction dans une UM et une autre dans une unité fixe. Cependant, les sous-transactions doivent attendre la validation globale avant de partager leurs modifications avec d'autres transactions. Cette contrainte peut limiter les performances du système mobile, où l'autonomie d'exécution est une nécessité, la communication sans fil doit être limitée et les déconnexions doivent être prises en compte.

2.3.2 Transactions emboîtées ouvertes

Les transactions emboîtées ouvertes [GR93] suivent les principes de structure des transactions emboîtées fermées. Néanmoins, afin de permettre plus de concurrence entre les transactions, l'isolation au niveau global est relâchée. En effet, les sous-transactions peuvent valider et rendre visibles leurs modifications avant la validation de la transaction racine.

Ainsi, ce type de transaction fourni les propriétés de ACD au niveau global et les propriétés AI au niveau des sous-transactions.

Les transactions emboîtées ouvertes paraissent bien adaptées pour l'environnement mobile. Les sous-transactions d'une transaction peuvent être exécutées sur des unités mobiles, valider et partager leurs modifications avec d'autres transactions. Ainsi, les unités mobiles profitent d'une certaine autonomie sans bloquer le système. Néanmoins, certaines adaptations doivent être mises en place afin de prendre en compte les particularités du contexte mobile (par exemple les déconnexions).

Ce modèle a été largement réutilisé et adapté. A titre d'exemple, le modèle de Sagas (cf. section 2.3.3) et DOM (cf. section 2.3.4) s'inspirent des transactions emboîtées ouvertes.

2.3.3 Sagas

Le modèle des transactions Sagas [GMS87] a été conçu pour des transactions de longue durée. Une Saga est une transaction emboîtée ouverte à deux niveaux, composée d'un ensemble de sous-transactions, relativement indépendantes, avec un ordre d'exécution prédéfini (séquentiel ou parallèle). Au fur et à mesure de leur exécution, les sous-transactions valident et partagent leurs résultats avant la validation de la Saga. Ainsi, les résultats partiels d'une Saga peuvent être perçus par d'autres transactions. L'exécution concurrente des Sagas n'est pas contrainte à des ordonnancements sérialisables car seules les sous-transactions commutables sont considérées. A chaque sous-transaction doit être associée une transaction de compensation qui est exécutée en cas d'annulation de la Saga. L'annulation d'une sous-transaction entraîne l'annulation de la Saga. En cas d'annulation, les transactions de compensation sont exécutées dans l'ordre inverse de validation des sous-transactions correspondantes.

Chacune des sous-transaction doit fournir les propriétés AID. Le modèle n'impose pas la cohérence lors de l'exécution partielle d'une Sagas. En effet, la cohérence des sous-transactions peut être relâchée. Au niveau global, l'isolation et l'atomicité sont relâchées. Le fait de ne pas imposer l'isolation et d'utiliser des transactions de compensation conduit à une atomicité sémantique [GM83].

En tant que modèle de transactions emboîtées ouvertes, les Sagas peuvent être appliquées à l'environnement mobile. Cependant, certaines modifications sont nécessaires afin de les adapter aux limitations de l'environnement mobile. Par exemple, l'exécution des transactions de compensation dans l'ordre inverse d'exécution des sous-transactions peut ne pas être possible lors de déconnexions des unités mobiles. Par ailleurs, il n'est pas toujours possible de définir ou de garantir l'exécution des transactions de compensation.

2.3.4 DOM

Le modèle des transactions DOM (*Distributed Object Management*) [BOH+92] s'adresse aux transactions de longue durée dans les systèmes multibases. Une transaction DOM est une transaction composée d'un ensemble de sous-transactions (appelés *toptransactions*). Les sous-transactions sont des transactions emboîtées ouvertes. Elles peuvent être vitales ou non. L'annulation d'une transaction vitale entraîne l'annulation de la transaction DOM, ce qui n'est pas le cas lors de l'annulation des non-vitales. Des dépendances de précédence peuvent être définies entre les sous-transactions. Ainsi, les sous-transactions peuvent être exécutées et validées de manière indépendante, sauf si des dépendances de précédence sont

spécifiées. A chaque sous-transaction doit être associée une transaction de compensation. En cas d'annulation, les transactions de compensation sont exécutées dans l'ordre inverse de validation des sous-transactions correspondantes (comme dans les Sagas). Par ailleurs, il est possible d'associer des "transactions de contingence" aux sous-transactions. Les transactions de contingence sont exécutées lorsqu'une condition de défaillance de la sous-transaction correspondante survient. La condition de défaillance peut être une annulation ou un échec sémantique de la sous-transaction.

Les sous-transactions des transactions DOM ont les propriétés AID. Comme dans Sagas, la cohérence des sous-transactions n'est pas imposée. Au niveau global, les transactions DOM relâchent l'atomicité et l'isolation. Les sous-transactions peuvent valider et rendre visible leurs modifications avant la validation de la transaction DOM.

Ce modèle offre plusieurs points positifs pour les transactions dans les environnements mobiles. Par exemple, l'existence des transactions non-vitales et des transactions de contingence permet de réduire le nombre d'annulations des transactions en cas de variations du contexte. Il faut cependant, comme dans les Sagas, faire certaines adaptations afin d'utiliser ce modèle dans l'environnement mobile.

2.3.5 Transactions Flexibles

Le modèle des transactions Flexibles [ELR90, ZNBB94, KPE92] a été proposé pour le contexte des systèmes multibases. Pour chaque transaction Flexible, le modèle permet de spécifier un ensemble de sous-transactions fonctionnellement équivalentes, où la terminaison de chacune représente l'accomplissement de la transaction Flexible. Le modèle permet, également, de spécifier des dépendances d'exécution entre les sous-transactions. Les dépendances peuvent être de défaillance, de succès, d'initiation, ou externes (par exemple de temps).

Les transactions Flexibles relâchent l'isolation et l'atomicité. Elles proposent la *semi-atomicité* qui permet de spécifier des états de terminaison acceptés (sous-ensemble de sous-transactions terminées avec succès). Ainsi, certaines sous-transactions peuvent annuler sans entraîner l'annulation de la transaction Flexible. Les sous-transactions peuvent ou non valider indépendamment de la transaction Flexible. Les sous-transactions peuvent ou non être compensables. En plus de la compensation, de techniques de refaire (cf. section 2.2.2.2) sont utilisées pour la récupération après pannes (*retry, redo*). L'ordre global assuré est la F-sérialisabilité [ZNB01]. La F-sérialisabilité relâche la sérialisabilité pour prendre en compte la compensation et la ré-exécution lors de la récupération après pannes.

Le fait de permettre d'avoir plusieurs sous-transactions équivalentes et de définir des états de terminaison acceptables, rendent les transactions Flexibles très tolérantes aux pannes. L'environnement mobile étant sujet à de fréquentes variations il peut profiter de cette caractéristique. Cependant, il faut faire quelques adaptations, notamment pour ne pas considérer comme pannes les variations du contexte, mais comme un état normal du système.

2.4 Conclusion

Nous avons commencé ce chapitre par une caractérisation des différents types de systèmes mobiles. Etant composé de SGBD et d'utilisateurs pouvant être localisés sur des UM, un système mobile par nature est très dynamique. Les UM se connectent et se déconnectent à partir de différents emplacements. Ce dynamisme entraîne différents niveaux d'hétérogénéité

et d'autonomie des UM et des SGBD. En particulier, l'autonomie est nécessaire car, d'une part, lors des déconnexions, les UM doivent être capables de continuer leurs activités de manipulation de données de manière indépendante. D'autre part, les limitations dans la capacité de communication amène à restreindre les échanges d'information, et, ainsi, le partage avec les autres composants du système.

Selon le contexte, un système mobile peut avoir une configuration architecturale variable. En effet, selon les caractéristiques de l'environnement mobile courant, les UM peuvent être (dynamiquement) des clients légers ou des serveurs.

Ces réflexions nous font conclure qu'un système mobile est proche des systèmes multibases hétérogènes avec une architecture qui varie du client-serveur au pair à pair.

Nous en avons fait une caractérisation des transactions dans les environnements mobiles. Notre but a été d'identifier les risques occasionnés par la participation d'UM dans l'exécution des transactions (annulations et coûts d'exécution imprévus, possibilité de transactions de longue durée, etc.) à cause des caractéristiques de l'environnement mobile (UM avec ressources limitées, fréquentes déconnexions, bande passante variable et faible, prix de communication élevé, etc.).

Ensuite, nous avons analysé quelques techniques pour assurer un ordre global ainsi que certaines techniques de validation et reprise utilisées dans les systèmes multibases afin d'analyser la possibilité d'adapter ces techniques pour les transactions mobiles.

Le tableau 2.2 montre une synthèse de notre analyse. En ce qui concerne l'ordre d'exécution global des transactions, les techniques qui relâchent la sérialisabilité globale (SG) nous semblent adéquates pour les environnements mobiles (sérialisabilité locale, sérialisabilité à deux niveaux et quasi-sérialisabilité). Parmi les techniques qui assurent la sérialisabilité, nous avons remarqué qu'OTM peut convenir à notre contexte avec certaines modifications. Celles qui ne tolèrent pas l'hétérogénéité, par exemple, dans les protocoles de contrôle de concurrence, nous semblent trop contraignantes.

Pour les techniques de validation et reprise, le protocole 2PC présente plusieurs inconvénients, surtout en présence de déconnexions. Par contre, les techniques de refaire et de compensation qui sont plus souples, peuvent être intéressantes pour l'environnement mobile.

En fin, nous avons analysé quelques modèles de transactions étendues. Notre analyse aboutit à la conclusion que les transactions emboîtées ouvertes, comme les Sagas, sont intéressantes pour l'environnement mobile, en particulier les modèles DOM et Flexible qui offrent beaucoup de souplesse.

Approche	Inconvénients	Avantages
Ordonnancements sérialisables (OTM)	Peuvent générer des abandons où il n'y a pas forcement de problème et ils peuvent entraîner le blocage du système.	Permettent l'hétérogénéité, faciles à implanter et assurent la SG.
Ordonnancements fortement sérialisables	La prévention des cycles nécessite de la présence des sites plus ou moins en permanence sinon le système peut être bloqué.	Assurent la SG.
Ordonnancements à point de sérialisation	L'exécution séquentielle des points de sérialisation peut entraîner de périodes d'attente indéfinis lors des déconnexions des UM.	Assurenet la SG et autorisent l'hétérogénéité.
Ordonnancements rigoureux	Les ressources peuvent être bloquées indéfiniment lors des déconnexions, ils suggèrent l'utilisation de 2PL rigoureux dans tous les sites, limitent l'autonomie.	Assurent la SG.
Ordonnancements localement sérialisables	Relâchent la SG.	L'autonomie est préservée et l'hétérogénéité autorisée, semblent faciles à implanter.
Sérialisabilité à deux niveaux	La définition des transactions peut devenir complexe du fait des contraintes d'accès aux données.	L'autonomie est préservée et l'hétérogénéité autorisée.
Epsilon-sérialisabilité	Il faut prévoir des techniques de contrôle de divergence pour rétablir la cohérence lorsque les limites d'incohérence sont dépassées .	L'autonomie est préservée et l'hétérogénéité autorisée.
2PC	Affecte l'autonomie, ne supporte pas la déconnexion, peut entraîner le blocage du système, nombre de messages élevé.	Assure l'atomicité.
Refaire	Relâche la sérialisabilité.	L'autonomie est préservée et l'hétérogénéité autorisée.
Compensation	Relâche l'atomicité et la sérialisabilité. L'utilisation des transactions de compensation peut être complexe.	L'autonomie est préservée et l'hétérogénéité autorisée.
Transactions emboîtées fermées	Assurent l'isolation au niveau global. Limitent la concurrence et l'autonomie vis à vis des transactions emboîtées ouvertes.	Grâce à leur structure d'arbre, elles permettent d'augmenter le parallélisme d'exécution et de gérer les annulations partielles sans entraîner des annulations globales.
Transactions emboîtées ouvertes	Relâchent l'isolation.	Grâce au relâchement de l'isolation elles permettent plus de concurrence et autonomie.
Sagas	Relâchent l'atomicité et l'isolation, pas toujours possible de compenser.	Très flexibles, l'autonomie est préservée et l'hétérogénéité autorisée.
DOM	Relâchent l'atomicité et l'isolation, pas toujours possible de compenser.	L'autonomie est préservée et l'hétérogénéité autorisée. Les transactions non-vitales et les transactions de contingence peuvent réduire le nombre d'annulations des transactions, en présence des variations du contexte.
Flexibles	Relâchent l'atomicité et l'isolation. La définition des transactions globales peut devenir complexe.	Avoir plusieurs sous-transactions équivalentes pour une transaction globale et définir des états de terminaison acceptables est convenable à l'environnement mobile.

TAB. 2.2 – *Synthèse des techniques multibases par rapport à l'environnement mobile.*

CHAPITRE 3

Les transactions mobiles

Dans le chapitre précédent, nous avons montré que les techniques et modèles transactionnels des systèmes multibases peuvent être applicables à l'environnement mobile avec certaines modifications. Dans ce chapitre, nous présentons plusieurs travaux dédiés aux transactions mobiles qui adaptent des techniques existantes mais aussi qui proposent de nouvelles approches.

Le chapitre est organisé de la manière suivante. Nous commençons par introduire les principales caractéristiques de l'ensemble des travaux analysés (section 3.1). Nous continuons par l'analyse des travaux existant selon les propriétés ACID (section 3.2), puis selon la façon dont ils supportent la mobilité et la déconnexion des utilisateurs (section 3.3). Ce chapitre se termine par nos conclusions (section 3.4).

3.1 Propositions sur les transactions mobiles

Cette section présente un nombre important de travaux dédiés aux transactions mobiles. Les projets analysés sont : Clustering (section 3.1.1), Two-tier replication (section 3.1.2), HiCoMo (section 3.1.3), IOT (section 3.1.4), Pro-motion (section 3.1.5), Reporting (section 3.1.6), Semantics-based (section 3.1.7), Prewrite (section 3.1.8), Kangaroo (section 3.1.9), MDSTPM (section 3.1.10), Moflex (section 3.1.11) and Pre-serialization (section 3.1.12). La section 3.1.13 introduit très brièvement quelques produits commerciaux. La section 3.1.14 résume les principales caractéristiques des propositions analysées.

3.1.1 Clustering

Clustering [PB95, PB99] offre un schéma de duplication pour les environnements mobiles où les clients peuvent souffrir des variations de communication. Il considère un système réparti et il est conçu pour maintenir la cohérence de la base de données. La base de données est divisée dynamiquement en grappes (*clusters*), chacune contient des données liées sémantiquement ou des données localisées dans une même région. Une grappe peut être répartie sur plusieurs sites fortement connectés. Lorsqu'une UM se déconnecte, elle devient une grappe à elle seule. Chaque objet possède deux copies, l'une d'entre elles est une "ver-

sion stricte", l'autre est une "version faible". La première doit être globalement cohérente, la deuxième peut tolérer un certain degré d'incohérence globale mais elle doit être localement cohérente. Le degré d'incohérence est représenté par une divergence bornée entre la version stricte et la faible[1]. Cette incohérence peut varier selon la disponibilité de communication entre les grappes. Les transactions mobiles (TM) sont également strictes ou faibles. Les transactions faibles accèdent uniquement les versions faibles des données tandis que les transactions strictes accèdent uniquement aux versions des données strictes. Les transactions strictes s'exécutent lorsque les sites sont fortement connectés et les faibles lorsque les UM sont faiblement connectées ou déconnectées. Deux types d'opérations sont introduits : lectures faibles et écritures faibles. Les transactions strictes contiennent des lectures et écritures standard qui sont considérées comme des opérations strictes. Les transactions faibles contiennent des opérations faibles. Lors d'une reconnexion, un processus de synchronisation (exécuté sur un serveur de données) conduit la base de données à une cohérence globale.

Les transactions réparties peuvent être exécutées seulement à l'intérieur d'une grappe comme des transactions strictes. Les UM peuvent participer à l'exécution mais uniquement en mode connecté. En mode déconnecté, les UM exécutent des transactions faibles.

3.1.2 Two-tier replication

Two-tier replication est issu de l'analyse faite dans [GHPS96] où les schémas de duplication impatients (*eager*) et paresseux (*lazy*) sont comparés. L'analyse conclut que les schémas impatients ne conviennent pas aux environnements mobiles principalement parce qu'il n'est pas possible de permettre des déconnexions. Two-tier replication est un mécanisme de duplication paresseux qui considère les transactions et la duplication de données pour les environnements mobiles où les UM sont connectées occasionnellement. Il existe dans le système une version maîtresse pour chaque donnée et plusieurs copies. Deux types de transactions sont supportés : transactions bases et tentatives. Les transactions bases accèdent les versions maîtresses (schème de duplication paresseux-maître [GN95]) tandis que les transactions tentatives accèdent les versions tentatives (copies locales). Ces dernières peuvent faire des mises à jour en mode déconnecté et lorsque la communication se rétablit, elles sont ré-exécutées comme des transactions bases afin d'obtenir une cohérence globale. La ré-exécution est coordonnée par la station base (SB) courante. Les résultats de cette ré-exécution peuvent correspondre à des "critères d'acceptation" qui permettent aux résultats de diverger. Ainsi, les ré-exécutions permettent la persistance des mises à jour locales.

3.1.3 HiCoMo

HiCoMo [LH02] (*High Commit Mobile*) est un modèle de transactions consacré à des applications de prise de décision. Son objectif est de permettre la mise à jour de données d'agrégation stockées sur des UM pendant les déconnexions. Il existe des tables bases sur des UF à partir desquelles les tables d'agrégation sont extraites. Les tables d'agrégation représentent un résumé ou des statistiques (moyenne, addition, minimum, maximum, etc.) qui sont stockés sur des UM. De manière similaire à Clustering et Two-tier replication, deux types de transactions sont considérés : HiCoMo et transactions bases. Les transactions HiCoMo

1. La section 3.2.3 donne plus de détails à ce propos.

s'exécutent sur des tables d'agrégation pendant les déconnexions. Les transactions bases reflètent les modifications faites par les transactions HiCoMo sur les tables bases. Ainsi, lors d'une reconnexion, une transaction HiCoMo est transformée en une transaction base – une par table base accédée pendant la génération des tables d'agrégation. La transformation se base sur une analyse complexe qui utilise de l'information sémantique des données et des opérations. Ce processus est un facteur clé dans cette proposition. Afin d'obtenir un haut degré d'exécutions réussies uniquement des opérations commutatives (addition et soustraction) sont considérées pour les transactions HiCoMo. Et comme dans Two-tier replication, une marge d'erreur des résultats des transactions HiCoMo et bases est toléré.

3.1.4 IOT

Coda [SKK+90] est un système de fichiers réparti qui supporte des opérations en mode déconnecté. Il utilise un shéma de duplication optimiste où seul les conflits écriture-écriture sont pris en compte. IOT [LS95, LS94] (*Isolation-Only Transaction*) étend Coda en considérant des conflits lecture-écriture avec un service transactionnel. Dans IOT, les transactions sont une séquence d'opérations d'accès aux fichiers. Les transactions sont classifiées en deux catégories (de manière similaire à Clustering, Two-tier replication and HiCoMo): transactions de première classe dont l'exécution ne contient aucun accès partiel aux fichiers (c'est-à-dire, le client maintient une connexion avec le serveur pour chaque fichier accédé) et transactions de seconde classe qui sont exécutées pendant les déconnexions. Les transactions de première classe valident immédiatement après avoir été exécutées, tandis que celles de deuxième classe entrent dans un état d'attente. Lors d'une reconnexion, les transactions de seconde classe sont validées selon le critère de cohérence espéré, par exemple, sérialisabilité locale, sérialisabilité globale, certification globale (cf. la section 3.2.3 donne plus de détails à ce sujet). Si la validation a été possible, les résultats sont intégrés et validés. Autrement, les transactions entrent dans un état de résolution. La résolution peut être automatique (ré-exécution, spécifique à l'application, annulation) ou manuelle (notification aux utilisateurs).

3.1.5 Pro-motion

Pro-motion [WC99, WC97] aborde le problème du cache de données sur les UM afin de rendre possible une manipulation locale de manière cohérente. Pro-motion est un système de gestion de transactions mobiles qui supporte le mode déconnecté. Pour permettre l'exécution locale, les *compacts* comme unité de cache et de contrôle sont introduits. La sémantique des objets est utilisée dans la construction des compacts afin d'améliorer l'autonomie et la concurrence. Un compact encapsule l'information nécessaire qui permet de le manipuler. Pro-motion utilise de transactions emboîtées-*split* [Chr93, RC96b]. Dans ce travail on considère que le système mobile est composé d'une transaction à longue durée extrêmement grande qui s'exécute sur le serveur. Les ressources utilisées pour créer les compacts s'obtiennent par l'intermédiaire de cette transaction qui contient des opérations de bases de données classiques. L'organisation pour la gestion de compacts suit une architecture à trois tiers : un gestionnaire de compacts localisé sur le serveur de données, un agent localisé sur l'UM et un gestionnaire de déplacement localisé sur les stations bases. Le gestionnaire des compacts interagit avec les serveurs de données qui le considèrent comme un client courant exécutant une grande transaction à longue durée. La construction de compacts est de la responsabilité du gestionnaire de compact. L'agent est responsable de la gestion du cache sur l'UM ainsi

que de la gestion des transactions, du contrôle de concurrence, de la journalisation et de la récupération après pannes. Le gestionnaire du déplacement se charge de la transmission entre les agents. Les transactions des UM s'exécutent localement même en mode connecté. Un processus de synchronisation est exécuté par l'agent et le gestionnaire des compacts lors de reconnexions. Ce processus vérifie les compacts modifiés par les transactions validées localement. Si les compacts préservent la cohérence globale alors une validation globale est faite.

3.1.6 Reporting

Reporting [Chr93] considère un environnement de bases de données mobile comme un système multibases particulier où les transactions sur les UM sont un ensemble de sous-transactions. Les transactions emboîtées fermées [Mos81] et ouvertes (comme les Sagas [GMS87], les transactions *split* [PKN88] et les multitransactions [RC96b]) sont analysées afin de montrer leurs limitations pour les environnements mobiles. Reporting propose un modèle de transactions emboîtées ouvertes qui supporte des transactions atomiques et non-compensables ainsi que deux types de transactions supplémentaires : reporting et co-transactions [Chr91, CR94]. Pendant leur exécution, les transactions peuvent partager leurs résultats partiels et maintenir partiellement l'état d'une sous-transaction mobile (exécutée sur l'UM) dans une SB. Une transaction est structurée comme un ensemble de transactions dont quelques-unes sont exécutées sur des UM. Les auteurs considèrent que les limitations sur les UM rendent nécessaire l'utilisation d'unités fixes, pour stocker ou traiter une partie des transactions. Des transactions emboîtées ouvertes avec des sous-transactions des types suivants sont proposées : (1) transactions atomiques qui ont les propriétés d'annulation et de validation traditionnelles. (2) Transactions non-compensables qui au moment de la validation délèguent à leur transaction mère toutes les opérations faites. (3) Transactions *reporting* qui reportent à une autre transaction une partie de leurs résultats à n'importe quel moment pendant leur exécution. Un report peut être considéré comme une délégation d'état entre transactions. (4) Co-transactions qui sont des transactions *reporting* où le control passe de la transaction *reporting* à celle qui reçoit le report. Les co-transactions sont suspendues au moment de la délégation et elles reprennent leur exécution lorsqu'elles reçoivent un report.

3.1.7 Semantics-based

Semantics-based [WC95] se focalise sur l'utilisation d'informations sémantiques des objets afin d'améliorer l'autonomie des UM en mode déconnecté. Ce travail se concentre sur la "fragmentation d'objets" comme une solution pour les opérations concurrentes et pour les limitations de stockage des UM. Cette approche utilise l'organisation des objets et la sémantique des applications pour découper (*split*) des données larges et complexes en fragments plus petits du même type qui sont plus faciles à manipuler. Chaque fragment peut être stocké de façon indépendante et manipulé de manière asynchrone. Les objets fragmentables peuvent être des agrégats, des ensembles, des queues, des *stacks*, etc. Les transactions mobiles sont demandées par l'UM. Du point de vu du serveur de données elles sont des transactions longues à cause des délais de communication. Les demandes de fragments faites par les UM incluent deux paramètres : un critère de sélection et des conditions de cohérence. Le premier paramètre indique les données qui doivent être stockées sur l'UM et la taille espérée du fragment. Les conditions de cohérence spécifient les contraintes nécessaires pour

préserver la cohérence de la donnée entière. La fragmentation des données exécutée sur le serveur permet un grain plus fin de contrôle de concurrence. Un fragment représente une copie exclusive donnée à une UM où des transactions peuvent être entièrement exécutées. Un processus de réconciliation est exécuté par le serveur lorsqu'une reconnexion se produit. Cette proposition peut être utilisée avec différents types de transactions.

3.1.8 Prewrite

Prewrite [MB01] vise à incrémenter la disponibilité des données sur les UM en proposant deux variations des données : "pré-écrite" et "écrite". La variante pré-écrite reflète l'état futur de la donnée mais sa structure peut être légèrement différente de celle de la version écrite. La valeur pré-écrite est une version petite de la valeur écrite. En conséquence, elle a besoin d'une capacité de stockage moins importante sur l'UM. Par exemple, la valeur pré-écrite d'un objet de type document est le résumé et la valeur écrite est le document complet (résumé, corps et bibliographie). Dans l'approche Prewrite, l'exécution de la transaction est répartie entre l'UM et le serveur de bases de données. Le gestionnaire de transaction sur l'UM exécute la transaction mais les modifications permanentes sont faites par le gestionnaire de données localisé sur le serveur. Prewrite garanti que la délégation de la responsabilité d'écrire sur le serveur de données, limite le traitement des transactions sur l'UM. Trois opérations qui peuvent être exécutées par le gestionnaire de transactions sont proposées : pré-lectures, pré-écritures et pré-validation. Les lectures ordinaires et les écritures permanentes sont faites par le gestionnaire de données. La SB a des capacités de journalisation et maintient une relation proche avec le gestionnaire de données. L'exécution des transactions se déroule comme suit. Le gestionnaire de transaction demande à la SB les verrous nécessaires (en mode connecté). La SB obtient les verrous auprès du gestionnaire de données et l'UM peut se déconnecter. Lorsque le gestionnaire de transactions termine l'exécution, une validation locale (pré-validation) est faite et reportée à la SB. Le gestionnaire de données rend les pré-écritures permanentes et valide la transaction mobile. Prewrite considère les transactions mobiles comme de longue durée et leur implantation peut être faite avec des transactions emboîtées et *split*.

3.1.9 Kangourou

Les transactions Kangourou (KT) [DHB97] proposent un modèle de transactions mobiles qui se focalise sur le mouvement des UM pendant l'exécution des transactions. Dans cette approche, les transactions mobiles sont des transactions globales qui se génèrent sur l'UM et s'exécutent complètement sur le réseau fixe. KT propose l'implantation d'un Agent d'Accès aux Données (AAD) qui se localise au-dessus des gestionnaires globaux de transactions. Cet agent se localise sur des stations bases et gère les transactions mobiles ainsi que le mouvement. Chacun des SGBD intervenant dans le système a la responsabilité de préserver les propriétés ACID des sous-transactions. Le modèle transactionnel utilise les concepts de transactions emboîtées ouvertes [Elm92] et des transactions *split* [PKN88]. L'exécution de la transaction mobile est coordonnée par la SB actuelle de l'UM. Lorsqu'une UM passe d'une cellule à une autre (et elle change de station) la coordination de la transaction mobile bouge également. Ce déplacement se capture par le découpage de la transaction originale en deux transactions appelées Transactions Joeys (TJ). Il existe une TJ par SB. Le découpage concerne uniquement la coordination de la transaction. Ainsi, si l'UM passe de la SB-1 à la

SB-2, la SB-1 coordonne les opérations exécutées pendant le laps de temps que l'UM passe dans la cellule de la SB-1. Les sous-transactions sont exécutées de manière séquentielle, ainsi, toutes les sous-transactions de TJ-1 s'exécutent et valident avant celles de TJ-2.

3.1.10 MDSTPM

MDSTPM [YZ94] (*Multidatabase Transaction Processing Manager*) propose un canevas dans un environnement de multibases de données hétérogènes afin de supporter la demande de transactions à partir des UM. La contribution liée aux déconnexions des UM est l'implantation d'un mécanisme de queues de messages qui gère les échanges entre les UM et le réseau fixe. MDSTPM est localisé sur chaque unité fixe au-dessus du SGBD local existant. Les SGBD locaux ont la responsabilité de gérer le traitement local. MDSTPM gère l'exécution de transactions globales, il assure l'ordonnancement et coordonne les validations globales. A cause des déconnexions, un coordinateur est désigné à l'avance. Ainsi, une fois qu'une UM soumet une transaction globale, elle peut se déconnecter et faire d'autres tâches sans attendre la validation de la transaction soumise. Le site coordinateur gère la transaction mobile au bénéfice de l'UM. Dans MDSTPM comme dans KT, la façon d'assurer les propriétés ACID des sous-transactions dépend de chacun des SGBD participant dans le système sur le réseau fixe.

3.1.11 Moflex

Le principal objectif dans Moflex [KK00] est de fournir l'accès aux systèmes multibases hétérogènes à partir d'UM. Moflex supporte la mobilité des UM, et permet une flexibilité dans la définition et l'exécution des transactions mobiles. Cette approche étend les transactions Flexibles [ELR90] (cf. section 2.3.5) où les transactions sont une collection de sous-transactions liées par un ensemble de dépendances d'exécution : de succès, de défaillance ou externes (de temps, de coût, etc.). En plus des transactions flexibles, Moflex permet la définition de transactions dépendantes de la localisation [DK98b] et le support de l'adaptabilité dans l'exécution des sous-transactions lors du *hand-off*. Les auteurs considèrent que le système est construit sur un système multibases autonome et hétérogène. Ils identifient trois couches dans le système : UM, SB et le niveau global. Dans la couche UM, les utilisateurs conçoivent les transactions Moflex qui sont soumises au gestionnaire des transactions mobiles de la couche SB. Le gestionnaire de transactions mobiles cordonne l'exécution des transactions soumises. Finalement, un gestionnaire global de la couche du système multibases (global) exécute les transactions qui doivent avoir les propriétés ACID

3.1.12 Pre-serialization

Pre-serialization [DG98, DG00] est orienté vers des systèmes multibases mobiles autonomes. Les UM demandent des transactions au système multibases dont les SGBD participants sont localisés sur des unités fixes. Les transactions mobiles sont considérées comme des transactions globales à longue durée composées par des sous-transactions compensables (appelés transactions site). Pre-serialization, à la différence de KT, MDSTPM et Moflex, renforce les propriétés d'atomicité et d'isolation des transactions globales tout en considérant les déconnexions et la migration des utilisateurs mobiles. Afin de minimiser les effets des transactions mobiles (exécutions longues à cause des déconnexions) Pre-serialization permet

aux transactions sites de valider unilatéralement de manière indépendante à la transaction globale. Ceci permet le relâchement des ressources locales au fur et à mesure que les transactions sites terminent leur exécution. De plus, un processus de pré-sérialisation globale s'exécute. Pour ce faire, un algorithme de sérialisation partielle nommé PGSG *Partial Global Serialization Graph* est défini afin de vérifier l'ordre global des transactions. Le gestionnaire de transactions globales est composé d'un coordinateur et d'une couche gestionnaire des sites. Cette couche comprend un ensemble de coordinateurs globaux localisés sur chaque SB ainsi que sur tout autre site qui supporte des utilisateurs externes. Il existe aussi une couche globale qui comprend un ensemble de gestionnaires de transactions sites localisés sur chaque SGBD participant. Les transactions globales sont demandées par une UM à un coordinateur global qui soumet les transactions sites aux gestionnaires des SGBD. La couche globale gère également les déconnexions, la migration des UM, journalise les réponses qui ne peuvent pas être délivrées (aux sites déconnectés) et exécute l'algorithme PGSG.

3.1.13 Produits commerciaux

Cette section introduit quelques produits commerciaux qui proposent des solutions de bases de données pour les environnements mobiles. Puisqu'il s'agit de produits commerciaux, nous n'avons pas la même information technique que nous avons eu pour les travaux de recherche décrits jusqu'ici. C'est pour cette raison que notre discussion va rester à un niveau très descriptif. Les produits abordés sont : PointBase, FastObjects j2 ainsi que les propositions d'Oracle et IBM. Il existe d'autres propositions (qui ne sont pas présentées ici) comme eXtremeDB [McO02], Sybase iAnywhere Solutions [Syb02] ou encore IBM Cloudscape [IBM02b].

PointBase. PointBase [Poi02] est une base de données relationnelle pour les environnements mobiles. Elle est directement embarquée (*embedded*) dans les applications "hors line" et peut tourner sur n'importe quelle UM comme les ordinateurs portables, les tablettes PC et les PDA. PointBase est entièrement écrite en Java avec une empreinte (fichier Jar) entre 45 Kb et 90 Kb pour la version *micro* et 1 Mb pour la version embarquée. Elle peut être exécutée sur toute plate-forme qui supporte une machine virtuelle Java (JVM). PointBase supporte les connexions multiples à partir d'une seule application tournant sur la même JVM. PointBase *micro* inclut un support transactionnel pour des transactions plates (opérations *auto-commit, commit* et *rollback*) et partiellement elle implante le standard de SQL92 et l'interface de programmation JDBC. PointBase *embedded* supporte des transactions réparties (validation à deux phases 2PC), elle utilise le verrouillage par ligne et fournit les quatre niveaux d'isolation de l'ANSI SQL92 (*read uncommitted, read committed, repeatable read, serializable*). Par l'intermédiaire de PointBase UniSync, une base de données PointBase peut être synchronisée bi-directionnellement avec une base de données corporative comme Oracle ou Microsoft SQL Server. Pour stocker des données corporatives sur des UM, PointBase UniSync utilise un mécanisme de publication-souscription. Ce mécanisme permet aux applications clientes de souscrire aux données publiées (sous-ensemble de lignes et tables) sur le serveur.

FastObjects j2. FastObjects j2 [Fas02] est une base de données à objets (anciennement appelé Navajo Poet). Elle fournit des composants base de données embarquées dans un paquet

Java de 450 Kb. FastObjects j2 est un composant single-host pour des applications embarquées qui tournent sur tout environnement Java. Elle a une architecture modulaire avec un noyau qui fournit les fonctions essentielles de bases de données comme la gestion du cache à objets ou la gestion de transactions. FastObjects j2 utilise un verrouillage par objet et fournit les quatre niveaux d'isolation définis dans l'ANSI SQL92. L'architecture modulaire permet l'incorporation de particularités comme la gestion de versions, des événements, le support de XML, la journalisation (récupération basée sur la technique défaire) et la duplication si elles sont demandées par les applications. FastObjects j2 permet un accès aux bases de données multifils hors-ligne et fournit des transactions ACID (plates), emboîtés et parallèles. Elle supporte JDOQL (*Java Data Objects Query Language*) et elle est conforme aux standards JDO et ODMG (*Object Data Management Group*). FastObjects j2 offre un mécanisme de duplication de style "base de données ombre" (*shadow database*). Les copies peuvent être modifiées par des transactions, les modifications sont journalisées et appliquées à la base de données primaire.

Oracle9iAS Wireless et Oracle9i Lite. Oracle aborde l'accès mobile aux données avec le serveur d'application Oracle9iAS Wireless [Ora02b]. Oracle9iAS Wireless rend une application web (localisée sur des unités fixes) accessible à partir d'UM. Il inclut des services mobiles comme PIM (*Personal Information Manager*), émail, basés sur la localisation des UM (pour des applications sensibles à la localisation des UM) et services de *push* – via SMS (*Short Message Service*), WAP (*Wireless Application Protocol*), émail et voix. En particulier, Oracle9iAS Wireless fournit des transactions mobiles sécurisées à partir de n'importe quelle UM (ceci est orienté commerce électronique). Afin de permettre une exécution hors-ligne, Oracle propose Oracle9i Lite [Ora02a] (une extension d'Oracle9iAS Wireless), une base de données relationnelle avec une empreinte de 50 Kb à 1 Mb (ceci dépend de la plate-forme utilisée). Elle tourne sous les systèmes d'exploitation : Windows CE, Windows 95/98/NT/2000, Palm OS ou Symbian EPOC. Oracle9i Lite supporte des transactions plates ACID et les quatre niveaux d'isolation de l'ANSI SQL92. L'accès concurrent est contrôlé avec un mécanisme de verrouillage par lignes. Multiples connexions JDBC ou ODBC sont supportées. Il est possible de créer des copies (*snapshots*) pour des UM à partir de sites maîtres Oracle. Les *snapshots* peuvent être modifiés en mode déconnecté. Une synchronisation bi-directionnelle (Mobile Sync) avec les bases de données corporatives est faite (en mode connecté) par le site maître. Mobile Sync synchronise multiple UM simultanément.

WebSphere Everyplace et DB2 Everyplace. IBM a développé la famille de produits WebSphere Everyplace pour la gestion de données dans les environnements mobiles. WebSphere Everyplace Access [IBM02c] permet aux UM d'accéder à des données (émail, PIM et applications de données d'affaires). Il délivre des pages web et des applications d'é-affaires aux téléphones cellulaires et aux PDA. Il contient un service de perception de localisation qui fournit des informations sur la localisation aux applications mobiles. IBM propose également DB2 Everyplace [IBM02a], une base de données relationnel hors-ligne mono-utilisateur avec une empreinte de 180 Kb qui facilite le stockage sur UM. Il est disponible pour les systèmes d'exploitation Palm OS, Symbian OS, Windows CE, Windows 95/98/NT/2000/XP, QNX Neutrino, Linux, et Linux embarqué. DB2 Everyplace supporte un sous-ensemble du standard SQL et fournit l'indexation. Il contient un serveur de synchronisation bi-directionnel qui synchronise des données relationnelles entre l'UM et les données sources de l'entreprise

	Type transaction	Model d'exécution	Exécution sur l'UM	Exécution sur le réseau fixe	Modes d'opération
Clustering	Transactions strictes et faibles	2 et 3	Transactions faibles et validation locale en mode déconnecté. Participation dans l'exécution des transactions strictes en mode connecté	Transactions strictes et validation des transactions faibles (synchronisation, mise-à-jour permanente)	Connecté, faiblement connecté, déconnecté
Two-tier replication	transactions bases et tentatives	2 et 3	Transactions tentatives en mode déconnecté. Participation dans l'exécution des transactions base en mode connecté	Transactions bases	Mode connecté et déconnecté
HiCoMo	transactions bases et HiCoMo	2	Transactions HiCoMo	Génération de tables d'agrégation tables et transactions base, exécution des transactions bases	Mode connecté, déconnecté
IOT	transactions de première et seconde classe	2	transactions de seconde classe en mode déconnecté. transactions de première classe en mode connecté	Validation et résolution des transactions de seconde classe	Mode connecté et déconnecté
Pro-motion	transaction emboitées-split à longue durée	2	Le compact agent exécute entièrement la TM et valide localement	Le compact manager se charge de la construction des compacts, de la validation des transactions validées localement (synchronisation, mise-à-jour permanentes)	Mode connecté et déconnecté
Reporting	Transactions emboîtées ouvertes contenant transactions atomiques, non-compensables, reporting et co-transactions	1 et 3	Sous-transactions et transactions globales	Transactions globales et sous-transactions	Mode connecté
Semantics-based	transactions à longue durée	2	TM et validation locale	En réponse aux demandes des UM, le serveur de bases de données fait la fragmentation d'objets (split) ainsi que la réintégration des mises-à-jour (merge)	Mode connecté et déconnecté
Prewrite	transaction à longue durée (emboîtée, split)	2	TM et validation locale	Gestion de verrous et validation des transactions validées localement (opérations d'écriture)	Mode connecté et déconnecté
KT	transaction emboîtée ouverte et split	1	Demande des transactions	Coordination et exécution de transactions	Déplacement en mode connecté
MDSTPM	Multitransactions	1	Demande des transactions	Coordination et exécution de multi-transactions	Déplacement en mode connecté et déconnecté
Moflex	Multitransactions et transactions dépendantes de la localisation	1	Définition des TM	Coordination et exécution de transactions	Déplacement en mode connecté
Pre-serialization	Multitransactions	1	Demande des transactions	Coordination, exécution de transactions et pre-serialization avec l'algorithme PGSG	Déplacement en mode connecté et déconnecté

TAB. 3.1 – *Principales caractéristiques des travaux de recherche sur les transactions mobiles*

	Information générale	**Accès aux données**	**Support Transac.**	**Duplication/Sync**
PointBase	Base de données relationnelle, 45-90 Kb et 1 Mb d'empreinte, application mono-utilisateur hors-ligne *(modèle d'exécution 2)* pour toute plateforme Java	Multiple connexions JDBC avec support de SQL92	Transactions plates dans la version *micro*. Transactions reparties, quatre niveaux d'isolation et verrouillage par ligne dans la version *embedded*	Mécanisme publication-souscription et PoinBase UniSync avec synchronisation bi-directionnelle avec Oracle ou Microsoft SQL
FastObjects j2	Base de données à objets, 450 Kb d'empreinte, application mono-utilisateur hors-ligne *(modèle d'exécution 2)* pour tout plateforme Java	Multiple connexions JDBC avec support de JDOQL	Transactions ACID parallèles et emboîtées (verrouillage par objet), quatre niveaux d'isolation	Base de données ombre pour sauvegarder des données applicatives
Oracle9iAS Wireless	Accès aux applications web en-ligne *(modèle d'exécution 1)*	Fournit accès au PIM, émail, services *push* et services basés sur la localisation		
Oracle9i Lite	Base de données relationnelle avec 50 Kb à 1 Mb d'empreinte, applications hors-ligne *(modèle d'exécution 2)* pour plusieurs systèmes d'exploitation	Multiple connexions JDBC/ODBC	Transactions ACID, quatre niveaux d'isolation, approche par verrouillage dans le contrôle de concurrence	Synchronisation bi-directionnelle des copies (*snapshot*)
WebSphere Everyplace Access		Accès aux applications web en-ligne *(modèle d'exécution 1)*	Fournit accès à l'émail, PIM et applications de données d'affaires, il fournit un service de perception de la localisation	
DB2 Everyplace	Base de données relationnelle avec 180 Kb d'empreinte, applications hors-ligne *(modèle d'exécution 2)* pour plusieurs plate-formes	Multiple connexions JDBC/ODBC avec un sous-ensemble de SQL. Exécution éloignée de requêtes et procédures stockées en mode connecté	Transactions plates	Synchronisation bi-directionnelle avec DB2 Everyplace Sync Server

TAB. 3.2 – *Survol de produits commerciaux de gestion de données pour les unités mobiles*

(DB2, Oracle, Informix, Sybase, MS SQL Server et Lotus Domino). DB2 Everyplace fournit la gestion de transactions plates (opérations *commit, auto-commit* et *rollback*). Il supporte des connexions multifils aux bases de données (ODBC/JDBC) de manière sérialisable. En mode connecté les UM peuvent demander l'exécution de requêtes et procédures stockées sur le serveur de données.

3.1.14 Résumé et discussion

Le tableau 3.1 résume les principales caractéristiques des projets de recherche abordés : type de transaction, modèles d'exécution, parties exécutées sur l'UM ou sur les UF et les modes d'opération (connecté, faiblement connecté, déconnecté).

Tous les modèles, à l'exception de Reporting considèrent que les transactions mobiles sont demandées par des UM. Dans Reporting, les transactions peuvent être demandées par n'importe quelle unité. Les modèles d'exécution (cf. section 2.1.2) utilisés par chaque transaction sont les suivants. Le premier modèle (complète exécution sur les UF) est appliqué par Pre-serialization, KT, MDSTPM et Moflex. Le deuxième modèle (complète exécution sur une UM) est utilisé par Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion, Semantics-based et Prewrite. Le troisième (exécution repartie entre une UM et des UF) est utilisé uniquement en mode connecté par Clustering, Two-tier replication et Reporting. Les quatrième et cinquième modèles d'exécution – où plusieurs UM font partie de l'exécution – ne sont pas supportés ni par les travaux de recherche ni par les produits commerciaux analysés. Ces derniers utilisent le premier et deuxième modèle d'exécution. Les transactions reparties sont supportées uniquement par PointBase.

En ce qui concerne les modèles transactionnels, Reporting a un apport original. Les auteurs étendent le modèle de transactions emboîtées ouvertes en appliquant des techniques de délégation afin de permettre la visibilité et la délégation de certaines responsabilités des UM aux UF. Dans la plupart des cas, les contributions proposent des nouvelles caractéristiques pour le support de transactions mobiles mais elles ne proposent pas des variations structurelles aux modèles transactionnels.

Le tableau 3.2[2] montre les caractéristiques basiques des produits commerciaux étudiés. D'une part, il est intéressant de remarquer les similitudes entre PointBase et FastObjects j2. Tous les deux ont une approche embarquée et ils sont 100% Java. Nous pouvons également remarquer que FastObjects j2 a un meilleur mécanisme transactionnel tandis que PointBase possède un processus de duplication/synchronisation plus complet. D'autre part, l'approche mobile d'IBM et Oracle, en plus de proposer une petite empreinte de base de données, abordent l'accès aux applications web en-ligne.

Il existe d'autres produits – orientés systèmes de bases de données à empreinte réduite – comme PicoDBMS [BBPV02] et GnatDb [Vin02].

2. Dans la suite, une case vide dans un tableau signifie que nous n'avons pas suffisamment d'information sur ce point.

3.2 Les propriétés ACID pour les transactions Mobiles

Cette section présente comment les TM assurent les propriétés ACID (*Atomicité, Cohérence, Isolation, Durabilité*) [HR83]. Les travaux sont comparés et les caractéristiques communes sont identifiées. Nous dédions une section à chaque propriété malgré le fait que les limites de chacune des propriétés ne sont pas très claires. En effet, la relation entre les propriétés ACID, le critère de correction et les protocoles d'exécution n'est pas facilement identifiable. Les choix suivants ont été faits pour l'organisation de notre analyse : la section concernant l'atomicité (3.2.1) introduit l'information liée aux protocoles de validation ; la section sur la cohérence (3.2.2) aborde la gestion de l'information sémantique ; la section sur l'isolation (3.2.3) décrit les aspects de visibilité et de contrôle de concurrence ainsi que la cohérence mutuelle (des particularités sur la duplication sont discutées ici). Finalement, la section sur la durabilité (3.2.4) est consacrée aux effets durables des TM et aux aspects de journalisation.

Les travaux analysés dans cette section sont Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion, Prewrite, Semantics-based et Reporting. Les propositions qui utilisent le modèle d'exécution 1 – execution complète sur le réseau fixe – se concentrent sur la gestion du déplacement et non sur les protocoles pour assurer les propriétés ACID. Ainsi, nous discutons de KT, MDSTPM, Moflex et Pre-serialization dans la section 3.3.

3.2.1 Atomicité

La propriété d'atomicité est la notion du "tout ou rien" – toutes les opérations d'une transaction doivent être faites ou aucune d'entre elles. Les protocoles de validation assurent cette propriété. La section 3.2.1.1 parle du processus de validation de chacun des travaux analysés. La section 3.2.1.2 parle de protocoles comme UCM [BPA00] et TCOT [KPDS02]. La section 3.2.1.3 montre une discussion qui résume notre analyse.

3.2.1.1 Processus de validation

A l'exception de Reporting, la validation des transactions se fait en deux étapes. La première se réalise sur les UM (validation) locale et la seconde (validation) sur le serveur de la base de données. Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion et Prewrite exécutent une validation locale avec des caractéristiques spécifiques :

- Clustering et Two-tier replication font une validation locale uniquement en mode déconnecté en utilisant des transactions particulières. En mode connecté, un protocole de validation est utilisé (par exemple, 2PC). Dans ce cas, plusieurs sites peuvent participer à la validation.
- HiCoMo, Pro-motion et Prewrite ne font pas la différence entre le mode connecté et le mode déconnecté. La validation locale est faite en utilisant un protocole de validation globale (2PC dans Pro-motion). Dans Pro-motion le programmeur des transactions décide si elles font ou non la validation locale.
- IOT fait aussi une validation locale (en mode connecté et déconnecté) mais la récupération en cas de panne n'est pas garantie. Les auteurs considèrent que dans les

UM le stockage est une ressource limitée et que défaire une transaction demande une quantité importante d'espace. En conséquence, le service de journalisation ne sera plus disponible lorsque l'espace du journal est épuisé. Les transactions validées localement entrent dans un état d'attente.

Dans la seconde étape du processus de validation, les transactions validées localement font les modifications permanentes sur le serveur de la base de données. La validation de la transaction peut comprendre des mécanismes de réconciliation ou la ré-exécution de transactions.

- La réconciliation dans Clustering se fait d'une manière syntaxique où les transactions faibles sont annulées ou font marche en arrière si leurs écritures faibles sont en conflit avec les transactions strictes.
- Dans Two-tier replication, les transactions tentatives sont ré-exécutées (comme des transactions bases) dans leur ordre de validation. Si cette ré-exécution échoue, même si le critère d'acceptation (attachée à chaque transaction tentative) a été pris en compte, alors la transaction tentative est annulée. Afin d'augmenter les chances de réussite, les transactions tentatives peuvent être définies comme des transactions commutatives.
- Dans HiCoMo l'ensemble de transactions bases généré à partir d'une transaction HiCoMo est organisé dans une transaction emboîtée étendue où chaque transaction base est une sous-transaction. Si une transaction base annule – à cause de problèmes avec les contraintes d'intégrité – d'autres transactions bases peuvent être générées (à partir de la même HiCoMo) et exécutée. Le critère pour arrêter de réessayer dépend des valeurs dictées dans la marge d'erreur tolérée. Ainsi, la validation globale peut être souvent garantie grâce aux considérations faites : les transactions HiCoMo sont commutatives, les marges d'erreurs sont tolérées et la ré-exécution des transactions bases est permise.
- IOT fournit quatre options pour réconcilier les transactions en attente : (1) ré-exécuter les transactions en utilisant les fichiers à jour du serveur (ceci est l'option par défaut), (2) invoquer l'*Application Specific Resolver* (ASR) des transactions, ici, le programmeur des transactions peut attacher un ASR à une transaction qui sera automatiquement invoqué par le système, (3) annulation de la transaction et (4) demander aux utilisateurs de résoudre les conflits manuellement.
- Dans Pro-motion, les compacts impliqués dans les transactions validées localement sont vérifiés. Si un compact n'est plus valable alors les transactions mobiles sont annulées et des "procédures de contingence" (attachées à chaque validation locale) sont exécutées afin d'obtenir une atomicité sémantique.
- Dans Prewrite, ni une réconciliation ni une ré-exécution sont faites. Par l'intermédiaire de la gestion des transactions et le protocole de verrouillage Prewrite assure que les transactions validées localement (pré-validées) vont valider sur le serveur de données. Ceci est dû au fait que la version d'écriture et pré-écriture sont en réalité différentes. Ainsi, la pré-lecture, la pré-écriture et la pré-validation sur les UM sont également différentes (mais avec une relation particulière, cf. section 3.2.3) des opérations de lecture, d'écriture et de validation qui sont exécutées pour faire les modifications permanentes.

	Processus de validation	
	Première étape sur l'UM	**Deuxième étape sur SB/serveur BD**
Clustering	Mode déconnecté : *validation locale* des transactions faibles. Mode connecté : 2PC pour transactions strictes	*Validation* implique réconciliation syntaxique avec annulation et marche en arrière dans la résolution des conflits
Two-tier replication	Mode déconnecté : *validation locale* des transactions tentatives. Mode connecté : protocole de validation atomique pour les transactions bases	Transactions tentatives sont re-exécutées en prenant en compte leur critère d'acceptation
HiCoMo	*Validation locale* des transactions HiCoMo	Exécution des transactions bases en prenant en compte de marges d'erreur prédéfinies. Si une transaction base annule, une autre peut être définie et exécutée
IOT	*Validation locale* des transactions locales	Quatre options de résolution pour les transactions de deuxième classe : re-exécution, spécifique à l'application, annulation et notification aux utilisateurs
Pro-motion	*Validation locale* des transactions locales (2PC)	Un processus de synchronisation vérifie les compacts impliqués dans des transactions locales. Dans le cas d'un conflit, les transactions locales sont annulées et des processus de contingence sont exécutés
Prewrite	*Validation locale* des transactions locales (opérations prewrite)	Les modifications locales deviennent permanentes par des opérations d'écriture
Semantics-based	*Validation locale*	Réintégration des modifications (*merging*). Comme les fragments sont des copies exclusives et qu'ils ont d'attaché des conditions de cohérence, il n'existe pas de conflit dans la réintégration.
Reporting	Toutes les sous-transactions sont atomiques et elles peuvent valider de manière indépendante de la transaction mère. En cas d'annulation, des transactions de compensation peuvent être associées aux sous-transactions (excepté les non-compensables)	

TAB. 3.3 – *Résumé du processus de validation*

Le processus de validation est différent dans Reporting où chacune des sous-transactions est atomique et où cependant la transaction globale mobile ne l'est pas. A l'exception des *sous-transactions non-compensables*, des transactions de compensation peuvent être associées aux sous-transactions, ainsi l'atomicité sémantique est garantie. Dans les *transactions non-compensables*, *reporting* et *co-transaction* la délégation n'affecte pas l'atomicité car celle-ci ne demande pas que la transaction qui demande une opération soit celle qui la valide ou l'annule. Une transaction est *quasi atomique* si toutes les opérations dont elle est *responsable* sont validées ou aucune d'entre elles. Les sous-transactions peuvent valider ou annuler unilatéralement sans attendre aucune autre sous-transaction ou sa transaction mère.

Dans Semantics-based, les transactions sont considérées à longue durée de vie. Comme les UM sont responsables de la validation des transactions locales, il est possible de supporter des transactions atomiques ou non-atomiques.

Le tableau 3.3 résume le processus de validation.

3.2.1.2 D'autres protocoles de validation

Nous étendons notre analyse et considérons d'autres travaux où les participants peuvent être des sites mobiles ou fixes (modèles d'exécution 3, 4 et 5). En général, la motivation derrière ces protocoles est de fournir un processus de validation mobile qui prend en compte (1) les caractéristiques limitées des réseaux sans fil en réduisant le nombre de messages et (2) la nature mobile des UM en considérant les SB dans le processus de validation.

UCM (*Unilateral Commit Protocol*) [BPA00] supporte les déconnexions et les exécutions hors-ligne (sur les UM). Ce travail est motivé par la faiblesse du protocole 2PC lorsqu'il est exécuté dans les environnements mobiles : pas de processus hors-ligne, la nécessité que l'UM supporte l'état de préparation et les 2 tours de messages. UCM est un protocole d'une phase où la phase de vote de 2PC est éliminée. Le coordinateur joue le rôle d'un "dictateur" et diffuse sa décision à tous les participants. Plusieurs hypothèses sont faites. Par exemple, tous les participants sont contraints de sérialiser leurs transactions avec un protocole de contrôle de concurrence pessimiste comme 2PL strict[3] et au moment de la validation les effets de toutes les transactions locales doivent être journalisées sur un stockage stable.[4] UCM garantit l'atomicité et la durabilité ; cependant, à cause des hypothèses, les données accédées par des transactions locales non-validées sont bloquées jusqu'à ce que les journaux des modifications soient tranférés (*flushed*) sur une UF.

TCOT (*Transaction Commit On Timeout*) [KPDS02, Kum00] utilise des *timeouts* (délais) afin de fournir un protocole non bloquant qui limite le besoin de communication. En effet, au lieu d'utiliser des messages pour savoir si un participant mobile est prêt à valider, le coordinateur de la validation attend que les délais expirent. Le coordinateur est installé sur l'actuelle SB de l'UM (qui demande l'exécution de la transaction) ou saute de SB en SB avec l'UM. Ainsi, les participants doivent envoyer un message de validation au coordinateur. Si les délais des participants expirent et si le coordinateur n'a pas reçu un message de validation ou d'annulation, la transaction est annulée. Si toutes les validations des participants sont reçues avant un délais global, la transaction est validée sans envoyer un message global de validation. Si nécessaire, les délais peuvent être re-négociés pendant l'exécution. Les participants peuvent valider localement avant la validation globale. Si la validation globale échoue, le coordinateur envoie un message global d'annulation et les transactions de compensation sont exécutées. Pour valider de manière unilatérale, les participants mobiles doivent envoyer leurs journaux de modifications au coordinateur. Lorsque la validation globale est réussie, le coordinateur envoie les modifications faites par l'UM au SGBD correspondant localisé sur le réseau fixe (des problèmes de réconciliation ne sont pas considérés). Le protocole considère le mode d'opération en veille (cf. section 1.1.1) mais ne supporte pas les déconnexions. TCOT fournit l'atomicité sémantique [GM83]. Afin de fournir l'atomicité (non l'atomicité

3. 2PL strict (la version la plus implanté de 2PL) libère les verrous d'écriture après la validation ou l'annulation de la transaction, les verrous de lecture peuvent être libérés lorsque la transaction termine. Les ordonnancements sont *strict* [BHG87] (c'est-à-dire ils sont recouvrables et évitent les annulations en cascade).

4. Le stockage physique des UM n'est pas considéré stable car les UM sont contraintes au pertes, endommagements ou déconnexions indéfinies. Ainsi, uniquement les UF sont considérées comme stockage stable.

sémantique) et éviter les annulations en cascade, TCOT propose l'utilisation de 2PL strict par chacun des participants.

3.2.1.3 Discussion

Conceptuellement, Semantics-based, Pro-motion, Prewrite et Reporting considèrent les transactions comme à longue durée. Si ces transactions sont exécutées sur des systèmes multibases, l'atomicité globale dépend du niveau d'autonomie de chaque SGBD [BGMS92]. S'il y a des SGBD qui ne peuvent pas participer dans une validation atomique globale, l'atomicité est difficile à garantir. Si les approches introduites dans la section 3.2.1.2 s'appliquent aux systèmes multibases, l'autonomie est violée car les SGBD sont contraints de transmettre leurs journaux aux coordinateurs de la validation (in TCOT) ou à une UF (UCM). De plus, UCM et TCOT assument que tous les participants dans le système utilisent 2PL strict (cette hypothèse peut être relâchée dans TCOT). De telles hypothèses peuvent conduire au blocage des données pour des périodes de temps indéfinis. Cependant, les annulations en cascade sont évitées.

Les annulations en cascade peuvent apparaître dans Clustering, Two-tier replication et Pro-motion. Cependant, comme les transactions validées localement modifient des données locales, seule l'annulation de transactions locales est générée. De plus, ces annulations concernent seulement des transactions faibles et tentatives car les résultats locaux sont exclusivement disponibles pour ce type de transactions. IOT prévient l'incohérence en cascade par la notification aux utilisateurs des objets accédés par des transactions non validées. Dans la réconciliation, où une transaction non validée est en train d'être résolue, l'état d'incohérence local et global de l'objet doit être exposé, de cette façon, le processus de résolution peut choisir l'état global ou local d'un objet.

A propos des travaux qui utilisent le modèle des transactions réparties (Clustering avec les transactions strictes, Two-tier replication avec les transactions base, UCM et TCOT), seul UCM supporte la déconnexion des UM. Dans Clustering et Two-tier replication les UM doivent être fortement connectées. Si une déconnexion survient, le processus de validation génère une erreur. Dans TCOT, si un participant mobile valide localement et ensuite se déconnecte, il ne sera pas informé d'une éventuelle annulation globale.

Généralement, la validation des transactions mobiles est faite en deux étapes : la *validation locale* est faite sur des UM et la *validation* est faite sur la SB/serveur de BD. Cette approche relâche l'atomicité et comparée aux techniques traditionnelles (comme 2PC) elle demande l'exécution d'un processus extra. Cependant, ce type de validation est bien adapté pour les environnements mobiles parce qu'il donne aux UM la possibilité de travailler en mode déconnecté sans entraîner le blocage de l'exécution du système.

3.2.2 Cohérence

La cohérence concerne la correction de la base de données. Généralement, les bases de données ont des contraintes d'intégrité (restrictions à propos de la relation entre les données) qui aident à maintenir un état cohérent particulier de la base de données. Tel état doit être préservé par les transactions, elles doivent conduire une base de données d'un état cohérent à un autre état également cohérent.

	Cohérence et information sémantique
Clustering	Définition de la fonction h et les degrés d'incohérence
Two-tier replication	Définition des transactions commutatives et le critère d'acceptation
HiCoMo	Définition : tables d'agrégation, transactions commutatives HiCoMo et marges d'erreurs. Pour exécuter la fonction de transformation des transactions
IOT	Résolution des conflits ASR
Pro-motion	Construction des compacts (méthodes de type spécifique, règles de cohérence et obligations) et procédures de contingence
Reporting	Délégation, transactions de compensation
Semantics-based	Fragmentation d'objets (conditions de cohérence et opérations *split/merge*)
Prewrite	Définition des variations de données (pré-écrite/écrite)

TAB. 3.4 – *Résumé des aspects de cohérence*

La cohérence est préservée en respectant les contraintes d'intégrité qui son dépendantes de l'application – l'information sémantique est utilisée pour les définir. La section 3.2.2.1 analyse la manière dont l'information sémantique est exploitée afin d'assurer la cohérence des données lorsque des transactions mobiles sont utilisées. La section 3.2.2.2 résume l'analyse.

3.2.2.1 Information sémantique

- Dans Clustering, l'information sémantique est utilisée pour spécifier le degré d'incohérence entre clusters. Ce degré peut être limité par (1) le nombre de validations locales, (2) le nombre de transactions qui peuvent travailler avec des copies incohérentes, (3) le nombre de copies qui peuvent diverger, etc. Il existe aussi la *fonction h* qui contrôle le degré d'incohérence en projetant les opérations strictes sur les versions faibles. La cohérence complète est atteinte par la réconciliation des différentes copies de la même donnée localisée sur différents clusters.

- Dans Two-tier replication, le critère d'acceptation est un test qui permet aux résultats des transactions bases d'être légèrement différents des transactions tentatives. Cette différence acceptable est basée sur la sémantique. L'information sémantique est également utilisée pour définir des transactions tentatives commutatives.

- Dans HiCoMo l'information sémantique est utilisée : (1) pour obtenir les tables d'agrégation, (2) pour définir les transactions commutatives HiCoMo, (3) pour définir les marges d'erreur permises et (4) pour générer les transactions bases. En particulier, la fonction de transformation des transactions (utilisée pour générer les transactions bases) nécessite comme entrée : les tables d'agrégation accédées, les types d'opération, la configuration des tables bases, les contraintes d'intégrité et les conflits entre les transactions bases concurrentes et la HiCoMo concernée.

- Dans IOT, l'*Application Specific Resolver* (ASR) qui est appliqué aux transactions en attente (cf. section 3.2.1) se base sur l'information sémantique.

Pro-motion et Semantics-based exploitent l'information sémantique pour construire les compacts et les fragments :

- Pour Pro-motion le *compact* représente un accord entre le serveur de la base de données et l'UM. Le gestionnaire des compacts et le serveur encapsulent dans les compacts : des données, des méthodes spécifiques, de l'information d'état, des règles de cohérence et des obligations. Si l'agent et le gestionnaire des compacts respectent toutes ces conditions, l'utilisation des compacts n'affecte pas la cohérence de la base de données. Le constructeur des compacts peut déterminer les critères de correction et les méthodes de contrôle de concurrence à utiliser par compact.

- Dans Semantics-based, pour préserver la cohérence, les objets doivent supporter les opérations de *split* (pour faire les fragments) et de *merge* (pour réconcilier les fragments). Une autre restriction pour préserver la cohérence est de fournir les *conditions de cohérence* – fournies par les applications – sur l'objet complet. Ces conditions incluent les opérations permises, les contraintes des valeurs d'entrée et les conditions sur l'état de l'objet.

Reporting ne propose pas de nouvelles manières pour préserver la cohérence mais les sous-transactions peuvent être liées aux transactions de compensation – excepté pour les non-compensables – afin de maintenir la cohérence sémantique en cas d'annulations.

3.2.2.2 Résumé

Le tableau 3.4 résume les principaux concepts utilisés pour préserver la cohérence. L'information sémantique est essentielle pour garantir la cohérence dans les applications mobiles. Tous les travaux analysés profitent de la sémantique des objets de manières différentes. Clustering défini des degrés d'incohérence basés sur la sémantique de l'application. Two-tier replication permet un critère d'acceptation entre transactions tentatives et bases. HiCoMo génère les transactions bases à partir de transactions commutatives HiCoMo, entre autres. IOT utilise un processus particulier (ASR) pour réconcilier les transactions de deuxième classe. Pro-motion utilise l'information sémantique pour construire les compacts et Semantics-based pour appliquer les opérations de *split* et *merge* sur les objets. Reporting base la délégation sur les contraintes sémantiques et Prewrite défini des variations de données "sémantiquement identiques" (objets pré-écrite/écrite).

3.2.3 Isolation

La propriété d'isolation assure que l'exécution de chaque transaction est isolée même si elle s'exécute en concurrence.

Cette section discute trois aspects qui concernent l'isolation : (1) le degré de visibilité pour les transactions validées localement (section 3.2.3.1); (2) le choix à propos des protocoles de contrôle de concurrence (section 3.2.3.2) ; et (3) la cohérence mutuelle (par exemple, sérialisabilité à une copie) pour la duplication des données (section 3.2.3.3). La section 3.2.3.4 introduit de nouvelles propositions sur le contrôle de la concurrence. La section 3.2.3.5 termine cette partie par une brève discussion.

3.2.3.1 Aspects de visibilité

En ce qui concerne la visibilité des résultats transactionnels intermédiaires, Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion et Semantics-based permettent la visibilite des résultats validés localement dans la même UM. Prewrite rend publics les résultats validés localement lorsqu'une validation locale est reportée à une SB. Dans Reporting, la visibilité est permise pour les transactions *atomiques*, *reporting* et *co-transactions* mais non pour les *non-compensatables*. Une transaction *atomique* peut valider son exécution avant la validation de sa transaction mère et ses modifications à la base de données deviennent visibles aux autres transactions. L'objectif des *reporting* et *co-transactions* est précisément de permettre la visibilité des résultats partiels pendant l'exécution.

Si nous prenons Pro-motion et Reporting comme des transactions emboîtées ouvertes, l'isolation globale n'est pas assurée car les sous-transactions ne sont pas exécutées isolement. Après le processus de synchronisation, Pro-motion coupe sa transaction longue. Toutes les opérations qui ont réussi la synchronisation forment une transaction à part qui est validée sur le serveur de données. Les résultats de cette transaction validée sont ensuite visibles dans tout le système.

3.2.3.2 Schémas de contrôle de concurrence

Pour gérer l'exécution concurrente, Clustering et Prewrite utilisent des protocoles du style de 2PL et proposent de nouvelles tables de conflit.

- Clustering utilise 2PL strict et propose quatre types de verrou qui correspondent aux opérations faibles et strictes (LF, EF, LS, ES). Quatre tables de conflit pour la compatibilité des verrous sont proposées. La *fonction de projection h* utilise les tables de conflit pour refléter les opérations strictes sur les versions faibles selon les contraintes de cohérence des applications. Par exemple, une cohérence stricte demande de passer une écriture stricte (ES) sur un objet à des écritures sur toutes les copies (les strictes et faibles). En conséquence, un verrou ES est incompatible avec n'importe quel autre verrou. Les transactions faibles libèrent leurs verrous à la validation locale et les transactions strictes à la validation.

  Si les transactions faibles et strictes s'exécutent de manière concurrente dans une grappe, un ordonnancement correct assure qu'une opération de lecture faible lit des données modifiées par la dernière opération d'écriture (faible ou stricte) et qu'une opération de lecture stricte lit des données modifiées par la dernière écriture stricte.

- Comme Clustering, Prewrite utilise le protocole 2PL et la table des opérations en conflit inclue les opérations de pré-lecture et de pré-écriture (PL, PE, L, E). Les verrous de pré-lecture et de pré-écriture concernent la version de pré-écriture de la donnée. Les verrous de lecture et d'écriture concernent la version d'écriture. Tous les verrous sont gérés par une SB. Afin de rendre les pré-écritures permanentes, le verrou de pré-écriture doit devenir d'écriture ; de cette façon le gestionnaire de données peut écrire et valider les transactions mobiles. Les verrous de pré-lecture sont libérés à la validation locale tandis que ceux de pré-écriture/écriture/lecture à la validation. Si l'on utilise des objets simples (sans les deux versions/variantes) les pré-écritures sont identiques aux écritures et l'algorithme se comporte comme 2PL relâché. Prewrite assure que le

traitement des transactions et le protocole à base de verrous produit uniquement des ordonnancements sérialisables. Cette sérialisabilité se base sur l'ordre de validation locale des transactions mobiles.

Dans HiCoMo, comme les transactions HiCoMo sont commutatives, leur exécution peut être faite sans de fortes restrictions d'ordre. Cependant, les transactions bases ne sont pas commutatives – elles peuvent exécuter des opérations de division ou de multiplication – ainsi, l'ordre entre celles-ci et les transactions HiCoMo est important. Une stratégie optimiste de contrôle de concurrence à base d'estampilles [OV99] est utilisée pour la détection des conflits.

De manière similaire, dans Two-tier replication si les transactions tentatives sont commutatives il n'y a pas besoin d'un mécanisme local de contrôle de concurrence. Cependant, pour exécuter les transactions bases, des mécanismes de verrouillage sont utilisés.

Dans IOT, le contrôle de concurrence se fait en deux niveaux. Le contrôle de concurrence globale des clients est maintenu en utilisant le schéma *optimiste* (OCC) [KR79]. A l'intérieur des clients, le contrôle de concurrence local est renforcé avec 2PL strict avec une détection périodique d'interblocage. La sérialisabilité est garantie localement.

Puisque dans Pro-motion le constructeur des compacts peut déterminer le critère de correction et les méthodes de contrôle de concurrence par compact, les auteurs proposent une échelle de dix niveaux. Les niveaux sont caractérisés selon le degré d'isolation définit dans le standard de l'ANSI SQL tel qu'il a été étendu dans [BBG+95]. Le niveau 9 représente une exécution sequentielle et le niveau 8 une exécution sérialisable. Chaque niveau successif représente un niveau d'isolation plus faible. Au niveau 0, il n'y a pas de garantie d'isolation. Puisque l'utilisation arbitraire des niveaux d'isolation peut conduire à des incohérences, Pro-motion propose quelques règles :

1. Les transactions imposent un niveau minimal pour les opérations de lecture et écriture.
2. Chaque opération est associée à un niveau.
3. Aucun niveau d'opération d'écriture doit être plus faible que le niveau d'écriture de la transaction.
4. Aucun niveau d'opération de lecture doit être plus faible que le niveau de lecture de la transaction.
5. Le niveau le plus faible de toute opération de lecture doit être plus fort ou égal que le niveau le plus fort requis par toute opération d'écriture.

Dans Semantics-based, pour assurer la sérialisabilité, les transactions locales ont accès au fragments stockés dans le *cache* avec l'utilisation de protocoles de contrôle de concurrence conventionnels (comme 2PL).

3.2.3.3 Aspects de duplication

La duplication est étroitement intégrée avec la gestion des transactions mobiles dans plusieurs travaux. Clustering et Two-tier replication maintiennent deux versions des données. Les deux versions sont localisées sur l'UM, l'une d'entre elles (faible/tentative) est utilisée pour supporter l'évolution des données en mode déconnecté. La deuxième (stricte/maîtresse)

doit être toujours cohérente. La cohérence dans les versions strictes/maîtresses est préservée en utilisant des méthodes pour la sérialisabilité sur une copie. Pour fournir la cohérence Clustering utilise le consensus à base de quorum [OV99] et Two-tier replication un protocole de duplication paresseux-maître [GN95]. Dans Two-tier replication, les versions des données tentatives sont abandonnées lors d'une reconnexion car elles sont remplacées par les versions maîtresses complètement fraîches.

HiCoMo, Pro-motion et Prewrite prennent une approche différente. Ils construisent un type de donnée particulier (à partir des sources stockées sur des UF) qui sera stocké sur l'UM et qui sera considéré comme une sorte de copie. Dans HiCoMo, les tables d'agrégation sont générées à partir des tables de base. Le critère de correction est la *convergence*[5] où les tables de base reflètent éventuellement les modifications faites dans les tables d'agrégation. L'approche est similaire dans Prewrite, où la version de pré-écriture est plus petite que la version d'écriture. Dans Pro-motion, à la différence de HiCoMo et Prewrite, les compacts contiennent non seulement des données spécifiques mais également de l'information particulière pour leur utilisation. La flexibilité offerte par les compacts permet à Pro-motion de supporter dynamiquement plusieurs schémas de duplication avec une variété de contraintes de cohérence et critères de correction.

Dans IOT, une variante de l'approche de duplication lit-une écrit-toutes les copies (*read-one, write-all approach, ROWA*) [BG84] est utilisée pour maintenir la cohérence dans un environnement fortement connecté. Avec ROWA, les transactions de première classe sont sérialisables avec toutes les transactions validées. En mode déconnecté, une évolution optimiste sur les UM est considérée. IOT défini un critère de correction appelé *certification globale*. Il demande qu'une transaction en attente soit sérialisable avec et après toutes les transactions validées auparavant. La certification globale est assurée avec une re-exécution systématique des transactions de seconde classe en attente. Ceci est le critère de correction par défaut.

[Dra03] présente une analyse poussée sur les techniques de duplication en général.

3.2.3.4 D'autres approches de contrôle de concurrence

2PL n'est pas approprié pour l'exécution de transactions réparties qui incluent des UM (modèles d'exécution 3, 4 et 5). Ceci est dû au fait qu'on ne connaît pas le temps de verrouillage dû aux déconnexions imprévues. Des variantes ont été proposées comme dans [MV00] où un schéma de contrôle de concurrence qui intègre des approches optimistes et pessimistes est présenté. Un délai (*timeout*) est associé à chacune des données verrouillées. Ce délai correspond à l'intervalle de temps estimé dans lequel est espéré que la transaction valide. Si la validation n'a pas lieu dans cette période de temps (à cause d'une déconnexion), alors la politique pessimiste est changée par une optimiste. A la reconnexion, les transactions validées de manière optimiste sont re-exécutées.

O2PL-MT (O2PL for Mobile Transactions) [JBE95] étend l'algorithme de verrouillage à deux phases optimiste (O2PL) [CL91] pour les environnements mobiles. Dans cet algorithme, les verrous de lecture sont octroyés à la demande et ceux d'écriture sont octroyés jusqu'à la validation. Dans un contexte dupliqué, l'algorithme O2PL-MT réduit le nombre de messages à transmettre lorsqu'il libère les verrous de lecture. O2PL-MT permet la libération d'un

5. La convergence dit qu'éventuellement toutes les copies vont converger vers le même état.

verrou de lecture dans n'importe quel site ayant une copie, sans considérer si ce site est différent de celui qui a mis le verrou.

3.2.3.5 Discussion

Usuellement, l'informatique mobile comprend en quelque sorte la duplication car les données stockées sur les UM sont extraites de bases de données localisées sur le réseau fixe. Sous un environnement dupliqué, deux niveaux de correction sont considérés : localement (sur chaque site) et globalement (sur tous les sites). Dans un environnement fortement connecté, il est possible d'assurer la correction globale (cohérence mutuelle) du système mobile. Par exemple : Clustering, Two-tier replication et IOT dans le tableau 3.5. Dans les environnements faiblement connectés ou déconnectés, le critère de correction global doit être relâché pour éviter le blocage des UM et l'exécution des UF. Ainsi, l'évolution locale peut continuer en permettant une certaine autonomie aux UM. La correction globale éventuelle [RC96a][RC96b] semble être appropriée pour les environnements mobiles car elle est atteinte ou requise "à un temps spécifique", "dans un temps" ou "après qu'une certaine valeur de la donnée est atteinte". Parmi les modèles analysés, il faut remarquer que dans Clustering, une cohérence éventuelle est proposée pour définir des degrés d'incohérence (cf. section 3.2.2.1). Ceci permet "un nombre maximal de transactions qui peuvent être exécutées en mode déconnecté", "dans un intervalle de valeurs acceptables qu'une donnée peut avoir", "un nombre maximal de copies divergentes par donnée", "un nombre maximal de modifications par donnée non-reflétées sur toutes les copies", etc.

Nous pouvons remarquer dans le tableau 3.5 que malgré le fait que les résultats locaux ne sont pas définitifs (à la reconnexion une validation globale doit être faite) la plupart des travaux analysés les rendent visibles localement. Il est intéressant de remarquer également que en général, la correction locale est assurée en utilisant les approches de verrouillage traditionnelles (comme 2PL). Cependant, comme 2PL n'est pas bien approprié pour des exécutions mobiles reparties, des protocoles optimistes sont en train de se développer (cf. section 3.2.3.4).

3.2.4 Durabilité

La durabilité est une condition précisant qu'une fois qu'une transaction est validée, ses effets sur la base de données doivent persister.

Pour rendre durable les effets des transactions mobiles, les validations locales doivent être transformées en globales sur le serveur de données. La section 3.2.4.1 montre quelle est la possibilité de réussite des transactions validées localement sur le serveur de données. Comme la plupart des travaux analysés n'abordent pas de techniques particulières de journalisation, la section 3.2.4.2 complète cette analyse avec d'autres propositions concernées. La section 3.2.4.3 donne une brève discussion des aspects analysés.

3.2.4.1 Durability Guarantees

Clustering, Two-tier replication, IOT et Pro-motion ne peuvent pas garantir la durabilité avant la validation sur le réseau fixe. Pro-motion avec les compacts peut donner quelques

	Visibilité	**Contrôle de concurrence**	**Aspects de duplication**
Clustering	Les résultats des transactions *validées localement* sont visibles aux transactions faibles locales sur la même UM.	2PL, 4 tableaux de conflit et de nouveaux types de verrous sont proposés	2 versions des données : stricte (sérialisabilité sur une copie en utilisant le protocole de quorum consensus), faible (degrés d'incohérence, évolution de données en mode déconnecté)
Two-tier replication	Les résultats des transactions *validées localement* sont visibles aux transactions locales tentatives sur la même UM	Mécanismes de verrouillage pour les transactions bases	2 versions des données : maîtresse (sérialisabilité sur une copie en utilisant un protocole de duplication paresseux maître), tentative (évolution locale des données en mode déconnecté)
HiCoMo	Les résultats des HiCoMo *validées localement* sont visibles aux autres HiCoMo sur la même UM	Ordonnancement optimiste à base d'estampilles pour une HiCoMo avec des transactions bases	Tables d'agrégation et base. La convergence comme critère de correction
IOT	Les résultats des transactions *validées localement* sont visibles sur la même UM	OCC globalement, 2PL localement	Une variation de ROWA pour un environnement fortement connecté, évolution optimiste en mode déconnecté, certification globale comme critère de correction
Pro-motion	Les résultats des transactions *validées localement* sont visibles aux transactions locales sur la même UM	Possibilité de différents niveaux d'isolation et contrôle de concurrence par compact	La définition de compacts permet plusieurs schémas de duplication
Reporting	Avec les sous-transactions *atomiques, reporting et co-transactions* la visibilité est permise avant la validation de la transaction globale		
Semantics-based	Les résultats des transactions *validées localement* sont visibles pour les transactions locales sur la même UM	2PL pour contrôler l'accès aux fragments stockés localement	
Prewrite	Les résultats des transactions *validées localement* sont visibles sur tous les sites	2PL étendu, une table de conflit et de nouveaux types de verrou sont proposés	Variations de données écriture et pré-écriture

TAB. 3.5 – *Résumé des aspects d'isolation*

garanties mais il peut exister des conditions qui ne peuvent pas être respectées à cause des déconnexions ; par exemple il existe un délai (spécifié dans le compact) qui ne peut pas être atteint. En conséquence, il est difficile d'obtenir la durabilité dans le processus de synchronisation. Dans Reporting, les sous-transactions sont durables si les transactions mères valident.

Les approches comme HiCoMo, Semantics-based et Prewrite garantissent la durabilité au moment de la validation locale. Cependant, le premier d'entre eux demande des transactions HiCoMo commutatives, en plus, il emploie une génération complexe des transactions bases (cf. section 3.2.2) et considère la re-génération et la re-exécution des transactions bases (en cas d'annulations). Semantics-based réduit la disponibilité des fragments car une UM peut conserver les fragments pendant une durée de temps indéfinie. Dans l'algorithme de Prewrite, si une transaction mobile fait une validation locale, celle-ci est assurée. L'inconvénient est l'échange de messages nécessaire pour obtenir les verrous des SB.

En ce qui concerne la journalisation, IOT (en réalité Coda [KS92]) propose un mécanisme pour réduire la taille du journal. En mode déconnecté, l'information suffisante pour rejouer les modifications est maintenue dans un journal. Pour réduire la taille de ce journal, dans les opérations de mise à jour, à la place d'enregistrer individuellement les opérations de "ouvrir", de "fermer" et les "écritures" intercalées, un seul registre de "fermer" est journalisé. Les auteurs écartent également les registres antérieurs de "stocker" d'un fichier, lorsqu'un nouveau est ajouté. Ceci est possible car l'opération "stocker" rend inutilisable toutes les anciennes versions d'un fichier.

3.2.4.2 Travaux liés à la journalisaton

Le projet *Little Work* [HH94], comme IOT, propose de réduire la taille du journal. Il suggère d'appliquer la technique basée en règles qui a été utilisée dans les optimisateurs du compilateur *peephole*. Les règles sont utilisées pour éliminer les opérations redondantes ou inutiles des journaux. L'optimisateur prend une liste de règles d'entrée, chacune consistant en un ensemble d'opérations sources suivi par un ensemble d'opérations cibles équivalentes a l'ensemble source. A titre d'exemple, une opération de "créer fichier$_i$" suivie par "renommer fichier$_i$ par fichier $_j$" sera remplacé par l'opération "créer fichier$_j$".

L'effet de la mobilité sur la journalisation est analysé dans [DK98a]. Le problème abordé est : si les transactions mobiles sont réparties sur plusieurs UM, alors où doivent résider les journaux afin de garantir la durabilité? Les auteurs proposent trois techniques.

1. L'approche par journalisation *SB home* maintient le journal de l'UM sur la SB home (la SB où l'UM a été enregistrée initialement) même si l'UM change de SB. Le journal d'une transaction mobile repartie sera éparpillé à travers le SB home des UM participants.

2. L'approche par journalisation *UM home* stocke le journal sur la SB couvrant l'UM au moment de création de la transaction . Le journal de la transaction est centralisé sur une SB.

3. L'approche par journalisation *SB local* stocke le journal sur l'actuelle SB. Ainsi le journal d'une transaction sera éparpillé sur un nombre de SB.

	Garanties de durabilité	Inconvénients
Clustering	Après la validation (réconciliation)	Les transactions validées localement peuvent faire marche en arrière à cause des conflits de réconciliation
Two-tier replication	Après la validation (re-exécution)	Les transactions validées localement peuvent faire marche en arrière à cause des conflits pendant la re-exécution
HiCoMo	Après la validation locale	Transactions HiCoMo commutative, génération des transactions bases complexes, re-génération et re-exécutions
IOT	Après la validation	Les transactions validées localement peuvent être annulées à cause des conflits de réconciliation
Pro-motion	Après la validation (réconciliation)	Les transactions validées localement peuvent faire marche en arrière à cause des conflits de réconciliation
Reporting	Si la transaction mère valide, les sous-transactions sont durables	
Semantics-based	Après la validation locale	Limitation de la disponibilité des fragments sur le serveur de bases de données
Prewrite	Après la validation locale	Important échange de messages entre les UM et les SB

TAB. 3.6 – *Résumé des aspects de durabilité*

3.2.4.3 Discussion

Le tableau 3.6 montre les garanties de durabilité et les inconvénients. Il faut remarquer que généralement, la durabilité est garantie après la validation, c'est-à-dire, lorsque les validations locales sont réintégrées au serveur de la base de données. Dans le cas où la validation locale garantie la durabilité, les inconvénients concernent la disponibilité des données, les coûts de communication ou une réintégration des données complexes.

Par ailleurs, des techniques d'optimisation des journaux ne sont pas proposées. Uniquement IOT et le projet *Little Work* abordent cet aspect. Cependant, du fait des limitations des ressources des UM, des méthodes pour réduire la taille des journaux sont nécessaires et plus d'efforts de recherche à ce propos sont nécessaires.

Puisque les UM ne sont pas considérées comme des stockages stables, il faut considérer le stockage des journaux sur des UF (comme les SB). En ce qui concerne les approches de localisation des journaux proposées dans [DK98a], le bénéfice d'une technique ou d'une autre dépend du profil de déplacement des UM ainsi que de la répartition de l'exécution des transactions réparties. La section 3.3 donne plus de détails sur cet aspect.

3.3 Gestion du déplacement et de la déconnexion

Les travaux analysés auparavant ne donnent pas de détails sur la gestion du déplacement des UM. Uniquement Pro-motion inclut dans son architecture un *gestionnaire du déplacement* qui s'occupe de la communication entre l'UM et le serveur de base de données sans fournir de détails sur son fonctionnement. Ainsi, ici nous proposons une analyse complémentaire pour la section 3.2.

Dans KT, MDSTPM et Moflex, les propriétés ACID ne sont pas affectées par le déplacement car l'exécution des transactions est sous la responsabilité des SGBD localisés sur des UF. Cependant, comme les transactions sont demandées à partir d'UM, le déplacement et les déconnexions sont gérées. Pre-serialization est un cas particulier dans cette section parce que même si les transactions sont exécutées sur le réseau fixe, les propriétés d'atomicité et d'isolation sont renforcées tout en permettant la déconnexion des utilisateurs pendant l'exécution des transactions.

La section 3.3.1 met en lumière la manière selon laquelle les travaux analysés gèrent le déplacement et les déconnexions. La section 3.3.2 discute et compare les différentes approches.

3.3.1 Les aspects liés au déplacement et à la déconnexion

Dans KT pour supporter le déplacement et les déconnexions des UM, l'agent d'accès aux données garde la trace des déplacements avec la gestion d'une liste de toutes le SB qui ont été des coordinateurs de la transaction mobile. Cette liste est utilisée en cas d'abandon. Il existe également des structures de données (*tableau de statut des transactions* et un *journal local*) qui stockent de l'information sur les transactions mobiles comme : l'identificateur global des transactions, le statut (active, validée, annulée), l'identificateur des Transactions Joey (TJ), les sous-transactions qui sont dans la TJ, les transactions de compensation (si elles existent), etc.

Dans KT deux modes différents de gestion sont supportés : *compensation* et *split*. Sous le mode de compensation, la défaillance de toute TJ cause la marche en arrière de la TJ actuelle ainsi que de toutes celles qui ont été exécutées antérieurement. Les TJ validées doivent être compensées. Ce mode d'opération nécessite que l'utilisateur fournisse les transactions de compensation et que le SGBD sous-jacent garantisse leur validation. Sous le *split*, lorsqu'une TJ annule, aucune autre transaction globale ou locale n'est demandée. La décision de la validation ou annulation des transactions actives est laissée aux SGBD locaux. Le mode par défaut est *split*. Ni le mode de compensation ni celui de *split* garantissent la sérialisabilité des transactions Kangourou. Le mode de compensation assure l'atomicité mais l'isolation peut être violée car le contrôle de concurrence est géré de manière autonome au niveau local. Il faut remarquer que le déplacement des utilisateurs n'affecte pas ces propriétés.

Dans MDSTPM le principal apport est le mécanisme de messages et queues *Message and Queuing Facility* (MQF) qui fait des échanges asynchrones de messages. Les messages sont de deux types : demande, acquittement et information. Avec MQF, l'UM peut soumettre des transactions globales et ensuite se déconnecter. L'UM et le site coordinateur maintiennent des tableaux et des journaux qui enregistrent l'état global de l'UM et de l'information des transactions globales (*queue de messages, queue de transactions, journal global, tableau de*

transactions globales, tableau de statut des sites). A n'importe quel moment, l'UM peut demander de l'information sur ses transactions globales.

En ce qui concerne les aspects de correction, dans MDSTPM, les SGBD participants sont autonomes et hétérogènes. Ainsi, les mécanismes de gestion des transactions (comme le contrôle de concurrence) peuvent être différents et l'information sur les exécutions locales (journaux) est limitée ou non partagée. Pour gérer les transactions globales MDSTPM implante 2PL strict et utilise la méthode optimiste des tickets (OTM) [GRS94] (cf. section 2.2.1.1) pour résoudre les conflits indirects [BGMS92]. Dans OTM toutes les sous-transactions globales sont forcées d'obtenir un ticket. Ceci cause des conflits additionnels entre les sous-transactions et l'ordre d'exécution est déterminé par l'accès au ticket.

Dans Moflex deux caractéristiques concernant la mobilité sont soulignées : l'exécution de transactions dépendantes de la localisation [DK98b] et l'influence du *hand-off* sur l'exécution des transactions. Dans la définition des transactions, les utilisateurs peuvent spécifier si une transaction est dépendante de la localisation ou non. Pour les transactions dépendantes de la localisation, de règles de contrôle de *hand-off* doivent être spécifiées. Les options sont :

- *continuer* l'exécution de la transaction dans la nouvelle cellule ;
- *re-initier* : annuler la transaction dans l'ancienne cellule et la re-initier dans la nouvelle cellule ;
- *split-continuer* : les opérations exécutées dans l'ancienne cellule valident et les opérations restantes sont exécutées dans la nouvelle cellule ;
- *split-re-initier* : les opérations exécutées dans l'ancienne cellule valident et la transaction est exécutée entièrement dans la nouvelle cellule.

L'opération *split* utilisée ici est similaire à celle utilisée dans KT. Lorsque l'UM saute de SB en SB, les transactions sont divisées (opération *split*) et la coordination est re-localisée sur la nouvelle SB. La définition des transactions peut également inclure des états objectifs qui indiquent des états finaux acceptés. Le protocole 2PC est utilisé. Le gestionnaire de transactions mobiles de la cellule où une des sous-transactions a atteint un état acceptable, devient le coordinateur du protocole 2PC pour la validation globale.

Dans Pre-serialization le principe est de renforcer les propriétés d'atomicité et d'isolation (A/I) tout en supportant la déconnexion et la migration d'utilisateurs mobiles pendant l'exécution de transactions mobiles (considérées comme des transactions globales). Les transactions sites des transactions globales sont organisées en *vitales* et *non-vitales*. Les propriétés A/I sont renforcées uniquement sur l'ensemble des transactions sites vitales. L'annulation d'une transaction site non-vitale ne force pas l'annulation de la transaction globale. La durée de temps entre la soumission de la première transaction site vitale et la terminaison de la dernière est appelée phase vitale d'une transaction globale. Les transactions globales peuvent être dans un des états suivants :

- *active*, l'utilisateur est connecté et l'exécution continue,

- *déconnectée*, l'utilisateur est déconnecté mais la déconnexion a été prévue et une reconnexion est attendue ; l'exécution continue,
- *suspendue*, l'utilisateur est déconnecté et il est suspecté une défaillance sévère,
- *validée* ou *annulée*, la transaction est validée/annulée.

L'exécution de la transaction n'est pas arrêtée lorsque l'utilisateur est déconnecté (d'une manière prévue). Toutes les réponses adressées à un utilisateur déconnecté sont délivrées à la reconnexion.

Le contrôle de la transaction globale migre d'un coordinateur global à un autre selon le déplacement des utilisateurs. Le gestionnaire des transactions sites supervise l'exécution des transactions sites. Celles-ci (vitales ou non-vitales) peuvent être dans un des états suivants : *active*; *terminée* (la transaction site validée dans la base de données locale mais elle n'a pas validée) ; *validée* (la transaction site et la transaction globale correspondante sont validées) et *annulée*. Si la transaction globale est dans l'état déconnecté, l'exécution des transactions sites continue. Un coordinateur global stocke les messages des utilisateurs déconnectés, les délivre à la reconnexion et réactive les transactions dans l'état déconnecté.

Le coordinateur global vérifie les propriétés A/I par l'exécution de l'algorithme Graphe Partial de Sérialisabilité Globale (PGSG) à la fin de la phase vitale. Si A et I sont violées, la transaction globale est annulée ; autrement elle est *pré-ordonnancée*. Après d'avoir été pré-ordonnancée, une transaction globale mobile peut initier uniquement des transactions sites non-vitales. Une fois pré-ordonnancée, la transaction globale établit son ordre de sérialisabilité et elle a la garantie de valider. A la fin de son exécution, chaque transaction pré-ordonnancée exécute une deuxième fois l'algorithme PGSG. L'objectif est de vérifier que les transactions sites non-vitales ne violent pas l'ordre de sérialisabilité établi lorsque la transaction globale a été pré-ordonnancée. Toute transaction site non-vitale qui viole cet ordre est annulée sans affecter la transaction globale. Une transaction globale pré-ordonnancée est annulée uniquement si elle affecte l'exécution d'une autre transaction globale pendant qu'elle est dans l'état *suspendu*. Ainsi, Pre-serialization garanti l'atomicité sémantique et la sérialisabilité globale. Les auteurs proposent une version modifiée de Pre-serialization où les propriétés A et I sont renforcées seulement pour les transactions vitales. Dans cette version, l'algorithme PGSG s'exécute juste une fois, à la fin de la phase vitale.

3.3.2 Discussion

KT et MDSTPM sont très similaires. Ils proposent d'ajouter une couche aux architectures multibases de données afin de gérer des transactions demandées par des UM. La principale différence est le choix du site du coordinateur. Dans MDSTPM la coordination de l'exécution des transactions mobiles est centralisée. L'UF coordinatrice est fixée à l'avance et elle ne change pas pendant toute l'exécution. Dans KT, la coordination est répartie entre toutes les SB visitées par l'UM. En effet, KT s'intéresse non seulement aux déconnexions mais aussi à la nature mobile des UM. Une coordination répartie réduit le coût d'exécution pendant l'exécution ; cependant, dans le cas d'une annulation en cascade, le coût de communication peut augmenter considérablement. A contrario, dans une coordination centralisée comme dans MDSTPM, l'annulation en cascade sera plus facile et mois coûteuse ; cependant, dans le cas de nombreux déplacements, la communication sera importante.

Une bonne analyse sur l'impact du déplacement sur les transactions demandées par des UM et exécutées sur des UF est faite dans [DK99]. Trois approches possibles pour la coordination des transactions sont analysées : (1) fixe sur l'UM, (2) fixe sur une UF, et (3) répartie entre les SB visitées. Par ailleurs, [DK98a] propose une définition de *donnée dépendante de la localisation*. Les auteurs analysent l'impact du mouvement sur ce type de données et leurs effets sur les propriétés ACID.

Dans l'informatique mobile, l'adaptabilité aux variations de l'environnement est une problématique importante. En plus de l'adaptabilité au *hand-off*, parmi les travaux analysés, seul Moflex s'intéresse aux transactions dépendantes de la localisation. Le principal inconvénient de Moflex est que la définition des transactions peut être compliquée. Des aspects concernant l'adaptabilité sont abordés dans [SA02, SARAL03].

La singularité de Pre-serialization est que les propriétés A/I sont abordées dans le contexte des multibases de données et tout en prenant en compte la déconnexion des UM. Cependant, l'algorithme PGSG est considéré comme coûteux à cause du principe de propagation (diffusion d'information sur la sérialisabilité) et car il doit être exécuté à deux reprises pendant la gestion des transactions globales (dans la version basique).

3.4 Conclusions

Depuis une dizaine d'années, plusieurs travaux de recherche et industriel ont été consacrés à la gestion de transactions dans les environnements mobiles. A la différence des environnements centralisés ou répartis, les environnements mobiles sont très versatiles et souffrent de plusieurs contraintes matérielles. En conséquence, l'exécution des transactions mobiles n'est pas prédictible et a besoin d'approches adaptées. Ces raisons motivent le développement de nouvelles approches autour des transactions mobiles.

Ce chapitre a analysé plusieurs travaux sur les transactions mobiles. Une analyse poussée sur les projets de recherche a été faite selon deux parties. La première partie se concentre sur la manière d'assurer ou de relâcher les propriétés ACID où des UM participent à l'exécution des transactions mobiles. Généralement, les travaux analysés dans cette partie ignorent le déplacement des UM. La deuxième partie aborde les travaux où les transactions sont demandées par des UM mais complètement exécutées sur des UF. Ces travaux prennent en compte le déplacement des UM lors de l'exécution des transactions. Quelques produits commerciaux sont analysés, cependant, du fait du manque d'information sur ces approches, l'analyse reste superficielle. L'ensemble des travaux abordés est analysé et résumé en plusieurs sections et tableaux.

Cette étude montre la difficulté d'assurer les propriétés ACID dans les environnements mobiles. Les principales raisons sont qu'il est nécessaire de préserver l'autonomie pour travailler en mode déconnecté ou faiblement connecté et que le déplacement est inhérent aux UM. La plupart des projets relâchent les propriétés ACID afin d'augmenter leur flexibilité et de gérer les contraintes de l'environnement mobile.

Du fait des variations de l'environnement mobile (localisation, debit de communication, etc.), l'adaptabilité est un aspect très important. Malgré le nombre important des travaux, l'adaptabilité transactionnelle aux variations du contexte reste une problématique ouverte. La plupart des travaux adaptent leur comportement pour supporter les modes d'exécution connecté et déconnecté. Très peu parmi eux considèrent l'adaptabilité au déplacement des

UM (hand-off). Cependant les variations de l'environnement ne se limitent pas qu'aux déconnexions et au hand-off. Il existe un grand éventail de variations qui peuvent affecter l'exécution des transactions mobiles.

Enfin, il est important de remarquer que l'exécution répartie des transactions mobiles où une UM participe à l'exécution a été très peu abordé et en plus, lorsqu'elle est supportée, souvent, les déconnexions ne sont pas permisses. Par ailleurs, l'exécution répartie où plusieurs UM participent à l'exécution n'a pas été abordée.

CHAPITRE 4

AMT : un modèle de transactions mobiles adaptables

Dans ce chapitre, nous proposons un modèle de transactions adaptables pour les environnements mobiles. Etant donnée l'importance de séparer l'aspect modèle de l'implantation, ce chapitre présente le modèle proposé, le chapitre 5 fait une étude analytique du modèle, enfin, le chapitre 6 présente les techniques de gestion appropriées pour son implantation.

Ce chapitre est organisé de la façon suivante. Dans une première section, nous présentons le modèle AMT et nous décrivons ses propriétés (section 4.1). Nous poursuivons avec une spécification formelle du modèle en ACTA (section 4.2). En fin de chapitre, nous concluons (section 4.3)

4.1 Le modèle AMT

Certains modèles transactionnels proposés dans le domaine des systèmes répartis et multibases, peuvent être applicables aux systèmes mobiles. Cependant, ils ne peuvent pas être utilisés directement, il est nécessaire de faire quelques adaptations afin de gérer, d'une manière convenable, les variations du contexte (cf. chapitre 2).

Cette nécessité a motivé de nombreuses propositions sur les transactions mobiles (cf. chapitre 3). Dans ces travaux, malgré l'éventail de caractéristiques de l'environnement mobile pouvant varier (cf. section 1.2.1), très souvent, seule la déconnexion des UM est prise en compte.

Dans notre travail, nous proposons l'*adaptation des transactions* comme une solution aux problèmes de variation de l'environnement mobile. En effet, l'adaptation de l'exécution des transactions mobiles au contexte peut permettre de diminuer les défaillances transactionnelles, tout en permettant de gérer les coûts d'exécution.

Nous proposons un modèle de transactions mobiles adaptables qui :

- permet de limiter le nombre de défaillances transactionnelles dues aux déconnexions des UM ;
- permet de s'adapter aux variations de l'environnement mobile afin de maîtriser les coûts d'exécution ;
- permet de définir des transactions mobiles pouvant utiliser les cinq modèles d'exécution ;
- limite le blocage des données du système ;
- est suffisamment général en ne ciblant pas des applications particulières.

4.1.1 Structure des transactions T_{AMT}

Le modèle de transactions mobiles adaptables, nommé AMT (*Adaptable Mobiles Transactions*) modélise des transactions mobiles nommées T_{AMT}. Une T_{AMT} est composée d'une ou plusieurs *alternatives d'exécution* tenant compte de l'environnement mobile. En effet, le modèle AMT permet de définir des transactions mobiles composées d'un ensemble d'alternatives d'exécution, chacune étant associée à un contexte particulier de l'environnement mobile.

Le modèle AMT peut être représenté par une structure d'arbre (cf. figure 4.1). La transaction racine T_{AMT} joue le rôle de coordinateur d'un ensemble d'alternatives d'exécution AE_k. Chaque AE_k coordonne un ensemble de *transactions composantes* t_{ki}. L'accès aux données se fait seulement par les transactions composantes, qui sont considérées comme les feuilles de l'arbre. Une T_{AMT} et ces AE_k sont seulement des unités de contrôle. Si on fait une analogie avec les multi-transactions, les t_{ki} sont des sous-transactions introduites par les AE_k d'une T_{AMT}.

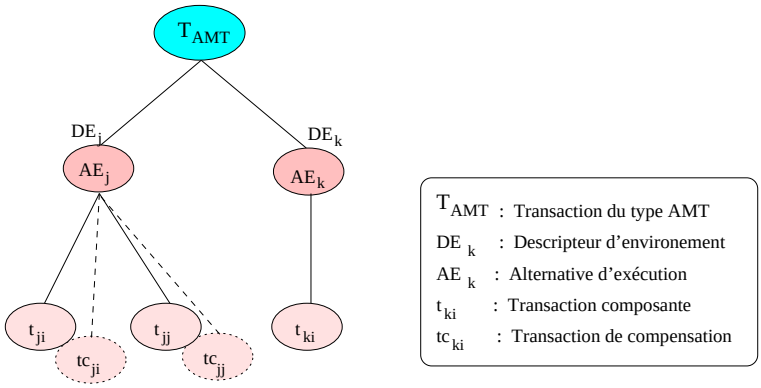

FIG. 4.1 – *Structure possible d'une* T_{AMT}

Dans la suite, nous présentons une définition intuitive du modèle AMT. La section 4.2 présente une spécification formelle en utilisant le formalisme ACTA.

Définition 4.1 *Une transaction mobile adaptable est une* $T_{AMT} = \langle AE_k \rangle$ *où :*

- $\langle AE_k \rangle$, $k > 0$, est une liste d'Alternatives d'exécution où AE_k a une priorité plus grande que AE_{k+1}.
- $AE_k = (DE_k, PE_k)$, une Alternative d'Exécution contient un plan d'exécution PE_k qui sera lancé si l'environnement mobile courant offre les caractéristiques spécifiées par le descripteur d'environnement DE_k.
- DE_k est un Descripteur d'Environnement sur lequel le plan d'exécution PE_k sera convenablement exécuté.
- $PE_k = \{(t_{ki}, tc_{ki}, SiteId)\}$, est un Plan d'Exécution contenant un ensemble de triplets dont chacun définit une transaction composante, sa transaction de compensation et le site sur lequel elles doivent être exécutées. Il existe une relation $\mathcal{RD}$ entre les éléments de PE_k. Ainsi,
  $\mathcal{RD}$ est une *relation de dépendance* sur PE_k,
  $\forall (t_{ki}, tc_{ki}, SiteId_x), (t_{kl}, tc_{kl}, SiteId_y) \in PE_k$ tel que
  $(t_{ki}, tc_{ki}, SiteId_x) \mathcal{RD} (t_{kl}, tc_{kl}, SiteId_y)$.

  SiteId indique le site (UM ou unité fixe) où t_{ki} doit s'exécuter.

Alternatives d'exécution. La définition montre qu'une T_{AMT} est composée d'une liste d'alternatives d'exécution $\langle AE_k \rangle$. Les différentes alternatives peuvent être sémantiquement équivalentes et l'exécution réussie de l'une d'entre elles représente l'exécution réussie de la T_{AMT}. En effet, chaque alternative représente une manière d'exécuter la T_{AMT}. Le descripteur d'environnement DE_k décrit les caractéristiques de l'environnement nécessaires pour exécuter le plan d'exécution de l'alternative. Le plan d'exécution PE_k d'une alternative contient, entre autres, une ou plusieurs transactions composantes à exécuter sur un ou plusieurs sites. Un site peut accueillir au plus une transaction composante par alternative.

Plan d'exécution PE_k. Le plan d'exécution PE_k est un ensemble dont chaque élément contient : une transaction composante, sa transaction de compensation et le site d'exécution de toutes les deux. Un plan d'exécution peut représenter différents modèles d'exécution (cf. section 2.1.2) :

1. exécution complète sur des unités fixes mais demandée par une UM ;
2. exécution complète sur une UM ;
3. exécution répartie entre une UM et des unités fixes ;
4. exécution répartie entre plusieurs UM ;
5. exécution répartie entre plusieurs UM et unités fixes.

Le fait de considérer tous ces modèles d'exécution offre une grande possibilité d'adaptation et une flexibilité permettant d'utiliser des T_{AMT} pour une grande variété d'applications mobiles.

Relation de dépendance $\mathcal{RD}$. $\mathcal{RD}$ peut être une *relation de dépendance* de parallélisme ($||$) ou de séquentialité ($<$) qui peut être appliquée aux éléments du plan d'exécution. Cette relation spécifie l'exécution parallèle ou séquentielle des transactions composantes du PE_k.

Le descripteur d'environnement DE_k. Le descripteur d'environnement DE_k spécifie l'environnement mobile (caractéristiques et son état) nécessaire pour exécuter le plan d'exécution qui lui est attaché. La section suivante (section 4.1.1.1) donne plus de détails à ce sujet.

Les transactions composantes t_{ki}. Les transactions composantes d'un plan d'exécution accèdent aux données des SGBD composant le système. Chaque transaction composante t_{ki} :

- est une transaction AID ; (la cohérence n'est pas imposée, elle dépend de la sémantique des alternatives d'exécution)
- est complètement exécutée par un SGBD sous-jacent ;
- peut être une transaction plate, répartie ou emboîtée fermée ;
- peut être associée à une transaction de compensation ;
- peut avoir une dépendance de valeur avec une autre transaction composante t_{kj}.

Le fait qu'une transaction composante soit une transaction plate, répartie ou emboîté fermée est transparent pour notre modèle. Comme nous le montrons plus tard (section 4.2.2.1), les propriétés du modèle AMT ne sont pas affectées pourvu que les transactions composantes préservent les propriétés AID.

Les transactions de compensation tc_{kj}. Afin de donner une certaine autonomie aux UM, nous proposons de relâcher l'atomicité globale des alternatives d'exécution en permettant la validation anticipée de transactions composantes (avant la validation globale). Cela motive l'utilisation de transactions de compensation (cf. section 4.1.2.1) tc_{kj} qui rétablissent sémantiquement la base de données lors de l'annulation de transactions composantes validées (cf. section 2.2.2.3). Les transactions de compensation sont exécutées en cas d'annulation d'une AE_k. Elles font une marche en arrière sémantique des transactions composantes validées en évitant les annulations en cascade. Notons que les transactions de compensation ne sont pas obligatoires. Une transaction composante peut ne pas être compensable, mais aussi dans certains cas, une compensation peut ne pas être nécessaire (par exemple, dans le cas des transactions de lecture).

La figure 4.2 illustre en UML la structure du modèle AMT. Nous pouvons voir qu'une AMT est composée d'une ou plusieurs alternatives d'exécution. Chaque alternative contient un descripteur d'exécution. Elle doit aussi avoir un plan d'exécution attaché, lequel doit avoir au moins une transaction composante t_{ki}. Une t_{ki} peut avoir ou non une transaction de compensation tc_{ki}.

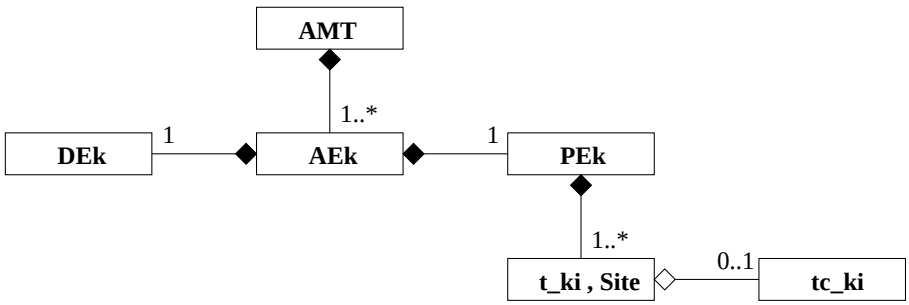

FIG. 4.2 – *Diagramme de la structure du modèle AMT*

De manière générale, lorsqu'une T_{AMT} est déclenchée, l'état courant de l'environnement mobile est examiné et l'alternative appropriée est démarrée. Si l'environnement mobile ne permet pas l'exécution d'une alternative, l'exécution de la T_{AMT} est reportée. Dans ce cas, une alternative sera démarrée aussitôt qu'un état acceptable de l'environnement mobile surviendra. Une AE_k peut annuler si une transaction composante t_{ki} annule ou si l'environnement change et le nouvel état ne satisfait pas le DE_k.

4.1.1.1 Le descripteur de l'environnement mobile DE_k

Les *descripteurs d'environnement* sont une abstraction de l'environnement mobile. En effet, les descripteurs sont une représentation des caractéristiques variables qui peuvent affecter l'exécution des transactions. L'ensemble de caractéristiques de l'environnement est spécifique au contexte de l'application. Il dépend du réseau sans fil utilisé, des UM ou encore des habitudes des utilisateurs. Par exemple, prendre en compte le débit communication peut être intéressant dans les réseaux UMTS mais pas dans les GSM où la bande passante est stable (cf. section 1.1). De la même façon, considérer le prix de communication peut être intéressant dans les réseaux où le prix varie selon l'endroit ou l'horaire de communication. Les applications dépendantes de la localité doivent également prendre en compte la localisation des utilisateurs.

L'environnement mobile peut être composé de plusieurs caractéristiques : le réseau sans fil, les unités mobiles et fixes participant aux transactions, la localité et les caractéristiques physiques des participants.

La définition des caractéristiques de l'environnement est très flexible afin de pouvoir satisfaire des besoins particuliers. Ces caractéristiques peuvent ou non concerner l'environnement mobile. Par exemple, la qualité de données requise (fraîcheur) peut être introduite comme l'une des caractéristiques du descripteur d'environnement. Les caractéristiques de l'environnement mobile que nous considérons sont liées à la communication sans fil et aux ressources des UM :

- connexion/déconnexion
- débit de communication

- prix de la communication
- capacité disponible de la batterie
- mémoire persistante et cache
- capacité de calcul
- temps de connexion prévu
- localité

Cet ensemble de caractéristiques est représenté dans le tableau 4.1. Chaque caractéristique est attachée à un ensemble d'**états**. Afin de simplifier, les états sont organisés en niveaux de qualité – *haut, moyenne, faible* –, cependant, ils peuvent être définis de façon particulière pour chaque caractéristique de l'environnement. Dans la pratique, ces états peuvent être traduits par des intervalles $[x, y]$. Par exemple, nous pouvons considérer que le prix-communication est *gratuit* s'il tombe dans l'intervalle $[0, 0]$, *modéré* dans $[0, 0.1]$ et *élevé* dans $[0.2, 0.4]$. Les caractéristiques des unités fixes ne sont pas prises en compte car nous les considérons comme des unités puissantes avec des ressources infinies où les variations sont considérées comme minimales.

	Caractéristique	**État**	**Unité**
RSF	état-connection	*connecté, déconnecté*	
	bande-passante	*forte, moyenne, faible*	Kbps
	prix-communication	*gratuit, modéré, élevé*	Euros/time
UM	batterie-disponible	*plaine, moitié, faible*	hh:mm:ss
	cache-disponible	*vide, moitié, plein*	Kbytes
	memoire-persistente-disponible	*vide, moitié, pleine*	kbytes
	capacité-calcul	*éleve, moyenne, faible*	Mhz/s
	temps-connection-estimé	*t*	hh:mm:ss
	localité	*l*	

TAB. 4.1 – *Caractéristiques de l'environnement mobile.*

Les descripteurs d'environnement (DE) indiquent les dimensions et leurs états à un moment donné de la façon suivante :

Définition 4.2 *Le Descripteur de l'Environnement (DE) contient un ensemble de caractéristiques avec leur(s) état(s) à un instant donné :*

DE = {Caractéristique = {état(s)}}

Ainsi, pour exécuter une transaction qui comprend un transfert de données important, l'environnement requis pourrait être :

Example 1 *DE = {état-connection = connecté, bande-passante = forte, coût-communication = {gratuit, modéré}}*

Si plusieurs UM participent à l'exécution d'une transaction, l'état requis pour chaque site peut être décrit ou non. Ainsi, pour chaque UM UM_x il y aurait un DE_x. Le descripteur pour une alternative d'exécution AE_k serait $DE_k = \{DE_x, DE_y, ...\}$.

Les concepteurs des transactions, qui connaissent bien l'application, sont les mieux placés pour déclarer le contexte d'exécution convenable à chaque transaction. Nous sommes conscients que ceci ajoute de la complexité au travail des concepteurs, néanmoins, pour le moment nous concentrons nos efforts pour fournir des concepts utilisables pour l'exécution des transactions dans les environnements mobiles. Dans le futur il serait nécessaire de développer des environnements pour fournir des outils spécifiques d'aide à la conception. Dans le chapitre 5, nous introduisons une étude analytique pouvant servir de guide pour le développement des transactions mobiles.

4.1.1.2 Exemple d'une transaction T_{AMT}

Afin de clarifier le modèle AMT, nous introduisons l'exemple suivant. Nous considérons un utilisateur avec une UM (avec des capacités de stockage) et un magasin électronique composé de deux sites serveurs, installés sur le réseau fixe – SCatalogue et SAchat. Le premier site permet d'interroger le catalogue du magasin tandis que le deuxième prend des commandes et gère les paiements.

Nous définissons une $T_{AMT_{achat}}$ contenant les transactions composantes suivantes :

- RécupereCatalogue permet aux clients de récupérer le catalogue du magasin.
- SélectionArticles permet aux clients de sélectionner des articles à partir d'une copie du catalogue.
- CommandePaiement permet aux clients d'envoyer une commande et de payer sur le serveur.
- PaiementLocal permet aux clients de payer sur l'UM, avec de l'argent électronique, sans communiquer avec les serveurs du magasin.
- Commande permet aux clients d'envoyer une commande (sans inclure le paiement).
- Sélection-PaiementLocal = SélectionArticles + PaiementLocal

Le tableau 4.2 introduit trois alternatives d'exécution qui utilisent les transactions composantes que nous venons de décrire. Une alternative est déclenchée si l'état de l'environnement mobile satisfait le descripteur d'environnement correspondant. Dans cet exemple, les caractéristiques du réseau mobile qui déterminent le choix d'une alternative sont : la disponibilité de la connexion, la bande passante et le prix de communication. La présence du catalogue sur l'UM est une caractéristique qui a été définie spécifiquement pour cette application. Cette caractéristique prend les valeurs *absent* (si le catalogue n'est pas sur l'UM), *présent* (si une version, probablement non à jour ou incomplète, existe dans l'UM) et à*jour* (si la dernière version est sur l'UM). Les caractéristiques qui n'ont pas d'importance pour cette application, n'apparaissent pas dans les descripteurs d'environnement.

Dans cet exemple, les alternatives d'exécution sont organisées selon leur coût d'exécution. L'exécution de AE_1 est moins cher que celle de AE_2. Dans AE_1, une version à jour du

AE_k	DE_k	PE_k
k=1	{état-catalogue= *àjour*}	{(SélectionArticles, UM) < (CommandePaiement, SAchat)}
k=2	{état-connection=*connecté*, bande-passante = *forte, moyenne*, prix-communication=*modéré* état-catalogue=*present, absent*},	{(RécupereCatalogue, SCatalogue) < (SélectionArticles, UM) < (CommandePaiement, SAchat)}
k=3	{état-connection=*connecté*, bande-passante=*faible*, état-catalogue=*absent*}	{(RécupereCatalogue, SCatalogue) < (Sélection-PaiementLocal, UM) < (Commande, SAchat)}

TAB. 4.2 – *Exemple* $T_{AMTachat}$.

catalogue existe sur l'UM, ceci permet d'économiser des messages. Il est possible de déclencher AE_1 en mode déconnecté et la transaction CommandePaiement peut être différée à une prochaine reconnexion. L'alternative AE_2 est déclenchée si la qualité de communication est acceptable (connexion avec une bande passante au moins *moyenne* et à un prix *modéré*). AE_3 s'exécute lorsque la qualité de communication est mauvaise. L'avantage de cette dernière alternative est que Sélection-PaiementLocale peut être faite en mode déconnecté car le paiement est sur l'UM. La transaction Commande est déclenchée lors d'une prochaine réconnexion.

La relation de dépendance $\mathcal{RD}$ entre les transactions composantes de chaque alternative est de séquentialité (<). Les transactions de compensation pour cet exemple peuvent être composées principalement d'opérations pour annuler et rembourser des commandes.

4.1.2 Les propriétés des transactions T_{AMT}

Cette section présente les propriétés du modèle AMT que nous considérons à trois niveaux différents : la T_{AMT}, les alternatives d'exécution et les transactions composantes. Les opérations *begin, abort, commit* sont considérées à chaque niveau :

begin : indique le début d'une T_{AMT} ($begin_{T_{AMT}}$), d'une alternative d'exécution AE_k ($begin_{AE_k}$) ou d'une transaction composante t_{ki} ($begin_{t_{ki}}$)

commit : indique la validation d'une T_{AMT} ($commit_{T_{AMT}}$), d'une alternative d'exécution AE_k ($commit_{AE_k}$) ou d'une transaction composante t_{ki} ($commit_{t_{ki}}$)

abort : indique l'annulation d'une T_{AMT} ($abort_{T_{AMT}}$), d'une alternative d'exécution AE_k ($abort_{AE_k}$) ou d'une transaction composante t_{ki} ($abort_{t_{ki}}$)

Rappelons que les transactions composantes doivent avoir des propriétés AID. Qui sont de la responsabilité des SGBD sous-jacents. Les sections qui suivent présentent les propriétés des alternatives d'exécution (cf. sections 4.1.2.1 et 4.1.2.2) et des T_{AMT} (cf. sections 4.1.2.3 et 4.1.3). Nous nous concentrons principalement sur les aspects atomicité, isolation et cor-

rection des transactions.

Une alternative d'exécution est une sorte de *Sagas* (cf. section 2.3.3) contenant un ensemble de transactions composantes dont l'exécution peut être répartie entre plusieurs unités mobiles et fixes. La relation de dépendance $\mathcal{RD}$ définie à l'intérieur des alternatives décrit la possibilité d'exécuter les composantes d'une façon séquentielle (similaire aux Sagas) ou parallèle. Dans la suite, nous introduisons les propriétés d'atomicité et d'isolation ainsi que le critère de correction des alternatives d'exécution.

4.1.2.1 Atomicité sémantique des AE_k

La validation des transactions composantes d'une alternative d'exécution AE_k, peut être faite de manière unilatérale mais aussi de manière coordonnée avec celle de l'AE_k. Le but est de permettre le plus d'autonomie aux SGBD sous-jacents tout en considérant le plus grand nombre d'applications possibles.

Pour les UM il est très important de pouvoir travailler d'une façon autonome qui permet un fonctionnement en mode déconnecté et ceci en bloquant le moins possible les ressources. Tenant compte des restrictions de l'environnement mobile nous proposons de relâcher la propriété d'atomicité. En effet, la *validation locale* (cf. section 3.2.1) des transactions composantes, qui est une approche optimiste, permet une certaine autonomie grâce à la libération prématurée des ressources. Ainsi, lorsqu'une transaction composante termine son exécution, les ressources utilisées localement peuvent être libérées (et partagées) avant la validation de l'alternative. Dans le cas où l'alternative annulerait, il est nécessaire d'utiliser des *transactions de compensation* qui annulent le travail fait par des transactions validées localement.

L'utilisation de transactions de compensation (cf. section 4.1.2.1) comme un moyen de récupération sémantique, donne comme résultat le relâchement de l'atomicité et offre une atomicité sémantique [GM83]. Dans une T_{AMT} seule les transactions compensables peuvent faire une validation locale optimiste. Les transactions de compensation des transactions composantes exécutées séquentiellement doivent s'exécuter dans l'ordre inverse de validation. Celles correspondant aux transactions de compensation exécutées en parallèle sont exécutées de façon concurrente.

Le relâchement de l'atomicité est bien approprié pour les transactions mobiles. Cependant, il n'est pas convenable de considérer seulement des transactions compensables. Ils existent des applications où il n'est pas possible de définir des transactions de compensation ou simplement la confidentialité de la transaction ne permet pas de relâcher prématurément les ressources utilisées. Ainsi, nous faisons une combinaison de l'approche optimiste et d'une approche pessimiste. Dans l'approche pessimiste, les ressources sont bloquées jusqu'à la validation (ou annulation) des alternatives. C'est-à-dire, lorsqu'une transaction composante termine, les ressources utilisées sont retenues jusqu'à ce qu'une décision globale (de validation ou annulation) soit prise. Avec cette mesure des transactions de compensation ne sont pas nécessaires.

Au niveau des alternatives d'exécution, nous adoptons la combinaison des deux approches que nous venons de décrire. Ceci donne le choix aux participants de faire une *validation locale optimiste* ou une validation *non-optimiste*. Dans le cas où aucun participant valide localement de manière optimiste, aucune transaction de compensation est nécessaire et on obtient

la *failure atomicity* (cf. section B.5.2).

Ainsi, une AE_k contenant un ensemble de t_{ki}, est **atomique sémantiquement** :

1. si AE_k valide, toutes les transactions t_{ki} doivent aussi valider ;
2. si AE_k annule, toutes les transactions t_{ki} doivent soit annuler soit compenser.

4.1.2.2 Ordonnancement global des AE_k

Nous considérons qu'il n'existe pas d'interférence entre les transactions composantes des différents SGBD sous-jacents car elles n'accèdent aucune donnée commune –les sources des différents SGBD sont considérées disjointes. Cependant, l'existence de dépendances de précédence entre les transactions composantes d'une même alternative oblige à préserver un ordre d'exécution aussi bien entre les transactions composantes t_{ki} qu'entre les AE_k concurrentes –appartenant à des T_{AMT} différentes.

Le critère utilisé pour contrôler la correction d'exécution des alternatives concurrentes est la *sérialisabilité globale* (cf. section 2.2.1). Nous considérons que sur chaque site, il existe une seule transaction composante par alternative et qu'il existe également des transactions locales au site hors celles issues des T_{AMT}.

Les AE_k sont **sérialisables globalement** si :

(1) toutes les transactions t_{ki} sont sérialisées localement et

(2) les AE_k son sérialisables dans le même ordre relatif sur chaque site.

Comme nous l'avons mentionné dans la section 4.1.1, les alternatives d'exécution d'une T_{AMT} peuvent être sémantiquement équivalentes. Une T_{AMT} est exécutée convenablement si une de ses alternatives est validée. Ceci nous conduit à la *semi-atomicité* [ZNBB94] des T_{AMT}.

4.1.2.3 Semi-atomicité des transactions T_{AMT}

La semi-atomicité d'une T_{AMT} est assurée, si (1) la validation d'une T_{AMT} implique la validation d'une seule AE_k et l'annulation ou la compensation des sous-transactions d'autres AE_l, si elles ont été démarrées, et (2) l'annulation de T_{AMT} implique l'annulation ou la compensation de toutes les sous-transactions de l'AE_k active.

1. Soit toutes les t_{ki} définis dans l'AE$_k$ valident et toutes les t_{li} d'une autre AE$_l$ démarrées dans la même T_{AMT} sont annulées ou compensées ;
2. soit aucun résultat partiel des transactions composantes t_{ki} de la T_{AMT} reste permanent dans les SGBD sous-jacents.

Il est important de souligner que l'atomicité sémantique des AE_k fait partie de la semi-atomicité. Autrement dit, pour assurer la semi-atomicité d'une T_{AMT}, il est nécessaire d'assurer l'atomicité sémantique des AE_k.

4.1.3 Critère de correction des transactions T_{AMT}

La sérialisabilité des T_{AMT} est fournie à travers celle des alternatives d'exécution. Cependant, même si un ordre global sérialisable est préservé, la cohérence sémantique est assurée. Ceci est dû au relâchement de l'atomicité et de l'isolation.

Afin d'obtenir une cohérence sémantique correcte lors de l'utilisation des transactions de compensation, nous adoptons les mesures suivantes :

- La validation partielle d'une T_{AMT} (valider une t_{ki} sur un site et annuler une autre t_{kj} sur un autre site) pourrait entraîner la violation des contraintes d'intégrité globales. Afin d'éviter ce problème, les contraintes d'intégrité globales ne sont pas considérées. Cette restriction ne pose pas de grands problèmes car nous considérons que les SGBD sous-jacents sont complètement indépendantes les uns des autres.
- Afin que l'exécution de tc_{ki} concerne seulement le site où t_{ki} est exécutée, les dépendances de valeur entre les transactions composantes compensables d'une même T_{AMT} sont interdites. Cela évite les annulations en cascade.

En ce qui concerne la propriété de durabilité, une fois q'une alternative valide (en conséquence sa T_{AMT}), la durabilité des transactions composantes est fournie par les SGBD sous-jacents.

Le tableau 4.3 résume les propriétés du modèle AMT pour tous les trois niveaux : t_{ki}, AE_k, T_{AMT}.

Propriété	t_{ki}	AE_k	T_{AMT}
Atomicité	$\surd$	Atomicité sémantique	Semi-atomicité
Cohérence	$\surd$ conditionnée	Cohérence sémantique	
Isolation	$\surd$	Relâchée (validation locale)	
Durabilité	$\surd$ conditionnée	SGBD sous-jacent	
Correction	Sérialisabilité	Sérialisabilité globale	

TAB. 4.3 – *Résumé des propriétés du modèle AMT.*

4.2 Spécification formelle du modèle AMT

Dans cette section, nous proposons une formalisation du modèle AMT en utilisant le formalisme ACTA. Dans l'annexe B, nous présentons ACTA, le but est de donner les concepts nécessaires pour la compréhension de cette section.

ACTA est un formalisme qui permet d'exprimer et de comparer les principales caractéristiques des modèles de transactions étendues. Plus spécifiquement, il permet de caractériser les effets que les transactions produisent les unes sur les autres ainsi que sur les objets qu'elles manipulent. Le pouvoir d'expression d'ACTA, ainsi que sa capacité d'extensibilité, sont largement suffisants pour spécifier les caractéristiques et propriétés de l'AMT.

La section est organisée comme suit : nous commençons par donner la définition axiomatique du modèle AMT (section 4.2.1). Afin d'obtenir les propriétés des transactions T_{AMT} tel qu'elles ont été définies dans la section 4.1.2, nous poursuivons avec une analyse et quelques déductions faites à partir des axiomes (section 4.2.2).

4.2.1 Définition axiomatique du modèle AMT

Les dépendances introduites dans la section B.3 ne sont pas suffisantes pour modéliser l'AMT. Puisque ACTA est extensible, nous proposons la dépendance suivante :

Dépendance de début unique (Unique Begin Dependency) $(t_j\ \mathcal{UBD}\ t_i) : t_i$ peut démarrer, si aucune autre t_j a démarré :
$(begin_{t_i} \in H) \Rightarrow \neg(begin_{t_j} \in H)$

Notation utilisée

Dans la suite, nous dénotons les éléments ainsi que les ensembles utilisés dans la définition axiomatique de l'AMT.

- T_{AMT} dénote une *Transaction Mobile Adaptable* contenant une liste d'alternatives d'exécution AE_k.

  $T_{AMT} = \langle AE_1, ..., AE_n \rangle,\ n > 0$

- Une alternative d'exécution AE_k est composée d'un ensemble de transactions composantes t_{ki}.

  $AE_k = \{t_{k1}, ..., t_{kn}\},\ n > 0$

  $AE_k \neq AE_l$

  Ici, nous faisons abstraction du descripteur d'environnement DE_k et du site d'exécution $SiteId$ décrit dans la définition 4.1.

- AE_k organise ses t_{ki} en deux ensembles, TC_k et TN_k, où TC_k contient les t_{ki} *compensables* et TN_k contient les t_{ki} *non-compensables*.

  $TC_k \cap TN_k = \phi$

  Les ensembles de transactions compensables et non-compensables doivent être disjoints.

- TS_k dénote la liste de t_{ki} qui s'exécutent de façon séquentielle :

  $(TS_k \subseteq TC_k) \vee ((t_{ki}, ..., t_{kn-1} \subseteq TC_k) \wedge (t_{kn} \in TN_k))$

  où $1 \leq i \leq n, t_{ki} \in TS_k$

  Seule les transactions compensables peuvent être exécutées de façon séquentielle à l'exception de la toute dernière.

- tc_{ki} dénote une *transaction de compensation* pour $t_{ki} : \forall t_{ki} \in TC_k\ \exists\, tc_{ki}$.

  Toutes les transactions compensables doivent avoir une transaction de compensation.

- t dénote t_{ki} ou tc_{ki}.

- S_u dénote l'ensemble de transactions qui s'exécutent sur le site u. AE_k a le droit d'exécuter une seule transaction composante par site.

Définition 4.3 *: Définition axiomatique du modèle AMT*

1. $ES_{T_{AMT}} = ES_{AE} = ES_t = \{\text{begin, commit, abort}\}$
2. $EI_{T_{AMT}},\ EI_{AE},\ EI_t = \{\text{begin}\}$
3. $ET_{T_{AMT}},\ ET_{AE},\ ET_t = \{\text{commit, abort}\}$
4. t satisfait les Axiomes fondamentaux I à IV
5. $Visibilité_{T_{AMT}} = \phi$
6. $Visibilité_{AE} = \phi$
7. $Visibilité_t = H^{(S_u)}$
8. $ConflitSet_{T_{AMT}} = \phi$
9. $ConflitSet_{AE} = \phi$
10. $ConflitSet_t = \{p_{t'}[ob] \mid t' \neq t, t', t \in S_u,\ Inprogress(p_{t'}[ob])\}$
11. $\forall ob\, \exists p\ (p_t[ob] \in H) \Rightarrow (ob$ est atomique)
    ob est *correct* et *sérialisable.*
12. $(commit_t \in H) \Rightarrow \neg(t\, \mathcal{C}^*\, t)$
    t peut valider *localement* si elle ne génère pas de cycles.
13. $\exists ob\, \exists p\ (commit_t[p_t[ob]] \in H) \Rightarrow (commit_t \in H)$
    La validation de $p_t[ob]$ implique la validation de t.
14. $(commit_t \in H) \Rightarrow \forall ob\, \forall p\ ((p_t[ob] \in H) \Rightarrow (commit_t[p_t[ob]] \in H))$
    La validation de t implique la validation de toutes ses opérations.
15. $\exists ob\, \exists p\ (abort_t[p_t[ob]] \in H) \Rightarrow (abort_t \in H)$
    L'annulation de $p_t[ob]$ implique l'annulation de t.
16. $(abort_t \in H) \Rightarrow \forall ob\, \forall p\ ((p_t[ob] \in H) \Rightarrow (abort_t[p_t[ob]] \in H))$
    L'annulation de t implique l'annulation de toutes ses opérations.
17. $(commit_{AE} \in H) \Rightarrow \neg(AE\, \mathcal{R}^*\, AE)$
    AE peut valider *globalement* que si elle ne génère pas de cycles.
18. $post(begin_{T_{AMT}}) \Rightarrow (((begin_{AE_k} \in H) \Rightarrow ConditionEnvironment) \wedge$
    $((AE_l\, \mathcal{UBD}\, AE_k) \in DepSet_{ct}) \wedge$
    $(t_{ki}\, \mathcal{BD}\, AE_k))$

    où $1 \leq k \leq m,\ 1 \leq l \leq m,\ k \neq l$

    Le début d'une AE_k est conditionné à la satisfaction d'une condition d'environnement. Dans une T_{AMT}, seule une AE_k doit être démarrée. Egalement, t_{ki} ne peut pas démarrer tant que son correspondante AE_k n'a pas démarré.

19. $post(begin_{AE_k}) \Rightarrow (((T_{AMT}\ \mathcal{AD}\ AE_k) \in DepSet_{ct}) \wedge$
$((AE_k\ \mathcal{AD}\ T_{AMT}) \in DepSet_{ct}) \wedge$
$(t_{ki} \in TS_k) \Rightarrow$
$((t_{ki}\ \mathcal{BCD}\ t_{ki-1}) \in DepSet_{ct}))$

où $1 \leq k \leq m$,

l'annulation d'AE_k implique l'annulation de T_{AMT} et vice versa. Les transactions composantes qui font parti de TS_k s'exécutent séquentiellement, toutes les autres peuvent s'exécuter d'une manière concurrente.

20. $post(begin_{t_{ki}}) \Rightarrow (((AE_k\ \mathcal{AD}\ t_{ki}) \in DepSet_{ct}) \wedge$
$((t_{ki} \in TC_k) \Rightarrow$
$((t_{ki}\ \mathcal{WD}\ AE_k) \in DepSet_{ct}) \wedge$
$((tc_{ki}\ \mathcal{BCD}\ t_{ki}) \in DepSet_{ct})) \wedge$
$((t_{ki} \in TN_k) \Rightarrow$
$((t_{ki}\ \mathcal{AD}\ AE_k) \in DepSet_{ct})))$

où $1 \leq i \leq n$, $1 \leq k \leq m$

L'annulation de t_{ki} implique l'annulation d'AE_k. Pour les transactions compensables, l'annulation d'AE_k implique l'annulation de t_{ki} (si t_{ki} n'a pas validé) et tc_{ki} ne peut démarrer que si t_{ki} valide. Pour les transactions non-compensables (TN_k), l'annulation d'AE_k implique l'annulation de t_{ki}. La validation des t_{ki} est retardée jusqu'à ce qu'AE_k valide (t_{ki} attend une annulation d'AE_k jusqu'à ce qu'AE_k valide).

21. $post(commit_{t_{ki}}) \Rightarrow (((tc_{ki}\ \mathcal{BAD}\ AE_k) \in DepSet_{ct}) \wedge$
$((tc_{ki}\ \mathcal{CMD}\ AE_k) \in DepSet_{ct}))$

où $1 \leq i \leq n$, $1 \leq k \leq m$

tc_{ki} peut démarrer si AE_k annule, et l'annulation d'AE_k, après la validation de t_{ki} implique la validation de tc_{ki}.

Types de dépendances utilisées

Ici, nous rappelons les types de dépendances utilisées dans les axiomes du modèle AMT. Ces dépendances sont un extrait de celles présentées dans l'annexe B.

Dépendance de début (Begin Dependency) $(t_j\ \mathcal{BD}\ t_i)$: t_j ne peut pas démarrer si t_i n'a pas démarré :
$(begin_{t_j} \in H) \Rightarrow (begin_{t_i} \rightarrow begin_{t_j})$.

Dépendance d'annulation (Abort Dependency) $(t_j\ \mathcal{AD}\ t_i)$: si t_i annule, alors t_j aussi :
$(abort_{t_i} \in H) \Rightarrow (abort_{t_j} \in H)$.

Dépendance début sur validation (Begin-on-Commit Dependency) $(t_j\ \mathcal{BCD}\ t_i)$: t_j ne peut pas démarrer jusqu'à ce que t_i valide :
$(begin_{t_j} \in H) \Rightarrow (commit_{t_i} \rightarrow begin_{t_j})$.

Dépendance d'annulation faible (Weak-Abort Dependency) $(t_j\ \mathcal{WD}\ t_i)$: si t_i annule et t_j n'a pas encore validé, t_j doit annuler aussi. En d'autres termes, si t_j valide et t_i annule, la validation de t_j doit précéder l'annulation de t_i dans l'historique :
$(abort_{t_i} \in H) \Rightarrow (\neg(commit_{t_j} \rightarrow abort_{t_i}) \Rightarrow (abort_{t_j} \in H))$.

Dépendance début sur annulation (Begin-on-Abort Dependency) $(t_j\ \mathcal{BAD}\ t_i)$: t_j ne peut pas démarrer jusqu'à ce que t_i annule :
$(begin_{t_j} \in H) \Rightarrow (abort_{t_i} \rightarrow begin_{t_j})$.

Dépendance de compensation (Force-Commit-on-Abort Dependency) $(t_j\ \mathcal{CMD}\ t_i)$: si t_i annule, t_j doit valider :
$(abort_{t_i} \in H) \Rightarrow (commit_{t_j} \in H)$.

4.2.2 Analyse et déductions sur les axiomes

Dans cette section, nous analysons les axiomes de la définition 4.3 afin d'en déduire les propriétés des transactions T_{AMT}.

Les Axiomes 1-3 de la définition 4.3 indiquent les événements significatifs de l'AMT sur les trois niveaux. L'Axiome 4 dit que les transactions composantes ainsi que les transactions de compensation (t) doivent respecter les axiomes fondamentaux. Les Axiomes 5-6 montrent que les T_{AMT} et les AE_k sont des points de contrôle et n'accèdent pas aux données. Par conséquent, dans 8-9 il n'existe pas de conflits possibles. L'Axiome 7 montre que la visibilité de t est limitée à la projection de l'historique H sur S_u (l'ensemble de transactions qui s'exécutent sur le site u). De ce fait, dans l'Axiome 10, l'ensemble des conflits de t est composé potentiellement de toutes les opérations en cours faites par d'autres transactions dans S_u.

Avant d'entrer dans plus de détails, nous tenons à souligner la particularité de l'Axiome 18. La dépendance $((begin_{AE_k} \in H) \Rightarrow ConditionEnvironment)$, introduit une condition de démarrage pour AE_k. L'objectif de cette condition est de limiter le démarrage de l'alternative dès la satisfaction de $ConditionEnvironment$. $ConditionEnvironment$ devient vraie lorsque le *descripteur d'environnement* (DE) de l'AE_k (voir section 4.1.1.1) coïncide avec l'état actuel de l'environnement mobile.

La suite de la section est organisée de la manière suivante. Dans une première section, nous montrons les propriétés des transactions composantes et de compensation (section 4.2.2.1), ensuite nous déduisons l'atomicité sémantique (section 4.2.2.2) et la sérialisabilité globale des AE_k (section 4.2.2.3). Nous terminons par la propriété de semi-atomicite des T_{AMT} (section 4.2.2.4).

Dans l'annexe A, nous expliquons les déductions faites dans le présent chapitre.

4.2.2.1 Les propriétés des t_{ki} et tc_{ki}

Dans cette section, nous montrons que les transactions composantes ainsi que de compensation sont atomiques (Lemme 4.2) donc avec les propriétés AID.

Lemme 4.1 *: Si* $t_i \in T_{AMT}$, t_i ***est failure atomic***

Preuve du Lemme 4.1 : t_i est *failure atomic* si elle satisfait les deux conditions de la définition B.5.

1. La condition 1 (la clause du tout) est dérivée des axiomes 13 et 14.
2. La condition 2 (la clause du rien) est dérivée des axiomes 15 et 16.

Lemme 4.2 *: Si* $t_i \in T_{AMT}$, t_i ***est une transaction atomique***

Preuve du Lemme 4.2 : t_i est une transaction atomique si elle satisfait les deux conditions du Théorème B.1

1. La condition 1 (*failure atomic*) est dérivée du Lemme 4.1.
2. La condition 2 (sérialisable) est dérivée des axiomes 11 et 12.

Puisque les transactions atomiques satisfont les propriétés AID (cf. Théorème B.1), alors les transactions composantes ainsi que de compensation sont des transactions AID.

Les transactions atomiques, réparties et emboîtées

Les transactions composantes peuvent être des transactions plates (atomiques) mais aussi des transactions réparties et emboîtées (cf. section 4.1.1) puisqu'elles satisfont les propriétés AID.

D'après le Théorème B.2, une transaction répartie est atomique (*setwise failure atomicity*) et sérialisable.

D'après le Théorème B.3, une transaction emboîtée est sérialisable. D'après la spécification introduite dans [Chr91], la racine d'une transaction emboîtée est *failure atomic* (Axiomes 11-14 et Lemme 4.4 dans [Chr91]).

Ainsi :

- Les transactions atomiques, les transactions réparties, ainsi que la racine des transactions emboîtées assurent la propriété d'atomicité. [1]
- Les transactions atomiques, les transactions réparties et emboîtées sont sérialisables. Donc,
- Une T_{AMT} peut être composée par des transactions composantes atomiques, réparties ou emboîtées.

1. *Failure atomicity* et *setwise failure atomicity* assurent l'atomicité d'une ou de plusieurs transactions respectivement.

4.2.2.2 L'atomicité sémantique d'une AE_k

Dans cette section, nous montrons que les alternatives d'exécution ont la propriété d'atomicité sémantique (cf. section 4.1.2.1).[2] Nous commençons par définir la validation d'une AE_k (Lemme 4.3) ainsi que son annulation (Lemme 4.4). Ensuite, nous obtenons l'atomicité sémantique d'une AE_k (Lemme 4.5).

Lemme 4.3 *: La validation d'une* AE_k

Soit H l'historique d'une alternative d'exécution AE_k *avec n transactions composantes.*

$((commit_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n \ (commit_{t_{ki}} \in H))$

La validation d'une alternative d'exécution AE_k, implique la validation de toutes les transactions composantes t_{ki} associées.

Preuve du Lemme 4.3 : Si AE_k valide, son ensemble de transactions composantes doit valider grâce à la dépendance d'annulation d'AE_k sur t_{ki} de l'Axiome 20 (la première) et à l'Axiome fondamental III :

$\forall i, 1 \leq i \leq n \ ((abort_{t_{ki}} \in H) \Rightarrow (abort_{AE_k} \in H)) \Leftrightarrow$
$((commit_{AE_k} \in H) \Rightarrow (commit_{t_{ki}} \in H))$

Lemme 4.4 *: L'annulation d'une* AE_k

Soit H l'historique d'une AE_k *avec n transactions composantes.*

$(abort_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n, \ \forall j, 1 \leq j \leq n, i \neq j$

$((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{tc_{kj}}))$

L'annulation d'AE_k implique l'annulation ou la compensation des transactions composantes associées.

Preuve du Lemme 4.4 :

Cas 1. Si AE_k annule lorsque t_{ki} est en cours, t_{ki} annule grâce aux dépendances $\mathcal{WD}$ ou $\mathcal{AD}$ de t_{ki} sur AE_k (Axiome 20). Grâce à l'Axiome fondamental II, il n'est pas nécessaire de spécifier que seules les transactions initiées sont annulées. Egalement, grâce à l'axiome fondamental III, il n'est pas nécessaire de spécifier que seule les transactions non validées sont annulées. Grâce à la dépendance $\mathcal{BCD}$ de tc_{ki} sur t_{ki} (Axiome 20) tc_{ki} ne démarre pas dans ce cas :

$(abort_{AE_k}) \Rightarrow \forall i, 1 \leq i \leq n \ (abort_{t_{ki}} \in H)$

Seules les $t_{ki} \in TC_k$ peuvent valider grâce à la dépendance $\mathcal{WD}$. Par contre, les $t_{ki} \in TN_k$ valident jusqu'à ce qu'AE_k valide grâce à la dépendance d'annulation de t_{ki} sur AE_k. Ainsi, toutes les $t_{ki} \in TN_k$ en cours seront annulées lors de l'annulation de l'AE_k.

2. L'expression de l'atomicité sémantique est similaire à celle des transactions Sagas [CR92].

Cas 2.

1. Si AE_k annule après que t_{ki} valide et avant que une t_{kj} démarre, tc_{ki} doit valider grâce à la dépendance $\mathcal{CMD}$ de l'Axiome 21 :
   $(abort_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq m\ (commit_{tc_{ki}})$
2. Étant donnée l'existence de la dépendance $\mathcal{BCD}$ de tc_{ki} sur t_{ki} dans l'Axiome 20, si tc_{ki} valide alors t_{ki} a du valider :
   $(begin_{tc_{ki}} \in H) \Rightarrow (commit_{t_{ki}} \rightarrow begin_{tc_{ki}})$
   Par l'Axiome fondamental II :
   $(commit_{tc_{ki}} \in H) \Rightarrow (begin_{tc_{ki}} \rightarrow commit_{tc_{ki}})$
   Ainsi, par la sémantique de la relation de dépendances, si tc_{ki} valide, elle le fait après t_{ki} :
   $(commit_{tc_{ki}} \in H) \Rightarrow (commit_{t_{ki}} \rightarrow commit_{tc_{ki}})$
3. A partir de 1 et 2 :
   $((abort_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq m\ (commit_{tc_{ki}})) \wedge$
   $((commit_{tc_{ki}} \in H) \Rightarrow (commit_{t_{ki}} \rightarrow commit_{tc_{ki}}))$
   Plus simplement :
   $(abort_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n (commit_{t_{ki}} \rightarrow commit_{tc_{ki}})$

Cas 3. Si une AE_k annule lorsqu'une t_{ki} est en cours (Cas 1) et après qu'une t_{kj} a validé (Cas 2) :

$(abort_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$
$((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{tc_{kj}}))$

Avant de montrer que les AE_k ont la propriété d'atomicité sémantique, nous définissons l'atomicité sémantique des alternatives d'exécution comme suit.

Définition 4.4 *: **Atomicité sémantique d'une** AE_k*

Chaque AE_k assure l'atomicité sémantique :

1. *si AE_k valide, toutes les transactions t_{ki} doivent aussi valider et*

2. *si AE_k annule, toutes les transactions t_{ki} doivent soit annuler soit compenser.*

Autrement dit :

1. $(commit_{AE_k} \in H) \Rightarrow \forall i,\ 1 \leq i \leq n\ (commit_{t_{ki}} \in H)$

2. $(abort_{AE_k} \in H) \Rightarrow$

   $\forall i,\ 1 \leq i \leq n\ (\neg(commit_{t_{ki}} \rightarrow abort_{AE_k}) \Rightarrow (abort_{t_{ki}} \in H)) \wedge$

   $((commit_{t_{ki}} \rightarrow abort_{AE_k}) \Rightarrow (commit_{t_{ki}} \rightarrow commit_{tc_{ki}})))$

Où tc_{ki} est une transaction de compensation pour t_{ki}.

Plus simplement:

$(abort_{AE_k} \in H) \Rightarrow$

$\forall i,\ 1 \leq i \leq n\ \forall j,\ 1 \leq j \leq n\ j \neq i\ ((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{tc_{kj}}))$.

Lemme 4.5 *: **Chaque** AE_k **assure l'atomicité sémantique***

Chaque AE_k assure l'atomicité sémantique en satisfaisant les deux conditions de la définition 4.4.

Preuve du Lemme 4.5 :

1. La condition 1 de la définition 4.4 (si AE_k valide, toutes les transactions t_{ki} doivent aussi valider) est satisfaite par le Lemme 4.3.
2. La condition 2 de la définition 4.4 (si AE_k annule, toute les transactions t_{ki} doivent soit annuler soit compenser) est satisfaite par le Lemme 4.4.

4.2.2.3 La sérialisabilité globale des AE_k

Dans cette section, nous montrons que les alternatives d'exécution sont séralisables globalement (cf. section 4.1.2.2). En effet, à partir de certains Axiomes de la définition 4.3, nous déduisons la sérialisabilité globale des AE_k (Lemme 4.6).

Nous commençons par définir la sérialisabilité globale d'un ensemble d'AE_k.

Soit t une transaction locale t_i ou composante t_{ki}.
Soit G l'ensemble des AE_k définies pour différentes T_{AMT}.
Soit S_u l'ensemble de transactions t qui s'exécutent sur le site u.
Soit $S_{u_{commit}}$ le sous-ensemble de S_u qui contient les transactions t validées.
Soit $\mathcal{R}$ une relation binaire sur G.

Définition 4.5 *: **Sérialisabilité globale des** AE_k*

Un ensemble d'AE_k assurent la sérialisabilité globale:

1. *si l'ordre d'exécution des transactions composantes assure un ordre sérialisable dans chaque site, et*
2. *si les AE_k assurent également un ordre sérialisable sur tous les sites.*

Autrement dit :

$\forall AE_k, AE_l \in G,\ AE_k \neq AE_l$

$(AE_k\ \mathcal{R}\ AE_l)$ *si*

$\exists S_u,\ \exists t_{ki} \in S_{u_{commit}}$, t_{ki} *composante de* AE_k, $\exists t_{li} \in S_{u_{commit}}$, t_{li} *composante de* AE_l, $\exists t_i \in S_{u_{commit}}$, $\exists ob, ob_1\ \exists p, q$

$(conflit(p_{t_{ki}}[ob], q_{t_{li}}[ob]) \wedge (p_{t_{ki}}[ob] \rightarrow q_{t_{li}}[ob])) \vee$

$(conflit(p_{t_{ki}}[ob], q_{t_i}[ob]) \wedge (p_{t_{ki}}[ob] \rightarrow q_{t_i}[ob])) \wedge$

$(conflit(p_{t_i}[ob_1], q_{t_{li}}[ob_1]) \wedge (p_{t_i}[ob_1] \rightarrow q_{t_{li}}[ob_1]))$

Un ensemble d'alternatives d'exécution est globalement sérialisable si :

1. $\forall u\ \forall t \in S_{u_{commit}}\ (commit_t \in H) \Rightarrow \neg(t\ \mathcal{C}^*\ t)$

2. $\forall AE_k \in G\ (commit_{AE_k} \in H) \Rightarrow \neg(AE_k\ \mathcal{R}^*\ AE_k)$

Lemme 4.6 *: Les* AE_k ***assurent la sérialisabilité globale***

Preuve du Lemme 4.6 : Pour assurer la sérialisabilité globale les deux conditions de la définition 4.5 doivent être satisfaites :

1. La condition 1 est dérivée de l'Axiome 12.
2. La condition 2 est dérivée de l'Axiome 17.

4.2.2.4 La semi-atomicité des T_{AMT}

Dans cette section, nous montrons que les T_{AMT} ont la propriété de semi-atomicité (cf. section 4.1.2.3). Nous commençons par introduire la validation des T_{AMT} (Lemmes 4.7 et 4.8) ainsi que leur annulation (Lemmes 4.9 et 4.10). Ensuite, nous obtenons la semi-atomicité des T_{AMT} (Lemme 4.11).

Afin d'obtenir ces propriétés, nous définissons d'abord la validation (complète) d'une T_{AMT} (Lemme 4.8). Pour y arriver nous montrons la validation (simple) d'une T_{AMT} (Lemme 4.7) ainsi que celle d'une AE_k (Lemme 4.3). De la même façon, nous définissons l'annulation d'une T_{AMT} (Lemmes 4.10, 4.9 et 4.4).

Lemme 4.7 *: La validation d'une* T_{AMT}

Soit H l'historique d'une T_{AMT} *et* AE_k *une alternative d'exécution associée à* T_{AMT}*.*

$((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ (commit_{AE_k} \in H))$

Ce lemme exprime que si l'historique comprend la validation de T_{AMT} alors il contient aussi celle d'une alternative associée k.

Preuve du Lemme 4.7 :

1. Si T_{AMT} valide, AE_k doit aussi valider grâce à la dépendance d'annulation de T_{AMT} sur AE_k établie dans l'Axiome 19 (la première) et à l'Axiome fondamental III, qui dit qu'une transaction doit valider ou annuler :

   $\forall k, 1 \leq k \leq m \; ((abort_{AE_k} \in H) \Rightarrow (abort_{T_{AMT}} \in H)) \Leftrightarrow$
   $((commit_{T_{AMT}} \in H) \Rightarrow (commit_{AE_k} \in H))$

2. Une seule AE_k valide grâce à la dépendance $\mathcal{UBD}$ de l'Axiome 18 où une seule AE doit démarrer :

   $\forall k, \; 1 \leq k \leq m, \; \forall l, \; 1 \leq l \leq m, \; l \neq k$

   $(begin_{AE_k} \in H) \Rightarrow \neg(begin_{AE_l} \in H)$

3. A partir de 1 et 2

   $((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m \; (commit_{AE_k} \in H))$

Lemme 4.8 *: La validation complète d'une* T_{AMT}

Soit H l'historique d'une T_{AMT} *avec n transactions composantes et* AE_k *une alternative d'exécution associée à* T_{AMT}.

$((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m \; (commit_{AE_k} \in H)) \wedge$

$((commit_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n \; (commit_{t_{ki}} \in H))$

Plus simplement :
$(commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m \; \forall i, 1 \leq i \leq n, \; (commit_{t_{ki}} \in H)$

Preuve du Lemme 4.8 : Ce lemme découle des Lemmes 4.7 et 4.3.

Lemme 4.9 *: L'annulation d'une* T_{AMT}

Soit H l'historique d'une T_{AMT} *et* AE_k *une alternative d'exécution associée à* T_{AMT}.

$((abort_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m \; ((abort_{AE_k} \in H))$

Ce lemme exprime l'historique dans lequel l'annulation d'une T_{AMT} entraîne celle de l'AE_k en cours d'exécution.

Preuve du Lemme 4.9 :

1. Si T_{AMT} annule, AE_k doit aussi annuler grâce à la dépendance d'annulation d'AE_k sur T_{AMT} de l'Axiome 19 (la seconde) :

   $(abort_{T_{AMT}} \in H) \Rightarrow (abort_{AE_k} \in H)$

2. Une seule AE_k annule grâce à la dépendance $\mathcal{UBD}$ de l'Axiome 18 où une seule AE_k doit démarrer, voir 2 dans Lemme 4.7

Lemme 4.10 *: L'annulation complete d'une* T_{AMT}

Soit H l'historique d'une T_{AMT} *avec n transactions composantes et* AE_k *une alternative d'exécution associée à* T_{AMT}.

$((abort_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m((abort_{AE_k} \in H)) \wedge$

$(abort_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$

$((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{tc_{kj}}))$

Plus simplement :
$(abort_{T_{AMT}} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$
$((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{tc_{kj}}))$

Ce lemme exprime l'historique dans lequel, l'annulation d'une T_{AMT} entraîne l'annulation de l'AE_k en cours et la compensation ou l'annulation des transactions composantes associées à AE_k.

Preuve du Lemme 4.10 : Ce lemme découle des Lemmes 4.9 et 4.4.

Grâce à l'analyse que nous venons de faire, nous pouvons dire que :

Théorème 4.1 *: L'exécution d'une* T_{AMT} ***produit l'un des historiques suivants :***

1. $((commit_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m\ (commit_{AE_k} \in H)) \wedge$
   $((commit_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n\ (commit_{t_{ki}} \in H))$
2. $((abort_{T_{AMT}} \in H) \Rightarrow \exists k, 1 \leq k \leq m((abort_{AE_k} \in H)) \wedge$
   $(abort_{AE_k} \in H) \Rightarrow \forall i, 1 \leq i \leq n,\ \forall j, 1 \leq j \leq n, i \neq j$
   $((abort_{t_{ki}} \in H) \wedge (commit_{t_{kj}} \rightarrow commit_{tc_{kj}}))$

Preuve du Théorème 4.1 : Ce théorème découle des Lemmes 4.8 et 4.10.

Avant de prouver la propriété de semi-atomicité des T_{AMT}, nous introduisons la définition de semi-atomicité.

Définition 4.6 *: Semi-atomicité d'une* T_{AMT}

Chaque T_{AMT} *assure la semi-atomicité si :*

1. *la validation d'une* T_{AMT} *implique la validation d'une seule* AE_k *et l'annulation ou la compensation des sous-transactions d'autres* AE_l, *et*

2. *l'annulation de T_{AMT} implique l'annulation ou la compensation de toutes les sous-transactions de l'AE_k active.*

Autrement dit:

1. $((commit_{T_{AMT}} \in H) \Rightarrow \exists k,\ 1 \leq k \leq m\ (commit_{AE_k} \in H)) \wedge$
   $((commit_{AE_k} \in H) \Rightarrow \forall i,\ 1 \leq i \leq n\ (commit_{t_{ki}} \in H)) \wedge$
   $\forall l, 1 \leq l \leq m,\ l \neq k\ ((abort_{AE_l} \in H) \Rightarrow \forall i,\ 1 \leq i \leq n\ (\neg(commit_{t_{li}} \rightarrow commit_{T_{AMT}}) \Rightarrow$
   $(abort_{t_{li}} \in H)) \wedge ((commit_{t_{li}} \rightarrow commit_{T_{AMT}}) \Rightarrow (commit_{t_{li}} \rightarrow commit_{tc_{li}})))$

2. $((abort_{T_{AMT}} \in H) \Rightarrow \exists k,\ 1 \leq k \leq m\ (abort_{AE_k} \in H)) \wedge$
   $((abort_{AE_k} \in H) \Rightarrow \forall i,\ 1 \leq i \leq n\ (\neg(commit_{t_{ki}} \rightarrow abort_{T_{AMT}}) \Rightarrow (abort_{t_{ki}} \in H)) \wedge$
   $((commit_{t_{ki}} \rightarrow abort_{AE_k}) \Rightarrow (commit_{t_{ki}} \rightarrow commit_{tc_{ki}})))$

Lemme 4.11 *: **Chaque T_{AMT} assure la semi-atomicité***

Chaque T_{AMT} assure la semi-atomicité en satisfaisant les deux conditions de la définition 4.6.

Preuve du Lemme 4.11 :

1. La condition 1 de la définition 4.6 (toutes les transactions dans AE_k valident) est satisfaite par le Lemme 4.8.

   L'annulation et/ou la compensation des sous-transactions d'autres AE_l n'est pas nécessaire grâce à la dépendance $\mathcal{UBD}$ de l'Axiome 18.

2. La condition 2 de la définition 4.6 (toutes les transactions dans T_{AMT} sont annulées ou compensées) est satisfaite par le Lemme 4.10.

Théorème 4.2 *: **Chaque T_{AMT} a les propriétés suivantes :***

(1) t_i a les propriétés AID,

(2) AE_k assure l'atomicité sémantique,

(3) AE_k assure la sérialisabilité globale.

(4) T_{AMT} assure la semi-atomicité,

Preuve (1) découle du Lemme 4.2, (2) découle du Lemme 4.5, (3) découle du Lemme 4.6 et (4) découle du Lemme 4.11.

Définition 4.7 *Le schéma de gestion d'une T_{AMT} est correct s'il est conforme à la définition 4.3.*

4.3 Conclusion

Dans ce chapitre, nous avons présenté le modèle de transactions mobiles adaptables (AMT). L'AMT est un modèle générique qui ne cible pas d'application particulière. Nous avons commencé par décrire le modèle d'une manière intuitive : sa structure, ses principes et ses propriétés. Ensuite, nous avons donné une spécification formelle du modèle AMT en utilisant ACTA [CR94].

Les principales caractéristiques d'une transaction T_{AMT} sont les suivantes :

- c'est une transaction emboîtée ouverte ;
- elle est associée à la description du contexte d'exécution ;
- elle peut correspondre aux cinq modèles d'exécution (cf. section 4.1.1) ;
- elle permet la définition de plusieurs alternatives d'exécution ;
- elle relâche l'atomicité et l'isolation ;
- elle offre la cohérence sémantique.

Notre principale contribution est le fait de considérer, dans la définition de la transaction, la description du contexte d'exécution. Ceci permet de limiter le coût d'exécution des transactions. Lors du lancement de la transaction, on compare son descripteur de contexte avec l'état courant de l'environnement mobile. S'ils coïncident, la transaction est déclenchée sinon soit elle peut attendre soit elle est annulée.

Par ailleurs, le fait de considérer plusieurs alternatives d'exécution pour une même transaction apporte l'adaptation au contexte. Chaque alternative a un descripteur d'environnement associé. Lors du lancement de la transaction, on choisit l'alternative la plus adéquate selon le contexte. L'exécution réussie d'une alternative d'exécution est considérée comme l'exécution réussie de la T_{AMT}.

Dans le chapitre 2, nous avons constaté que les modèles de transactions étendues peuvent être applicables au contexte mobile. Le modèle AMT s'inspire principalement des Sagas (cf. section 2.3.3), de DOM (cf. section 2.3.4) et des Flexibles (cf. section 2.3.5) :

- Concernant la validation unilatérale des sous-transactions, le principe des Sagas facilite l'exécution de transactions en mode déconnecté sans entraîner le blocage du système. Dans le cas d'une annulation globale, la compensation des sous-transactions validées évite les annulations en cascade.

- Les transactions Flexibles sont conçues comme un ensemble de sous-transactions fonctionnellement équivalentes. Les DOM permettent d'associer aux sous-transactions des transactions de contingence. Ces deux idées, nous ont permis de concevoir l'adaptation au contexte en utilisant des alternatives d'exécution.

Dans le chapitre précédent, nous avons montré que les propositions actuelles sur les transactions mobiles se concentrent sur l'exécution d'une transaction : (1) sur une unité mobile, (2) sur des unités fixes ou (3) entre une unité mobile et des unités fixes. Ce dernier cas est considéré uniquement en mode connecté. Nos contributions par rapport à ces travaux sont :

- l'utilisation de transactions réparties (4) entre plusieurs unités mobiles et fixes mais aussi (5) entre unités mobiles ;
- l'utilisation des cinq types d'exécution ;
- l'exécution des transactions mobiles en mode connecté et déconnecté.

Comme la plupart des modèles transactionnels étendus (multibases et mobiles), l'AMT relâche la propriété d'atomicité et d'isolation.

Notre approche ajoute une certaine complexité dans la définition des transactions car il faut décrire le contexte d'exécution. En effet, savoir identifier le contexte le plus adéquat pour l'exécution d'une transaction n'est pas simple. D'une part, il faut avoir une connaissance précise de l'environnement mobile disponible pour l'application, d'autre part, il faut savoir quelles sont les variations possibles. En étant conscient de cette problématique, dans le chapitre suivant, nous faisons une étude analytique du modèle AMT qui permet de modéliser le contexte d'exécution et ses variations.

Le modèle AMT a été motivé par les variations des environnements mobiles cellulaires, cependant, il peut être également appliqué à d'autres contextes variables comme les réseaux *ad hoc* ou les contextes pair à pair.

CHAPITRE 5

Étude analytique du modèle AMT

L'objectif de ce chapitre est, d'une part, montrer que l'utilisation de transactions de type AMT augment la probabilité de réussite et limite les coûts d'exécution des transactions. D'autre part, donner une idée de comment représenter l'environnement mobile et comment concevoir les transactions T_{AMT}. L'intérêt du dernier point est né du fait que lorsqu'on désire exploiter au maximum les avantages d'adaptation offertes par le modèle AMT, la définition des transactions peut devenir complexe.

Ce qui est présenté dans ce chapitre provient d'un travail collectif. Nous remercions Cyril Labée qui est à l'origine du modèle analytique pour sa participation à cet aspect de notre recherche.

Le chapitre est organisé de la façon suivante. Dans la section 5.1, nous utilisons un modèle probabiliste afin d'analyser plusieurs alternatives d'exécution une par une – séparément du modèle AMT. Pour chacune d'entre elles, nous évaluons la probabilité d'être déclenchée et le coût d'exécution. La section 5.2 souligne les bénéfices obtenus grâce à l'adaptabilité de l'AMT et grâce à la perception de l'environnement mobile. Entre autres, nous montrons que la probabilité de déclenchement d'une T_{AMT} est toujours plus grande que celle d'une alternative d'exécution. Enfin, nous discutons le fait que le modèle AMT permet de définir des T_{AMT} en fonction du critère de qualité de l'application b : réduction des coûts, optimisation de temps de réponses, etc.

5.1 Les alternatives d'exécution

Dans cette section, nous introduisons la matrice qui décrit les caractéristiques de l'environnement mobile (section 5.1.1), ensuite nous montrons comment obtenir la probabilité de déclenchement d'une alternative dans un certain état de l'environnement (section 5.1.2). Nous montrons comment définir une matrice de coût pour l'environnement mobile (section 5.1.3), nous utilisons cette matrice pour calculer le coût moyen des dimensions d'une alternative d'exécution (section 5.1.4) ainsi que le coût moyen de chacune des alternatives (section

		$j=1$ Bonne	$j=2$ Moyenne	$j=3$ Mauvaise
$i=1$	état-connexion	*connecté*		*déconnecté*
$i=2$	bande-passante	*forte*	*moyenne*	*faible*
$i=3$	prix-communication		*modéré*	*élevé*
$i=4$	état-catalogue	*àjour*	*présent*	*absent*

TAB. 5.1 – *Environnement mobile considéré pour* $T_{AMTachat}$.

5.1.5).

Un environnement mobile peut être défini par un ensemble de dimensions avec ses états correspondants (cf. section 4.1.1.1). Tout au long de ce chapitre, nous utilisons l'exemple $T_{AMTachat}$ introduit dans la section 4.1.1.2. Le tableau 5.1 montre l'environnement considéré pour cette transaction mobile. Les lignes correspondent aux dimensions (caractéristiques) de l'environnement. Les colonnes aux différents états que chaque dimension peut avoir à un instant donné. Les états des dimensions sont organisés selon leur niveau de qualité, la première colonne représente une bonne qualité, la deuxième une qualité moyenne et ainsi de suite. D'autres niveaux de qualité peuvent être introduits, les niveaux utilisés dans le tableau 5.1 le sont à titre d'exemple. Les cases vides indiquent que la correspondante caractéristique de l'environnement n'a pas un état pour ce niveau de qualité.

5.1.1 Matrice du descripteur d'environnement

Une T_{AMT} est une transaction mobile adaptable composée d'un ensemble d'alternatives d'exécution (AE_k). Ici nous montrons comment chaque alternative spécifie l'environnement nécessaire pour son exécution.

Définition 5.1 *Pour chaque* AE_k*=(*DE_k*,* PE_k*) nous dénotons par* Δ^k *la matrice booleenne où :*

$$\delta_{ij}^k = \begin{cases} 1 & si\ l'etat\ j\ de\ la\ dimension\ i\ est\ acceptable\ pour\ AE_k \\ 0 & autrement \end{cases}$$

Dans l'environnement mobile, il existe une matrice de référence (Δ^R) qui considère tous les états possibles j des dimensions considérées i. Dans l'environnement mobile considéré ici, la matrice ($\Delta^{AMTachat}$) représente la matrice de référence. Cette matrice doit correspondre aux dimensions et montrées dans le tableau 5.1.

$$\Delta^R = \Delta^{AMTachat} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Exemple 5.1 *Les matrices Δ^k qui correspondent aux trois alternatives de $T_{AMTachat}$ (cf. Tableau 4.2) sont :*

$$\Delta^1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \Delta^2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \Delta^3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Si une dimension i n'apparaît pas dans DE_k, n'importe quel état est accepté : si $i \notin DE_k$ alors $\forall j$, si $\delta^R_{ij} = 1$ alors $\delta^k_{ij} = 1$. Δ^1 et Δ^3 illustrent ce cas. Dans Δ^1, la seule dimension spécifiée est état-catalogue = à*jour* ($\delta^1_{41} = 1$). Dans Δ^3, la dimension prix-communication n'est pas considérée, $\delta^3_{3j} = 1$, sauf $\delta^3_{31} = 0$ car cet état est à 0 dans la matrice de référence $\Delta^{AMTachat}$.

5.1.2 Probabilité de déclenchement d'une AE_k

Une fois l'environnement mobile identifié et décrit, nous pouvons définir la probabilité que l'état courant de l'environnement coïncide avec les différents états des dimensions considérées.

Définition 5.2 *p_{ij} est la probabilité de la dimension i d'être dans l'état j.*

Dans la matrice qui en résulte $P = (p_{ij})$, $\forall i,\ \sum_j p_{ij} = 1$. Dans l'exemple qui suit, nous montrons que cette matrice dépend de l'environnement mobile considéré. Autrement dit, P est défini d'après le réseau mobile et les unités mobiles utilisés ou encore selon les habitudes des utilisateurs.

Exemple 5.2 *La matrice de probabilité d'un environnement comme celui que nous considérons dans la $T_{AMTachat}$ peut être :*

$$P_1 = (p_{ij}) = \begin{bmatrix} 0.8 & 0 & 0.2 \\ 0.7 & 0.2 & 0.1 \\ 0 & 0.4 & 0.6 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

Dans cet exemple, la probabilité d'être *connecté* est donnée par p_{11}, la probabilité d'avoir un catalogue à*jour* est donnée par p_{41} et ainsi de suite. p_{31} a une probabilité 0 car dans l'environnement considéré, la communication ne peut pas être gratuite. L'état-connection =

connecté a une probabilité de 0.8 et le **bande-passante** = *forte* de 0.7 ce qui signifie que les utilisateurs ont, la plupart du temps, une bonne qualité de communication.

Définition 5.3 *Soit q^k la probabilité pour que AE_k soit sélectionnée pour être exécutée. q^k sera appelée la probabilité de déclenchement d'AE_k. Comme la probabilité que la dimension i ait un état acceptable pour AE_k est donnée par $\sum_j \delta_{ij}^k p_{ij}$, nous avons :*

$$q^k = \prod_i (\sum_j \delta_{ij}^k p_{ij})$$

La probabilité de déclenchement d'AE_k est positive ($q^k > 0$), si et seulement si $\forall i, \exists j$ tel que $\delta_{ij}^k = 1$. Autrement dit, AE_k a une opportunité d'être initiée si pour chaque dimension i il existe un état j acceptable.

Comparaison des AE_k

Exemple 5.3 *Dans $T_{AMTachat}$, la probabilité d'être sélectionnée pour chacune des alternatives est de :*

$$q^1 = (0.8 + 0.2) * (0.7 + 0.2 + 0.1) * (0.4 + 0.6) * (0.2) = 0.2$$

$$q^2 = (0.8) * (0.7 + 0.2) * (0.4) * (0.3 + 0.5) = 0.23$$

$$q^3 = (0.8) * (0.1) * (0.4 + 0.6) * (0.5) = 0.04$$

Exemple 5.4 *Afin d'étudier les AE_k dans de différents types d'environnement, nous nous proposons d'analyser les indices de performance en fonction de la probabilité que* **bande-passante** = *faible (p_{23}):*

$$P_2 = \begin{bmatrix} 0.8 & 0 & 0.2 \\ (1-p_{23})/2 & (1-p_{23})/2 & p_{23} \\ 0 & 0.4 & 0.6 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}$$

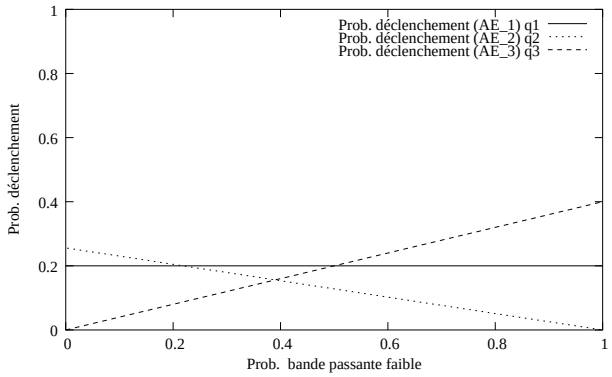

FIG. 5.1 – *Probabilité de déclenchement vs bande passante (AE_k)*

Remarquer la différence entre P_1 et P_2. Dans P_2 nous faisons varier la bande passante.

La figure 5.1 montre la probabilité de déclenchement des AE_k. Nous pouvons voir que si la bande passante est souvent *faible* (par exemple $p_{23} = 0.8$), l'alternative EA_3 a la probabilité la plus forte d'être initiée. q^1 est constante car elle ne dépend pas de la probabilité de bande-passante = *faible* mais uniquement de état-catalogue = à*jour*.

Voici les probabilités de déclenchement des trois alternatives avec $P_{23} = 0.8$:

$$q^1 = (0.8 + 02) * (01 + 0.1 + 0.8) * (0.4 + 0.6) * (0.2) = 0.2$$

$$q^2 = (0.8) * (0.1 + 0.1) * (0.4) * (0.3 + 0.5) = 0.05$$

$$q^3 = (0.8) * (0.8) * (0.4 + 0.6) * (0.5) = 0.32$$

5.1.3 Matrice de coût

Afin de calculer le coût d'exécution d'une alternative nous définissons ici une matrice de coût. Cette matrice est conçue en associant un coût aux dimensions et états considérés dans le tableau 5.1.

Définition 5.4 *C^k est une matrice où c^k_{ij} est le coût d'exécution d'AE_k dans l'état j de la dimension i.*

c^k_{ij} doit être défini par le programmeur de l'application en fonction des besoins d'optimisation des coûts.

Exemples de coût

Pour la transaction $T_{AMTachat}$ nous pourrions identifier l'utilisation de la mémoire comme un coût associé à la dimension état-catalogue ou la consommation du CPU comme un coût associé à la dimension état-connection, en considérant qu'en mode déconnecté plus d'opérations sont faites sur l'unité mobile.

On choisit ici de ne considérer qur des coûts liés au prix de communication et à l'utilisation de la batterie qui peuvent respectivement être associés aux dimensions prix-communication et bande-passante. Une bande passante réduite accroît la consommation de batterie car le temps d'exécution augmente. Le réseau sans fil utilisé est UMTS. C'est un réseau à commutation par paquets où la bande passante peut varier théoriquement de 144 Kbps (avec une mobilité véhiculaire), 384 Kbps (mobilité à pied) à 2 Mbps à l'intérieur d'un bâtiment. Ces débits de communication correspondent aux états *forte, moyenne, faible* de la bande passante. Le prix de communication dépend du volume des données transmise (paquets). Par exemple, l'exécution d'AE_1 a besoin de trois messages logiques transmis par le réseau sans fil. Tout d'abord, il y a un message d'enregistrement (*login*), ensuite, il y a un autre message utilisé pour passer la commande d'achat (transaction composante CommandePaiement), finalement, un message est reçu par l'UM pour confirmer la commande (acquittement). Les exécutions d'AE_2 et d'AE_3 ont besoin de 2 messages supplémentaires. Un pour demander le catalogue et un deuxième pour son transfert (transaction composante RécupereCatalogue).

Nous identifions trois types de messages : petits (login, acquittement et demande de catalogue), moyen (pour CommandePaiement et Commande) et grands messages (pour RécupereCatalogue). Nous considérons que les messages petits, moyens et grands sont respectivement composés par 1, 10 et 20 paquets. Ainsi, si le plan d'exécution d'AE_k comprend n_s petits, n_m moyens et n_l grands messages, alors le nombre total de paquets envoyés ou reçus [1] par l'UM pendant l'exécution est $n_p = n_s + 10n_m + 20n_l$.

Afin d'estimer le coût de communication (prix de communication et consommation de batterie) nous considérons que :

- envoyer un seul paquet dans l'état *modéré* (respective *élevé*) coûte une unité de prix (respective deux unités).
- si le la bande passante est forte (respective *moyenne, faible*) envoyer/recevoir un paquet utilise 0.1% (respective 0.2%, 0.4%) de la capacité de la batterie. Dans ce cas, le coût associé à la dimension bande-passante est la consommation de la batterie.

Avec ces considérations nous avons :

$$C^k = \begin{bmatrix} 0 & 0 & 0 \\ 0.1n_p & 0.2n_p & 0.4n_p \\ 0 & n_p & 2n_p \\ 0 & 0 & 0 \end{bmatrix}$$

1. Éventuellement, il pourrait être intéressant de distinguer l'envoie et la réception des messages. Pour l'UM il est moins coûteux de recevoir des messages que de les envoyer.

où c^k_{2j} est la consommation de la batterie pour l'exécution d'AE_k, lorsque **bande-passante** est dans l'état j. Le coût c^k_{2j} résulte de la multiplication de la consommation de la capacité de la batterie – $0.1\%, 0.2\%, 0.4\%$ – (sous les différents états de la capacité de bande passante) par le nombre de paquets envoyés/reçus n_p.

c^k_{3j} est le prix de l'exécution d'AE_k lorsque le **prix-communication** est dans l'état j. Autrement dit, c^k_{3j} est la multiplication du prix de communication (1 et 2 unités de prix, sous les états *modéré, élevé*) par le nombre de paquets envoyés/reçus n_p.

Dans $T_{AMTachat}$, $n_p = 2 + 10 + 0 = 12$ pour AE_1 et $n_p = 3 + 10 + 20 = 33$ pour AE_2 et AE_3, ainsi les coûts d'exécution des trois alternatives est :

$$C^1 = \begin{bmatrix} 0 & 0 & 0 \\ 1.2 & 2.4 & 4.8 \\ 0 & 12 & 24 \\ 0 & 0 & 0 \end{bmatrix} \quad C^2 = C^3 = \begin{bmatrix} 0 & 0 & 0 \\ 3.3 & 6.6 & 13.2 \\ 0 & 33 & 66 \\ 0 & 0 & 0 \end{bmatrix}$$

5.1.4 Coût moyen des dimensions dans une AE_k

Lorsque l'alternative d'exécution k est lancée par le système, elle s'exécute dans l'état j de la dimension i avec la probabilité :

$$\frac{\delta^k_{ij} p_{ij}}{\sum_j \delta^k_{ij} p_{ij}}$$

Ainsi, le coût moyen associé à la dimension i de l'exécution de AE_k est donné par :

$$c^k_i = \frac{\sum_j \delta^k_{ij} c^k_{ij} p_{ij}}{\sum_j \delta^k_{ij} p_{ij}}$$

Exemple 5.5 *Avec P_1, Δ^k et C^k nous pouvons calculer:*

$$c^1_2 = \frac{0.7 * 1.2 + 0.2 * 2.4 + 0.1 * 4.8}{0.7 + 0.2 + 0.1} = 1.8\%$$

$$c^1_3 = \frac{0.4 * 12 + 0.6 * 24}{0.4 + 0.6} = 19.2 \textit{ unités de prix}$$

$$c^2_2 = \frac{0.7 * 3.3 + 0.2 * 6.6}{0.7 + 0.2} = 4.03\%$$

$$c^2_3 = \frac{0.4 * 33}{0.4} = 33 \textit{ unités de prix}$$

$$c^3_2 = \frac{13.2 * 0.1}{0.1} = 13.2\%$$

$$c_3^3 = \frac{33 * 0.4 + 66 * 0.6}{0.4 + 0.6} = 52.8 \; \textit{unités de prix}$$

Ces calculs mettent en évidence le fait que AE_1 est l'alternative la plus avantageuse, elle a le prix et la consommation de batterie les plus réduits (en moyenne). L'alternative la plus chère est AE_3.

La figure 5.2 illustre la consommation de batterie associée à chaque AE_k dans $T_{AMTachat}$ en utilisant la matrice de probabilité P_2 de l'exemple 5.4). Il est possible de voir que si la bande passante est souvent *faible* (par exemple $p_{23} = 0.8$) alors AE_3 est l'alternative avec la consommation de batterie plus haute (13.2).

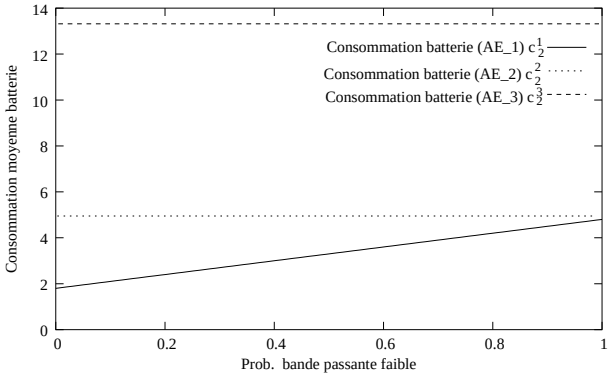

FIG. 5.2 – *Consommation de batterie vs bande passante (AE_k)*

5.1.5 Coût moyen d'une AE_k

Le fait d'avoir associé un coût aux dimensions des alternatives nous permet de construire un coût associé à chaque AE_k. Ceci peut être donné par la fonction suivante.

Définition 5.5 *Notons c^k le coût moyen d'exécution d'AE_k. c^k est calculé en fonction des coûts moyens associés aux dimensions de l'environnement considéré pour AE_k : $c^k = f(c_1^k, ..., c_i^k, ...)$.*

Cette fonction f peut être choisie par le concepteur de la T_{AMT} selon ses intérêts de coûts particuliers : temps de communication, consommation de batterie, prix de communication, etc.

Exemple 5.6 *Dans $T_{AMTachat}$ nous pouvons considérer que le coût moyen de chaque AE_k est donné par $c^k = c_2^k + c_3^k$. Ainsi, $c^1 = 21$, $c^2 = 37.03$, et $c^3 = 66$.*

Exemple 5.7 *Afin de montrer un exemple plus intuitif, considérons une transaction T_{AMTx} où c_i^k est le temps moyen d'exécution de l'alternative k et c_j^k le prix moyen par unité de temps. Ainsi, le coût économique (prix) moyen de chaque AE_k est donné par $c^k = c_i^k * c_j^k$.*[2]

5.2 Étude analytique d'une T_{AMT}

Cette section analyse le déclenchement d'une transaction T_{AMT} (section 5.2.1), le coût moyen de ses dimensions (section 5.2.2) et le coût moyen général (section 5.2.3).

La perception de l'environnement mobile permet au système de choisir une AE_k si l'état de l'environnement correspond au DE_k associé. Puisque AE_k a une priorité[3] plus grande que AE_{k+1} nous pouvons considérer, sans perte de généralité que :

Propriété 5.1 *Si l'état de l'environnement est adéquat pour une AE_k il ne l'est pas pour une $AE_{k'}$ de la même T_{AMT}. C'est-à-dire : $\forall(k,k'), k \neq k', \exists i$ tel que $\forall j, \delta_{ij}^{k'} \neq \delta_{ij}^{k}$*

Exemple 5.8 *A titre d'exemple, rappelons la matrice Δ^1 et modifions légèrement Δ^2 (cf. exemple 5.1) :*

$$\Delta^1 = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad \Delta^2 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ \boldsymbol{1} & 1 & 1 \end{bmatrix}$$

Si AE_1 est plus prioritaire qu'AE_2, alors AE_2 ne sera jamais exécutée lorsque l'état de l'environnement sera : état-connection=*connecté*, bande-passante=*forte, moyenne*, prix-communication = *modéré* et état-catalogue = *àjour*.

5.2.1 Probabilité de déclenchement d'une T_{AMT}

Définition 5.6 *Soit q_{AMT} la probabilité de déclenchement de T_{AMT}. Grâce à la propriété 5.1, nous avons : $q_{AMT} = \sum_k q^k$.*

Ainsi, en utilisant la matrice P_1 (cf. section 5.1.2) :

$$q_{AMT} = q^1 + q^2 + q^3 = 0.2 + 0.23 + 0.04 = 0.47$$

2. Cet exemple est applicable lors de l'utilisation des réseaux où la facturation dépend du temps de connexion qui dépend de la qualité de la bande passante.

3. Les priorités peuvent être désignées selon : le coût d'exécution, la qualité des réponses, la probabilité de déclenchement, etc. des AE_k.

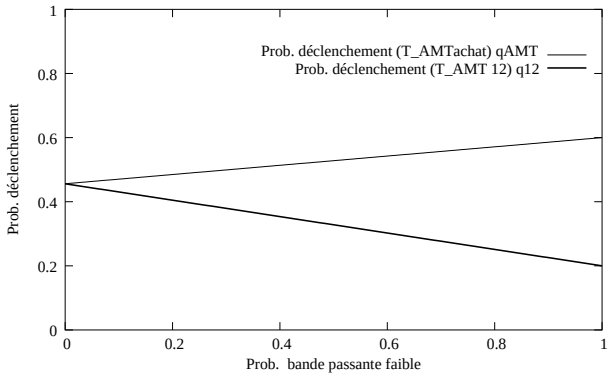

FIG. 5.3 – *Probabilité de déclenchement vs bande passante (deux T_{AMT})*

En utilisant la matrice P_2 de l'exemple 5.4, la figure 5.3 montre que la probabilité de déclenchement de $T_{AMTachat}$ n'atteint jamais 1. Ceci est dû au fait que pour certains états il n'y pas d'alternative défini. Par exemple, si l'UM est *déconnecté* avec l'état du catalogue *absent*, aucune alternative ne peut être initiée (voir δ^k_{13} et δ^k_{43} dans les matrices $\Delta^1, \Delta^2, \Delta^3$ de l'exemple 5.1). La figure 5.4 montre que la probabilité de déclenchement des AE_k est plus basse que la probabilité de déclenchement de $T_{AMTachat}$.

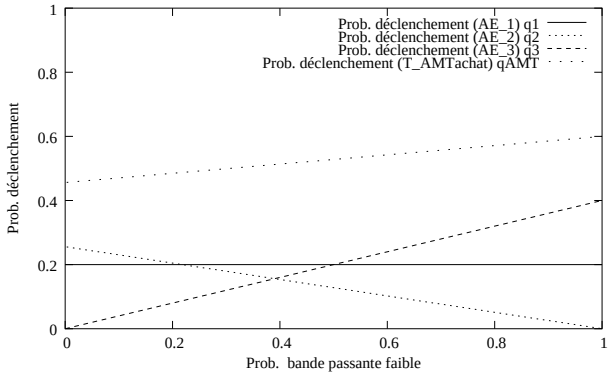

FIG. 5.4 – *Probabilité de déclenchement vs bande passante (T_{AMT} et AE_k)*

A titre d'exemple, nous étudions la probabilité de déclenchement de $T_{AMTachat}$ en fonction de la probabilité que la bande passante soit *faible* (p_{23}). Nous faisons varier p_{23} de 0 à 1. La figure 5.3 illustre les paramètres de performance pour $T_{AMTachat}$ ainsi qu'une variante sans AE_3. Cette nouvelle transaction est appelée T_{AMT12}. Nous pouvons voir que $T_{AMTachat}$ a une meilleure probabilité de déclenchement. Ceci est dû au fait qu'elle dispose d'une AE_k supplémentaire (AE_3).

5.2.2 Coût moyen des dimensions d'une T_{AMT}

Définition 5.7 *Notons c_i le coût moyen, associé à la dimension i, de l'exécution de T_{AMT}. En considérant que l'environnement est stable pendant l'exécution, AE_k est initiée avec une probabilité q^k, donc le coût de la T_{AMT} par rapport à une dimension est c_i^k, avec une probabilité q^k, d'où :*

$$c_i = \frac{\sum_k c_i^k q^k}{\sum_k q^k}$$

Exemple 5.9 *En considérant P_1 et les résultats des exemples 5.3 et 5.5, le coût de $T_{AMTachat}$ par rapport à la consommation de batterie est :*

$$c_2 = \frac{1.8 * 0.2 + 4.03 * 0.23 + 13.2 * 0.04}{0.2 + 0.23 + 0.04} = 3.86$$

Le coût de $T_{AMTachat}$ par rapport au prix d'exécution est :

$$c_3 = \frac{19.2 * 0.2 + 33 * 0.23 + 52.8 * 0.04}{0.2 + 0.23 + 0.04} = 28.81$$

A titre d'exemple, nous étudions la consommation moyenne de la batterie de $T_{AMTachat}$ en fonction de la probabilité que la bande passante soit *faible* (p_{23} dans P_2).

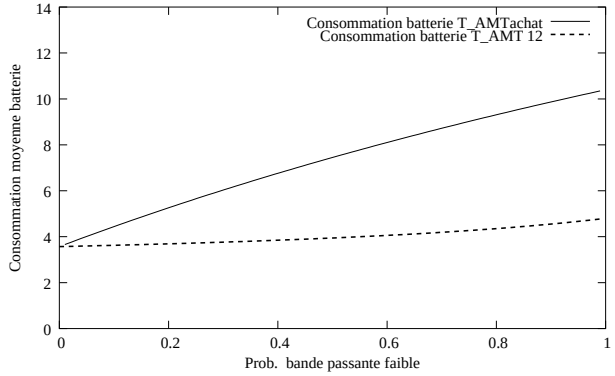

FIG. 5.5 – *Consommation de batterie vs bande passante (deux T_{AMT})*

La figure 5.5 montre que la consommation de batterie moyenne de $T_{AMTachat}$ augmente avec la probabilité que la bande passante soit *faible*. Ceci est dû au fait que la probabilité

de déclenchement d'AE_3 est plus grande que celle d'AE_2 (cf. figure 5.1). T_{AMT12} a une meilleure moyenne de consommation de batterie car elle ne comporte pas AE_3 qui a un coût très élevé.

5.2.3 Coût moyen d'une T_{AMT}

Le coût d'exécution d'une T_{AMT} peut être calculé en fonction des coûts moyens associés aux dimensions (cf. section 5.2.2) mais aussi en fonction des coûts moyens associés aux alternatives d'exécution (cf. section 5.1.5).

Définition 5.8 *C^D_{AMT} est le coût moyen d'exécution d'une T_{AMT} calculé en fonction des coûts moyens associés aux dimensions de l'environnement considéré pour cette T_{AMT} :* $C^D_{AMT} = f(c_1, ..., c_i)$.

Définition 5.9 *C^A_{AMT} est le coût moyen d'exécution de T_{AMT} calculé en fonction des coûts moyens associés aux alternatives d'exécution considérées pour cette T_{AMT} :* $C^A_{AMT} = f(c^1, ..., c^i)$.

Ces fonctions doivent être définies par le programmeur de l'application selon ses intérêts de coûts particuliers.

Exemple 5.10 *En suivant l'exemple 5.6, considérons que* $C^A_{AMT} = c^1 + c^2 + c^3$. *Ainsi,* $C^A_{T_{AMTachat}} = 21 + 37.03 + 66 = 124.03$

Exemple 5.11 *De la même manière, en suivant l'idée de l'exemple 5.7, considérons que* $C^A_{AMT} = c^i + c^j$.

Exemple 5.12 *En suivant l'exemple 5.9, considérons que* $C^D_{AMT} = c_2 * c_3$. *Ainsi,* $C^D_{T_{AMTachat}} = 3.86 * 28.81 = 111.21$

5.3 Conclusion

Traditionnellement, le système exécute une transaction quel que soit l'état de l'environnement. Si l'environnement ne permet pas une exécution correcte d'une transaction, celle-ci est annulée même si une autre façon de l'exécuter aurait pu réussir. De façon similaire, les différents états de l'environnement génèrent des coûts différents. Permettre au système de choisir le type d'exécution en fonction de l'état de l'environnement, est une façon de borner le coût d'exécution.

Ce chapitre montre que l'approche suivie dans le modèle AMT – l'adaptabilité à l'environnement dans l'exécution des transactions mobiles – améliore les performances et permet de choisir la façon dont les transactions s'exécutent en fonction des coûts d'exécution : prix de communication, consommation des ressources, temps d'exécution, etc.

Le type d'analyse fait dans ce chapitre permet de faire des estimations liées aux variations d'état de l'environnement mobile et à l'exécution des transactions type AMT (T_{AMT}). D'une part, la modélisation de l'environnement mobile permet de calculer la probabilité de déclenchement des T_{AMT} et de ses alternatives d'exécution (AE_k). D'autre part, la modélisation des coûts d'exécution de différentes alternatives dans les différents états de l'environnement mobile, permet d'établir un compromis sur les différents critères de qualité. Par exemple, avec un environnement mobile qui est souvent dans un état générant des coûts de communication élevés, le critère de qualité de l'application peut être (1) optimisation des coûts de communication en dépit de la qualité de résultats produits ou (2) optimisation des temps de réponse en dépit des coûts de communication.

Avec n alternatives d'exécution disponibles, le concepteur d'applications peut choisir entre plusieurs combinaisons différentes pour une même T_{AMT}. Avec $n = 1$ (une alternative) on a n combinaisons différentes pour une même T_{AMT}, avec $n = 2$ (deux alternatives) on a $n * (n-1)$ et ainsi de suite jusqu'à $n!$, soit au totale : $n + n * (n-1) + n * (n-1) * (n-2) + n * (n-1) * (n-2) * (n-3) + ... + n!$

Il faut souligner que la conception de la T_{AMT} qui donne les meilleures probabilités de déclenchement ou les meilleurs coûts d'exécution ne dépend pas du nombre d'alternatives définis mais de la capacité de couvrir les variations d'état de l'environnement.

En conclusion, l'adaptabilité aux variations d'état de l'environnement mobile offre l'opportunité d'améliorer les performances et les coûts d'exécution. Cependant, la conception de transactions adaptables peut s'avérer très compliquée. Pour faire face à ce nouveau degré de complexité, des outils d'aide à la conception basés sur le type d'analyse fait dans ce chapitre peuvent être définis.

CHAPITRE 6

Mise en œuvre du modèle AMT : l'intergiciel TransMobi

Dans ce chapitre, nous introduisons TransMobi, un intergiciel conçu pour mettre en œuvre le modèle AMT proposé dans le chapitre 4. L'objectif de notre approche est de faciliter l'accès transactionnel aux bases de données faiblement couplés, localisées sur des unités mobiles et fixes, tout en rendant transparent l'aspect mobile des clients ou des sources de données (SGBD ou serveurs).

Ce chapitre est organisé de la façon suivante. Nous commençons par donner une vue générale de l'intergiciel TransMobi (section 6.1). Ensuite, nous parlons de son architecture à trois tiers (section 6.2) et de la perception de l'environnement (section 6.3). Nous continuons avec l'introduction les techniques utilisées pour fournir les propriétés du modèle AMT (section 6.4) dont le protocole de validation CO2PC. Finalement, nous parlons de la décomposition fonctionnelle de chaque tiers de TransMobi (section 6.5) et du prototype implanté (section 6.6).

6.1 Vue générale

Les principales fonctions d'un moniteur transactionnel sont : la gestion de l'accès sécurisé aux données, la distribution de charge et la coordination de l'exécution et la validation des transactions réparties. TransMobi se concentre sur cette dernière fonction. Il contrôle l'exécution et la validation de transactions mobiles T_{AMT} qui s'adaptent aux variations de l'environnement mobile.

TransMobi est un intergiciel situé entre le code applicatif et des SGBD existants (cf. figure 6.1). Les SGBD sous-jacents et les applications peuvent être localisés sur des unités mobiles et/ou fixes. TransMobi suit une architecture de type *client-agent-serveur* répartie entre des unités mobiles et fixes.

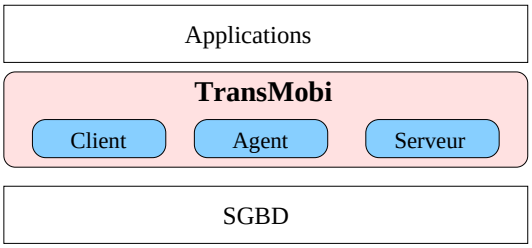

FIG. 6.1 – *Vue globale d'un système mobile*

La capacité de gestion des SGBD considérés est très variable. Ils peuvent être très puissants (généralement localisés sur des unités fixes) ou très légers (généralement localisés sur des unités mobiles). L'existence de plusieurs "SGBD légers" commerciaux – comme FastObjects [Fas02], PointBase [Poi02] ou Oracle9i Lite [Ora02a] – permet de considérer des unités mobiles capables de gérer des données localement. TransMobi, dans son rôle d'intégrateur, tient compte de l'hétérogénéité des SGBD intégrant le système mobile – aucun type de SGBD particulier n'est ciblé. Afin d'être compatible avec le plus grand nombre de SGBD, les interfaces requises par TransMobi sont uniquement celles de *begin, commit, abort*. TransMobi prend en compte, également, les limitations des SGBD légers, par exemple, le fait qu'ils ne soient pas prévus pour gérer des transactions réparties.

TransMobi préserve l'autonomie des différents SGBD sous-jacents. Leur mode de fonctionnement n'est pas modifié et ils ne sont pas contraints de fournir des informations. Nous considèrons que les sources de données sont faiblement couplées.

TransMobi rend transparent les variations de l'environnement mobile aux SGBD sous-jacents. Les SGBD ne s'occupent pas de l'environnement mobile, ils n'ont pas connaissance du caractère mobile des clients ou des autres sources.

L'objectif général de TransMobi est de gérer l'interaction de différents SGBD faiblement couplés, autonomes et hétérogènes par l'intermédiaire des transactions T_{AMT}. Du fait des caractéristiques de l'environnement mobile, nous cherchons à fournir aux SGBD le plus d'indépendance d'exécution possible.

Les applications peuvent être de différents types et différents niveaux de complexité. Elles peuvent demander des transactions d'un des cinq modèles d'exécution présentés dans la section 2.1.2. En effet, le concepteur de l'application peut définir des transactions qui seront exécutées sur des SGBD localisés sur le réseau fixe, sur une unité mobile ou sur des unités mobiles et fixes.

D'une manière très générale, le fonctionnement global se déroule comme suit. L'application demande à TransMobi l'exécution d'une transaction T_{AMT}. Selon l'état courant de l'environnement mobile, TransMobi choisit parmi les alternatives d'exécution de la T_{AMT}, la plus adéquate. Ensuite, il distribue les transactions composantes aux SGBD correspondants. TransMobi doit assurer un ordre correct d'exécution globale ainsi que la validation des T_{AMT}.

6.2 Architecture à trois tiers de TransMobi

Dans TransMobi le tiers client est nommé *TMClient*, l'agent *TMAgent* et le serveur *TMServeur* (cf. figure 6.2). Seul le TMAgent est contraint d'être localisé sur une unité fixe. Les autres tiers peuvent être situés aussi bien sur des unités mobiles que fixes. Les tiers installés sur des unités mobiles (appelés TMClients-mobiles ou TMServeurs-mobiles) doivent être attachés à un TMAgent. Toute communication entre un tiers mobile et un tiers fixe doit passer par un TMAgent.

Les sections qui suivent (6.2.1, 6.2.3 et 6.2.2) donnent en détail les fonctions de chacun des tiers de l'architecture.

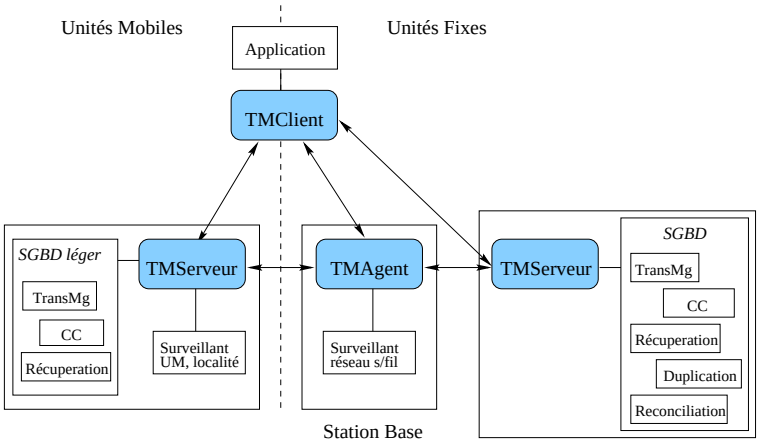

FIG. 6.2 – *Architecture client-agent-serveur de TransMobi*

6.2.1 TMClient

Un *TMClient* interagit directement avec l'application mobile. Il est appelé *TMClient-fixe* lorsqu'il s'exécute sur des unités fixes et *TMClient-mobile* lorsqu'il s'exécute sur des unités mobiles. Les principales caractéristiques du TMClient sont les suivantes :

- Il existe un TMClient par site. Chaque TMClient peut interagir avec plusieurs applications mobiles.
- TMClient interagit avec les TMServeurs des différents SGBD sous-jacents.
- Un TMClient-mobile doit être attaché à un TMAgent pour pouvoir communiquer des TMServeurs-fixes.
- Sur un même site, il peut y avoir une application et un SGBD. Dans ce cas, le TMClient et le TMServeur peuvent interagir directement qu'ils soient fixes ou mobiles.

Le TMClient reçoit, de l'application, la demande d'exécution d'une T_{AMT}. Il analyse les descripteurs d'environnement des alternatives de la T_{AMT} et selon l'état courant de l'environnement, il décide quelle alternative peut démarrer. Il est possible qu'au moment du déclenchement de la T_{AMT}, aucune alternative d'exécution ne puisse démarrer. Dans ce cas, l'exécution de la T_{AMT} peut être reportée jusqu'à ce que l'état de l'environnement soit adéquat ou elle est annulée.

6.2.2 TMServeur

Le *TMServeur* interagit directement avec le SGBD sous-jacent. Il est appelé *TMServeur-fixe* lorsqu'il s'exécute sur des unités fixes et *TMServeur-mobile* lorsqu'il s'exécute sur des unités mobiles. Un TMServeur :

- interagit avec les SGBD serveurs en demandant l'exécution de transactions composantes pour le compte des applications mobiles. Il utilise des interfaces classiques pour communiquer avec eux (*begin, commit, abort*).
- participe au processus de validation globale (cf. section 6.4.1) ;
- peut coordonner le processus de validation globale lorsqu'il est un TMServeur-fixe ;
- peut gérer le *graphe de sérialisabilité globale* (cf. section 6.4.2) lorsqu'il est un TMServeur-fixe ;
- lorsqu'il est mobile, il interagit avec un surveillant de l'état des ressources locales (cf. section 6.3). Un TMServeur-mobile fournit l'information sur les ressources de l'unité mobile où il s'exécute et éventuellement sur les services offerts.

Dans le processus global, le TMServeur reçoit les transactions composantes et coordonne leur exécution sur le SGBD sous-jacent. Il peut avoir d'autres responsabilités (coordination de la validation globale ou gestion du graphe de sérialisabilité globale) s'il est localisé sur un site fixe.

6.2.3 TMAgent

Les *TMAgents* jouent un rôle particulièrement important. Ils sont installés sur des stations base ou sur des unités fixes capables de communiquer avec des unités mobiles. La fonction principale d'un TMAgent est de soutenir les tiers mobiles.

L'utilisation des agents est très favorable aux unités mobiles qui peuvent avoir des déconnexions et qui généralement ont des ressources limitées. Un TMAgent réduit les besoins de communication sans fil car il assure la présence de l'unité mobile. Ceci limite la communication directe avec l'unité mobile. De même, il permet d'économiser des ressources sur les unités mobiles.

Lorsqu'un TMClient-mobile a besoin de répartir des transactions composantes, il les transmet à son correspondant TMAgent qui se charge de la répartition. Lorsqu'il s'agit d'un TMClient-fixe, ce dernier fait lui-même la répartition des transactions composantes. Pour communiquer avec un TMServeur-mobile, le TMClient-fixe doit passer par le TMAgent correspondant.

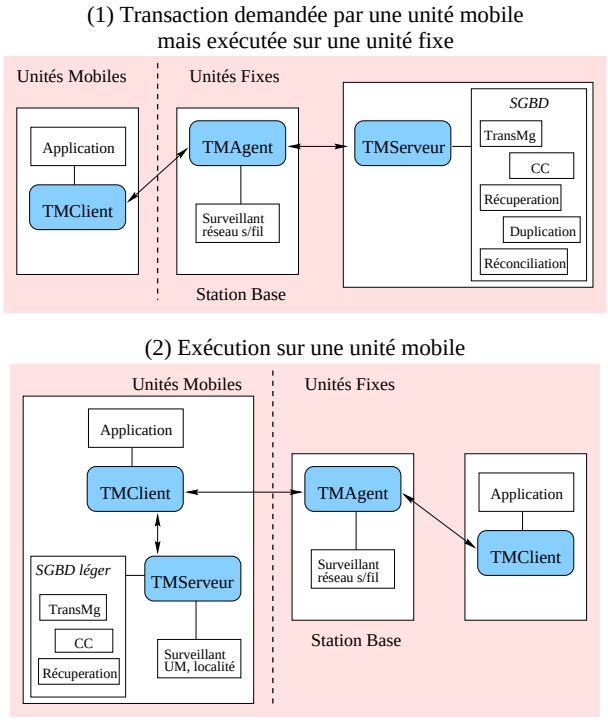

FIG. 6.3 – *Configurations 1 et 2 de l'architecture de TransMobi*

D'une façon générale, le rôle du TMAgent vis-à-vis des tiers mobiles est le suivant. Un TMAgent :

- répartit l'interaction entre les TMClients et les TMServeurs en deux parties : (1) le TMClient avec le TMAgent et (2) le TMAgent avec le TMServeur. Ceci permet d'utiliser des protocoles différents pour chaque partie et l'interaction peut être faite de façon indépendante ;
- est un représentant des tiers mobiles qui stocke les messages destinés aux unités mobiles déconnectées ;
- peut aider les TMClients pendant la durée des déconnexions. Par exemple, un TMClient-mobile peut soumettre ses demandes au TMAgent, se déconnecter, et récupérer les résultats une fois la connexion rétablie ;
- peut être utilisé pour économiser, par exemple, la vie de la batterie. Un TMClient-mobile peut soumettre ses demandes à son TMAgent, entrer en mode veille et attendre que le TMAgent l'avertisse lorsque la réponse est prête ;

- suit le déplacement des unités mobiles (en coopération avec d'autres TMAgents). Par exemple, lors d'un hand-off[1], le dernier TMAgent envoie l'information concernant l'unité mobile au nouveau TMAgent ;
- peut aider à libérer de la place sur le support persistant des TMServeurs-mobiles en sauvegardant des données temporairement ;
- stocke l'information de coordination transactionnelle des TMServeurs-mobiles (l'état des transactions en cours, le processus de validation, etc.) ;
- peut coordonner le processus de validation globale à défaut d'un TMServeur-fixe participant à l'exécution d'une T_{AMT} (cf. section 6.4.1) ;
- interagit avec un surveillant de l'environnement (service d'événements) qui perçoit l'état du réseau sans fil. Le TMAgent envoi l'information sur le réseau sans fil aux TMClients mobiles ou fixes.

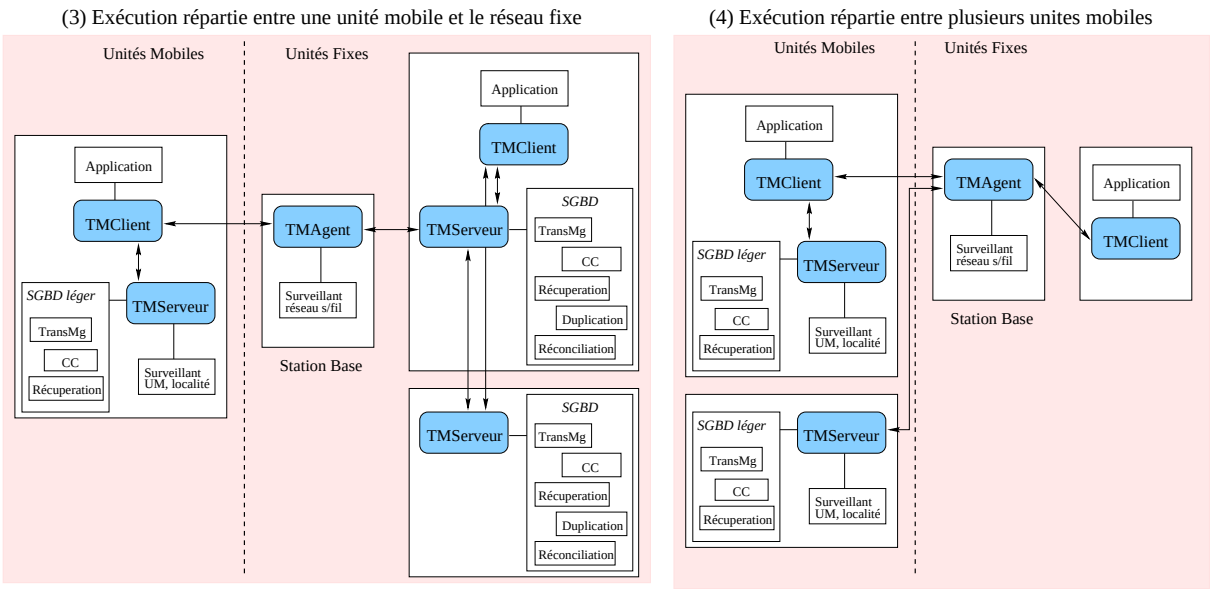

FIG. 6.4 – *Configurations 3 et 4 de l'architecture de TransMobi*

6.2.4 Configurations possibles de l'architecture

Les trois tiers de l'architecture TransMobi peuvent être composés de différentes façons. La composition dépend principalement du caractère mobile des TMClients et des TMServeurs.

1. Changement de cellule sans perte de connexion.

Afin de montrer quelques configurations possibles, nous prenons comme référence les cinq modèles d'exécution des transactions mobiles (cf. section 2.1.2).

La figure 6.3 (1) montre une application localisée sur une unité mobile qui peut demander l'exécution de transactions sur des SGBD localisés sur le réseau fixe. Pour cette configuration l'unité mobile peut être très limitée de ressources. Dans la figure, nous montrons seulement un SGBD sur le réseau fixe mais plusieurs peuvent exister. Dans la figure 6.3 (2), l'application peut être sur le réseau fixe ou sur l'unité mobile et le SGBD sur une unité mobile. Pour cette configuration, l'unité mobile doit être capable de gérer des données localement. Il est intéressant de noter que si l'application et le SGBD se localisent sur le même site, le TMClient et le TMServeur peuvent interagir directement.

La figure 6.4 (3) montre une configuration avec plusieurs SGBD, dont un sur une unité mobile. L'application peut aussi bien être sur l'unité mobile que sur le réseau fixe (voir figure 6.4 (3) et (4), et figure 6.5 (5)). Il faut remarquer que la communication entre les tiers fixes se fait de manière directe. Dans la configuration de la figure 6.4 (4), la communication entre les sites mobiles se fait par l'intermédiaire d'un TMAgent. Dans ce cas particulier, nous considérons que les deux unités mobiles partagent le même TMAgent. Cependant, elles peuvent être attachées à différents TMAgents.

La configuration de la figure 6.5 (5) est la plus complète, car les transactions mobiles peuvent être largement réparties entre sites mobiles et fixes.

(5) Exéction répartie entre plusieurs unités mobiles et fixes

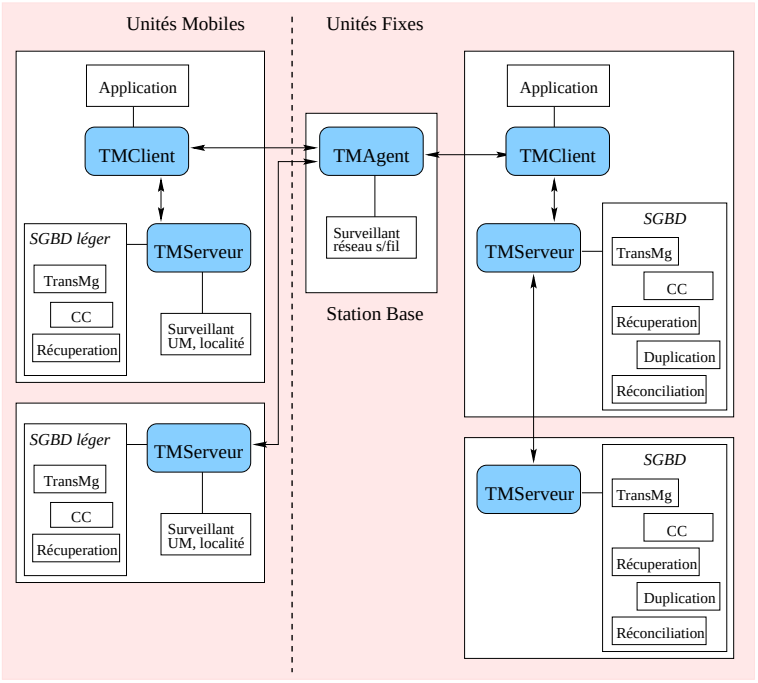

FIG. 6.5 – *Configuration 5 de l'architecture TransMobi*

6.3 Perception de l'environnement mobile

TransMobi met en œuvre les transactions mobiles de type AMT. L'objectif général est d'adapter leur exécution selon l'état courant de l'environnement. Afin de rendre possible l'adaptabilité, il est nécessaire de "percevoir" l'état de l'environnement mobile à un moment donné.

Dans cette section, nous présentons l'approche par événements qui permet de percevoir l'état de l'environnement (section 6.3.1). Nous présentons également comment les événements peuvent être produits (section 6.3.2).

6.3.1 Evénements de l'environnement mobile

L'environnement mobile inclut principalement les caractéristiques du réseau sans fil (RSF) et des unités mobiles. D'autres caractéristiques dépendant de la sémantique et des intérêts de l'application peuvent faire partie de l'environnement (cf. tableau 4.1).

Afin de percevoir l'environnement, TransMobi utilise un *service d'événements*. Le tableau 6.1 introduit les types d'événements identifiés. La définition des événements de ce tableau est très flexible, d'autres événements peuvent être introduits. Les événements sont produits à partir de l'information fournie par différents surveillants de l'environnement (cf. section 6.3.2).

	Type d'événement	**Information attachée**	**Mode Not.**
RSF	*e-connexion*	Id de l'UM	*Push/Pull* synchrone/ asynchrone
	e-déconnexion	Id de l'UM	
	e-hand-off	Id de la nouvelle SB	
	e-bande-passante-change	Largeur de la bande passante	
	e-prix-communication-change	Prix de communication	
UM	*e-batterie-disponible*	Batterie disponible	
	e-cache-disponible	Cache disponible	
	e-mémoire-persistant-disponible	Mémoire persistante disponible	
	e-capacité-calcul-disponible	Capacité calcul disponible	
	e-localité-changes	Localité	

TAB. 6.1 – *Type d'événements pour la perception de l'environnement mobile.*

Le besoin de détection et de notification des événements dépend de l'environnement considéré ainsi que des contraintes sémantiques des applications et des clients. Par exemple, si la bande passante du réseau ne souffre pas de grands changements, l'événement *e-bande-passante-change* n'est pas nécessaire. De même, si le prix de communication ne varie pas, *e-prix-communication-change* n'a pas lieu d'être. Egalement, *e-prix-communication-change* peut ne pas être intéressant à surveiller si les utilisateurs sont prêts à payer n'importe quel prix afin d'avoir accès aux applications. Ainsi, l'implantation du service d'événements peut varier selon le contexte d'exécution de chaque application mobile.

Certains événements peuvent varier fortement sans pour autant affecter l'exécution des transactions. Dans ce cas, un filtrage peut être implanté pour notifier les événements seulement lorsque les changements sont significatifs. Par exemple, lorsque les dimensions de l'en-

vironnement mobile (comme bande-passante) changent de valeur (de *forte* à *moyenne*). Dans la suite, nous introduisons les différentes façons de faire la notification pour TransMobi.

Type de notification. En général, les événements sont notifiés avec des mécanismes de *pull* ou *push*. Dans le premier cas, le consommateur (TransMobi) récupère les événements auprès du mécanisme de notification. Par contre, dans la technique *push*, le mécanisme de notification signale les événements au consommateur.

Moment de la notification. Les deux types de notification – *push/pull* – peuvent être exécutés de manière immédiate ou différée. Dans le mode immédiat, la notification est faite lorsque l'événement est généré. Par contre, en mode différé les notifications sont faites à des instants précis (périodiquement) ou lorsque le consommateur a besoin des événements.

La notification d'événements dans TransMobi

TransMobi a besoin de connaître l'état de l'environnement mobile : (1) lorsqu'une demande d'exécution de transaction arrive et (2) lorsqu'une T_{AMT} attend un état particulier pour déclencher une de ses alternatives d'exécution.

1. Connaître l'état de l'environnement au moment du lancement d'une T_{AMT} demande des notifications de type *pull* différé. Si l'état correspond au descripteur d'environnement (DE_k) d'une des alternatives, celle-ci est déclenchée.

2. Connaître l'état de l'environnement dès qu'un certain état se produit, demande des notifications de type *push* immédiat. Si l'état ne convient à aucun descripteur d'environnement des alternatives, le déclenchement peut être retardé jusqu'à ce que l'environnement soit adéquat.

L'approche par événements facilite le déclenchement immédiat et retardé des T_{AMT}. En effet, la notification des événements de l'environnement peut être réglée (adaptée) au caractère particulier de chaque ressource surveillée, mais aussi aux besoins des transactions. Plus concrètement, la notification des différents événements peut être faite de la manière la plus appropriée aux caractéristiques de l'environnement mais aussi aux besoins des applications.

6.3.2 Surveillance de l'environnement mobile

La surveillance de l'environnement mobile peut être faite par le système d'exploitation, la couche de communication, etc. Fréquemment les systèmes d'exploitation prévoient des fonctions pour la surveillance des ressources. [2] Ces fonctions peuvent être directement utilisées ou modifiées afin d'obtenir la surveillance adéquate. Certaines caractéristiques de l'environnement sont faciles à surveiller, d'autres ont besoin d'un matériel particulier. La nécessitée de surveillance dépend fortement du contexte de l'application, des clients, des serveurs et de la communication sans fil. Dans la suite, nous présentons à titre d'exemple quelques manières de surveiller l'environnement mobile afin de produire les événements correspondants.

2. Un bon exemple est le système de fichiers *proc* qui offre des informations sur l'état courant du noyau Linux.

Surveillance de la communication sans fil

e-connexion. La connexion des unités mobiles peut être obtenue par des surveillants comme ceux proposés par Columbia Mobile IP [IDM91] – *mhmicp* ou *pumicp*.

e-déconnexion. La déconnexion d'une unité mobile peut être obtenue lorsqu'un message d'erreur de communication particulier est reçu.

e-hand-off. L'information concernant le changement de cellule (processus de *hand-off*) peut être obtenue à travers la carte du réseau sans fil.

e-bande-passante-change. La bande passante peut être estimée selon l'interface de la carte réseau sans fil active. Autrement, l'estimation peut être calculée d'après la latence de communication et le *throughput* de la connexion.

e-prix-communication-change. Cette information peut être obtenue par l'intermédiaire de *profils d'environnement.* Par exemple, l'information, concernant les contrats avec les fournisseurs de communication, peut être stockée sous forme de profil. Ainsi, selon le type de communication utilisé, le prix de communication peut être obtenu.

Surveillance des ressources des unités mobiles

e-batterie-disponible. Afin d'obtenir de l'information concernant la consommation de la batterie, PowerScope [FS99] – un outil pour surveiller la consommation d'énergie – utilise de l'échantillonnage statistique pour profiler l'utilisation d'énergie d'un système informatique (logiciel, matériel). Le fichier */proc/apm* de Linux fournit des informations sur l'état de la gestion de la consommation d'énergie.

e-cache-disponible. Des fonctions concernant l'utilisation du cache sont généralement fournies par les systèmes d'exploitation, par exemple, le fichier */proc/meminfo* de Linux. Ce fichier est utilisé, par exemple, par la commande *top*.

e-mémoire-persistant-disponible. L'information pour produire cet événement peut être fournie par des fonctions comme *df* contenues dans le système d'exploitation Linux.

e-capacité-calcul-disponible. Les systèmes d'exploitation contiennent souvent les fonctions *prof*, *gprof* qui profilent l'utilisation du CPU ou le fichier */proc/cpuinfo* de Linux. De telles fonctions peuvent fournir l'information nécessaire pour la génération de l'événement *e-capacité-calcul-disponible*.

e-localité-changes. Afin d'obtenir la localité d'une UM, un senseur GPS (*Global Position System*) peut être utilisé. Les positions géographiques visitées fréquemment par l'utilisateur peuvent être traduites – de longitude et latitude – à maison, travail, mairie, centre ville, etc.

Certaines informations liées aux caractéristiques surveillées peuvent être attachées aux événements. Par exemple, dans *e-prix-communication-change*, le nouveau prix de communication peut être attaché.

6.4 Techniques pour garantir les propriétés des T_{AMT}

Le tableau 6.2 rappel les propriétés des transactions AMT (cf. tableau 4.3). Nous considérons que les propriétés AID des transactions composantes sont assurées par les SGBD sous-jacents. La cohérence des transactions composantes est conditionnée à la sémantique des alternatives d'exécution. En effet, la cohérence d'une exécution partielle dépende de la sémantique de l'alternative.TransMobi doit garantir les propriétés correspondant aux niveaux des alternatives d'exécution (AE_k) et des T_{AMT} (niveau racine). Plus particulièrement, l'atomicité sémantique, la cohérence sémantique, la sérialisabilité globale et la semi-atomicité.

Propriété	t_{ki}	AE_k	T_{AMT}
Atomicité	$\surd$	Atomicité sémantique	Semi-atomicité
Cohérence	$\surd$ conditionnée	Cohérence sémantique	
Isolation	$\surd$	Relâchée (validation locale)	
Durabilité	$\surd$ conditionnée	SGBD sous-jacent	
Correction	Sérialisabilité	Sérialisabilité globale	

TAB. 6.2 – *Rappel des propriétés du modèle AMT.*

Afin de gérer les particularités de l'environnement mobile (fréquentes variations, communication sans fil limitée et chère, etc.), nous considérons que les techniques qui assurent les propriétés des transactions mobiles doivent prendre en compte les points suivants :

- réduire le nombre de messages transmis par le réseau sans fil;
- éviter d'échanger de grands volumes d'information ;
- permettre la déconnexion des participants ;
- prendre en compte les ressources limitées des unités mobiles ;
- prendre en compte le déplacement des unités mobiles ;
- utiliser des approches optimistes plutôt que pessimistes afin d'éviter le blocage du système.

Dans les chapitres 2 et 3, nous avons montré que les techniques traditionnelles, utilisées pour assurer l'atomicité et un ordre d'exécution correct, ne sont pas toujours adaptées pour le contexte mobile. Nous avons également montré que les techniques plus souples peuvent être applicables si quelques adaptations sont faites. Dans le chapitre 3, nous avons montré l'existence de nouvelles techniques proposées pour satisfaire aux besoins de l'environnement mobile. Malheureusement, ces techniques ne prennent pas toujours en compte tous les points que nous venons de citer.

La semi-atomicité est garantie par chaque TMClient, du fait que celui-ci doit assurer l'exécution d'une seule alternative. La cohérence sémantique est obtenue du fait de l'utilisation de transactions de compensation. Dans la section 6.4.1, nous proposons un protocole pour la gestion de l'atomicité sémantique des alternatives d'exécution, nommé CO2PC. Ensuite, dans la section 6.4.2, nous proposons une adaptation de la technique OTM afin d'assurer la sérialisabilité globale des alternatives d'exécution. Les deux techniques prennent en compte les points cités ci-dessus.

6.4.1 Le protocole de validation CO2PC

Du fait de l'environnement mobile, le protocole utilisé pour assurer l'atomicité des alternatives d'exécution doit permettre aux SGBD une liberté d'exécution et limiter la communication sans fil et le blocage de ressources. Il doit permettre aux transactions composantes de choisir entre une validation locale anticipée et une validation coordonnée avec la validation des alternatives d'exécution.

Dans la suite de cette section, nous proposons CO2PC, un protocole pour assurer l'atomicité sémantique des alternatives d'exécution. Nous commençons par montrer le principe et le fonctionnement du protocole (section 6.4.1.1) – indépendamment de TransMobi – puis la reprise après pannes (section 6.4.1.2) et ses principales avantages (section 6.4.1.3). Ensuite, nous montrons l'intégration de CO2PC dans TransMobi (section 6.4.1.4). Nous terminons par un positionnement de CO2PC par rapport à d'autres protocoles de validation (section 6.4.1.5).

6.4.1.1 Fonctionnement de CO2PC

CO2PC est une Combinaison d'une approche Optimiste avec le protocole de validation à deux phases 2PC [Gra78]. CO2PC permet une validation optimiste (unilatérale/anticipée) ou non-optimiste des transactions composantes.

L'approche optimiste permet la validation locale des transactions composantes avant la validation de l'alternative d'exécution. En cas d'annulation de l'alternative, la validation anticipée rend nécessaire l'exécution de transactions de compensation. En effet, pour que les transactions composantes puissent valider unilatéralement, elles doivent être compensables.

L'approche non-optimiste est utilisée pour les transactions composantes non-compensables ou pour celles qui ne permettent pas la validation anticipée. Le choix de l'utilisation du protocole 2PC est motivé par son interface *préparation*. En effet, l'état *préparation* rend possible la rétention des ressources des transactions composantes non-compensables jusqu'à la validation ou l'annulation de l'alternative correspondante. Dans 2PC les composantes qui terminent leurs opérations entrent dans l'état de *préparation*. Dans cet état, une transaction ne peut être ni annulée ni validée de manière autonome par le SGBD sous-jacent. Seule une opération de validation/annulation émise par un gestionnaire global peut faire sortir de cet état les transactions composantes.

Dans le protocole CO2PC, il existe un coordinateur qu'assure la validation globale d'un ensemble de participants. Chaque participant a un *proxy* associé.

- Un **participant** est représenté par le *proxy* qui intéragit avec le SGBD sous-jacent exécutant une transaction composante. Le *proxy* est un intermédiaire entre l'application et le SGBD sous-jacent. Il cache la validation globale au SGBD sous-jacent. Il rend possible la participation, dans une validation globale, du SGBD même s'il n'est pas conçu pour le faire. Il permet également de réduire le nombre de messages nécessaires pour l'exécution de CO2PC.

  Le *proxy* est installé dans le même site que son SGBD sous-jacent. Ils peuvent être localisés sur des unités mobiles ou fixes et ils peuvent valider les transactions composantes de manière optimiste ou non-optimiste.

- Le rôle de **coordinateur** est joué par un des participants fixes. Un participant mobile ne doit pas être coordinateur car il est sujet à des déconnexions fréquentes. De plus ses ressources peuvent être limitées ce qui ne permet pas d'effectuer des opérations de journalisation ou de transmettre des messages de communication. Le coordinateur est désigné au démarrage de l'alternative d'exécution. Chaque validation globale peut avoir un coordinateur différent.

D'une façon générale, les transactions composantes (t_{ki}) d'une alternative d'exécution (AE_k) sont réparties. Chacune des t_{ki} est exécutée sur un SGBD sous-jacent. Afin de valider globalement, tous les participants (*proxies*) doivent envoyer un message de *vote* (*validation* ou *annulation*) au coordinateur qui prend une décision globale, et la propage aux participants. Si tous les *votes* sont *validation*, la décision prise est la validation globale, dans le cas contraire, l'alternative est annulée globalement.

La figure 6.6 présente le diagramme de séquences de CO2PC. Le cas de configuration de la figure comporte deux participants, un mobile et un fixe. Le participant fixe exécute une validation non-optimiste et le mobile une validation optimiste. Le rôle du coordinateur est joué par le participant fixe.

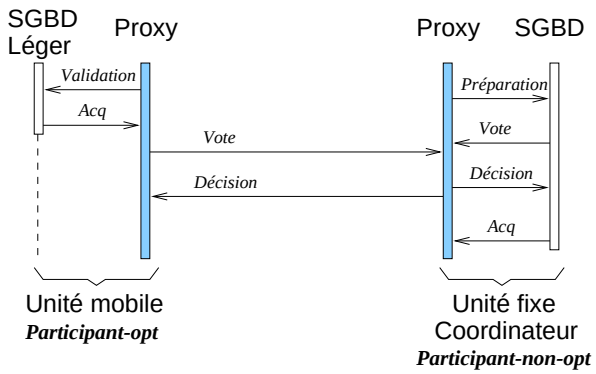

FIG. 6.6 – *Le protocole de validation CO2PC.*

Validation optimiste. Un participant faisant une validation optimiste (appelé *participant-opt*), demande l'exécution de la t_{ki} au SGBD sous-jacent correspondant. Une fois l'exécution terminée, t_{ki} est validée ou annulée de manière unilatérale. Ensuite, le participant-opt envoie son *vote* au coordinateur. Celui-ci prend la décision globale et la propage. Si la décision est de valider, les participants-opt ont fini. Si la décision est d'annuler, les participants-opt doivent démarrer une transaction de compensation (tc_{ki}).

Validation non-optimiste. Un participant faisant une validation non-optimiste (appelé *participant-non-opt*) demande l'exécution de la t_{ki} au SGBD sous-jacent correspondant. Une

fois l'exécution terminée, le participant-non-opt exécute le protocole 2PC localement.[3] Afin d'exécuter localement le protocole 2PC, le *proxy* joue le rôle de coordinateur du SGBD sous-jacent qui joue le rôle de participant. Il commence le protocole 2PC avec son SGBD sous-jacent (message de *préparation*). Le SGBD lui envoie le *vote* qui est redirigé au coordinateur (global). Si le vote est une *annulation*, le participant annule t_{ki} de façon unilatérale, sinon, t_{ki} entre dans l'état de *préparation*. Le coordinateur prend une décision globale et la propage aux participants. Dès que la décision du coordinateur arrive au participant, la *validation/annulation* correspondante est demandée au SGBD. Il faut remarquer que la partie 2PC du protocole CO2PC se fait en locale et n'implique pas d'échange de messages entre sites différents.

Situations de blocage. Les participants-opt ne peuvent pas être bloqués grâce à l'approche optimiste, mais ils peuvent entraîner le blocage des participants-non-opt. Un participant-non-opt peut être bloqué s'il vote pour une validation et :

- l'un des participants (optimiste ou non-optimiste) n'envoi pas son vote (à cause d'une panne ou d'une déconnexion indéfinie) ;
- le coordinateur tombe en panne ;
- se déconnecte indéfiniment.

Afin de limiter la durée des blocages, des délais (*timeouts*) sont utilisés. Chaque AE_k, ainsi que chacune de ses t_{ki}, a un délai assigné. Le délai de l'AE_k doit être plus grand que celui des t_{ki}. Chaque participant doit envoyer le vote avant l'expiration du délai de la t_{ki} qu'il exécute. Les participants peuvent renégocier les délais, par exemple lorsqu'ils vont se déconnecter et qu'ils connaissent la date de leur prochaine reconnexion. Si le coordinateur ne reçoit pas tous le votes avant l'expiration du délai de l'AE_k, il décide de son annulation. Après avoir voté, les participants attendent la décision globale jusqu'à l'expiration du délai de l'AE_k. Si un participant se déconnecte, lors du processus de déconnexion (cf. section 6.4.3), il délègue au TMAgent l'attente de la décision globale. Lors de la prochaine connexion, les participants reçoivent du TMAgent la décision du coordinateur et prennent les mesures nécessaires – annulation/validation ou compensation des t_{ki}.

Annulation d'une AE_k. Une AE_k valide uniquement lorsque le coordinateur a reçu tous les votes et que ceux-ci sont tous des validations. Par contre, l'annulation d'une AE_k peut avoir plusieurs raisons :

- une t_{ki} vote pour une annulation globale ;
- un cycle dans l'ordre d'exécution se produit et un problème de sérialisabilité survient (cf. section 6.4.2) ;
- le délai de l'AE_k expire sans que le coordinateur ait reçu tous les votes.

3. Une transaction non-compensable ne peut pas être exécutée sur un SGBD sous-jacent qui ne fournit pas l'interface 2PC.

6.4.1.2 Reprise après panne

Cette section montre les mesures prises dans le protocole CO2PC en cas de panne ou annulation (en particulier la compensation des t_{ki}). Elle parle également des garanties de durabilité fournies par CO2PC après la validation globale.

Journalisation Pendant l'exécution de CO2PC, le déroulement de ses différentes étapes est enregistré afin de fournir la récupération après pannes. La journalisation se fait aussi bien dans le journal du coordinateur que dans le journal des participants. Puisque les défaillances et les déconnexions peuvent arriver à n'importe quel moment, l'information journalisée doit être écrite sur un support persistant avant d'envoyer les messages.

Site	**Opération à journaliser**
Participant-opt	*Demande de Validation*
	Acquitement
Participant-non-opt	*Début2PC*
	Préparation
Tous les participants	*Vote*
	Décision
Coordinateur	*DébutCO2PC*
	Chaque *Vote*
	Décision

TAB. 6.3 – *Journalisation pendant l'exécution du protocole CO2PC.*

Le tableau 6.3 illustre l'information à journaliser par les participants et le coordinateur. Afin de préserver l'autonomie des SGBD sous-jacents, nous ne faisons aucune supposition sur leur façon d'assurer les propriétés ACID. Nous ne demandons pas, par exemple, que les SGBD partagent leurs journaux concernant les opérations transactionnelles.

Compensation. Lorsqu'une alternative d'exécution est annulée, les participants gérant des transactions non-compensables demandent l'annulation de t_{ki} et ceux gérant des transactions compensables demandent l'exécution de transactions de compensation si la correspondante t_{ki} a été validée (cf. figure 6.7). Une fois que les transactions de compensation ont démarrée, elles doivent être validées. Cette caractéristique est appelé *persistance de compensation* [GMS87]. Elle est assurée par la résoumission des transactions de compensation jusqu'à leur validation. Si la persistance de compensation est garantie, il n'y a pas besoin d'utiliser un protocole de validation pour assurer l'atomicite de l'ensemble des transactions de compensation d'une alternative. Les transactions de compensation des transactions composantes qui ont validé séquentiellement sont exécutées dans l'ordre inverse de validation.

Durabilité. Afin de respecter l'autonomie et de limiter le besoin de communication, CO2PC délègue la responsabilité d'assurer la durabilité des transactions aux SGBD sous-jacents. A la différence d'autres protocoles, CO2PC ne force pas l'écriture des journaux des modifications sur des supports stables (unités fixes) avant la validation des transactions composantes. En effet, nous avons fait un compromis pour concevoir un protocole convenable

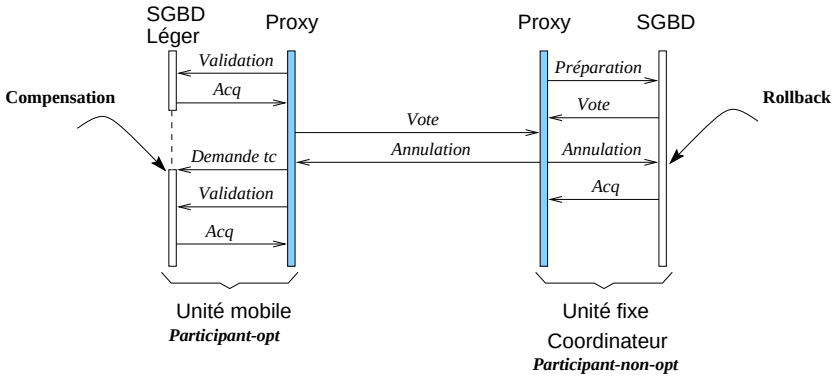

FIG. 6.7 – *L'annulation dans le protocole CO2PC.*

pour l'environnement mobile. En utilisant CO2PC, les ressources mobiles (communication sans fil, disponibilité de données locales) sont gérées en prenant en compte les limitations des environnements mobiles.

CO2PC offre aux SGBD sous-jacents la possibilité de décider de la manière d'assurer la durabilité des modifications faites par des transactions validées. En effet, un SGBD localisé sur une unité mobile peut décider de faire le transfert des modifications sur un support stable : après la validation de chacune des transactions, après la validation d'un nombre défini de transactions, lorsqu'il bénéficie d'une bonne connexion, etc.

6.4.1.3 Les avantages de CO2PC

CO2PC est un protocole flexible qui offre plusieurs avantages. Il :

- permet la déconnexion des participants ;
- permet la validation unilatérale des transactions composantes (validation optimiste) ;
- permet la validation coordonnée avec l'alternative d'exécution (validation non-optimiste) ;
- limite le nombre de messages nécessaires pour son exécution (2 par participant) : un message de *vote* et un autre de *décision* ;
- délègue aux SGBD la décision d'assurer la durabilité de la manière la plus convenable.

CO2PC est conçu pour relâcher l'atomicité, cependant, si toutes les transactions composantes d'une alternative d'exécution font une validation non-optimiste, l'atomicité n'est pas relâchée.

6.4.1.4 CO2PC et TransMobi

Dans l'implantation de CO2PC dans TransMobi, les TMServeurs assurent le rôle des *proxies* et la communication entre les unités mobiles et les fixes se fait par l'intermédiaire d'un TMAgent. Comme le rôle du coordinateur doit être sur une unité fixe, à défaut de TMServeur participant dans une alternative d'exécution, un TMAgent peut jouer ce rôle. La figure 6.8(a) illustre le cas dans lequel un TMAgent est un coordinateur.

La figure 6.8(b) montre qu'aussi bien les unités mobiles que fixes peuvent valider d'une manière optimiste ou non-optimiste. Les participants sont : une unité mobile (UM C) et deux unités fixes (UF D et E).

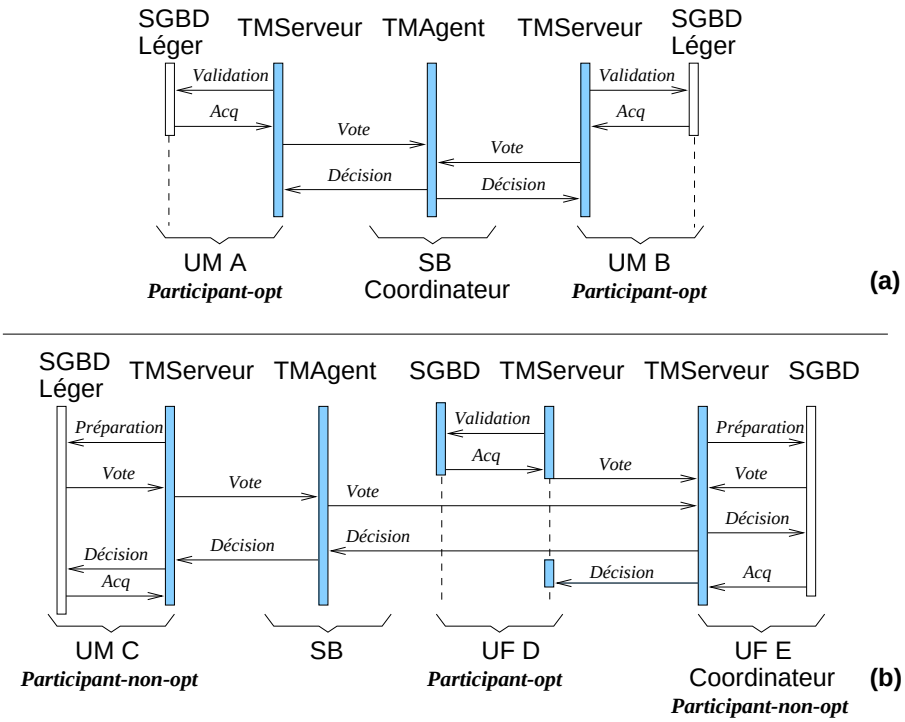

FIG. 6.8 – *Le protocole CO2PC et TransMobi.*

En ce qui concerne la journalisation, Les TMServeurs-mobiles doivent journaliser l'information liée au CO2PC dans leur journal mais aussi dans celui du TMAgent correspondant. Ceci est dû au fait que le stockage physique des unités mobiles n'est pas considéré comme stable car elles peuvent subir des pertes, dommages et déconnexions. Si l'unité mobile se déconnecte, la journalisation sur le TMAgent va attendre une prochaine reconnexion.

6.4.1.5 Positionnement de CO2PC par rapport à d'autres protocoles

Cette section positionne le protocole CO2PC par rapport aux protocoles 2PC (cf. section 2.2.2.1), UCM et TCOT (cf. section 3.2.1.2). Le premier est le protocole le plus répandu pour les SGBD traditionnels. Les deux derniers ont été proposés pour le contexte mobile.

Les protocoles de validation ont comme objectif principal d'assurer la propriété d'atomicité des transactions (le tout ou rien). Après la validation d'une transaction, ses modifications doivent être persistantes, ceci est la propriété de durabilité. Traditionnellement, les protocoles de validation coopèrent également à assurer la durabilité. Assurer les deux propriétés implique d'importantes contraintes pour les systèmes : un nombre élevé de messages, blocage des ressources, transfert des journaux de modifications, etc. En effet, si le système veut bénéficier des propriétés d'atomicité et de durabilité, il doit, en échange, pénaliser la disponibilité des données et son autonomie d'exécution.

Le compromis entre les contraintes imposées et les propriétés offertes conduit à des variations entre les différents protocoles (cf. figure 6.9). 2PC et UCM garantissent l'atomicité et la durabilité, néanmoins, les contraintes nécessaires pour les mettre en place peuvent être considérées comme strictes. UCM est plus souple que 2PC car il ne demande pas d'interface particulière aux participants, il réduit le nombre de messages et permet la déconnexion des sites pendant le processus de validation.

Afin d'assouplir les contraintes imposées au système, TCOT et CO2PC relâchent les propriétés offertes. Le fait de permettre une validation anticipée, les oblige à relâcher l'atomicité et à assurer l'atomicité sémantique.

De manière similaire, afin de limiter le besoin de communication sans fil, CO2PC ne force pas le transfert des journaux des modifications. CO2PC délègue la responsabilité d'assurer la durabilité aux SGBD sous-jacents. Afin de réduire les contraintes, CO2PC relâche les propriétés d'atomicité et de durabilité.

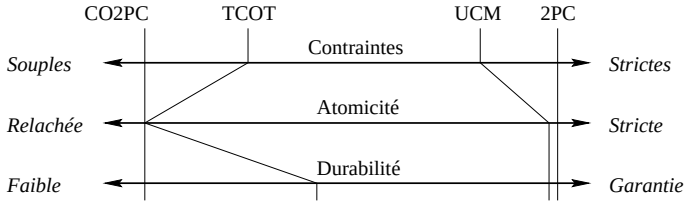

FIG. 6.9 – *Contraintes et propriétés de quelques protocoles de validation et CO2PC.*

La figure 6.10 montre les avantages et inconvénients des protocoles 2PC, UCM, TCOT et CO2PC. Nous pouvons constater, par exemple, que 2PC et UCM peuvent entraîner l'interblocage des participants et qu'ils ne permettent pas la validation anticipée des transactions. Pour le blocage des ressources, fréquament des techniques pessimistes de controle de concurrence sont utilisées, par exemple, le protocole 2PL rigoureux [BHG87, EGLT76]. L'utilisation de 2PL rigoureux dans l'environnement mobile peut entraîner des blocages indéfinis (du fait des déconnexions) et limite fortement la disponibilité des données (cf. section 2.2.1.4). UCM et TCOT forcent le transfert des journaux des modifications avant de permettre la valida-

tion des transactions. A notre sens, ceci représente une violation de l'autonomie des sites et génère des messages avant la validation des transactions.

Prot	Avantages	Inconvénients
2PC	- Son interface est largement implementée par les SGBD traditionnels - Génere des systèmes récuperables - Evite les annulations en cascade - Garantit l'atomicité et la durabilité	- Grand nombre de messages (4 par participant) - Blocage des ressources de tous les participants jusqu'à la validation globale (temps d'attente indéfini) (le protocole de concurrence 2PL est fréquement utilisé) - La déconnection des participants n'est pas considérée - Demande une interface particulière - La plupart des SGBD légers ne supportent pas son interface, en particulier l'état "préparation" - Ne permet pas la validation anticipée des transactions
UCM	- Il supporte la déconnection de participants - Nombre de messages 2 par participant (une seule phase) - Pas d'interface spécifique nécessaire - Evite les annulations en cascade - Garantit l'atomicité et la durabilité - Ne depande pas une interface particulière	- Chaque opération reçue doit etre acquittée - Blocage des ressources jusqu'à la validation de la transaction globale (temps d'attente indéfini) - Pour pouvoir valider, chaque participant doit transférer son journal des modifications (violation de l'autonmie des sites) - Doit etre utilisé avec un protocole de concurrence pessimiste comme le protocole 2PL - Ne permet pas la validation locale anticipée des transactions
TCOT	- Permet la validation anticipée des transactions - 2 messages nécessaires par participant en utilisant des délais - Rénégotiation des délais - Garantit la durabilité	- Utilisation de transactions de compensation - Relachement de l'atomicité en garantissant l'atomicité sémantique - Pour pouvoir valider, chaque participant doit transférer son journal des modifications (violation de l'autonmie des sites) - Ne supporte pas la déconnection de participants (uniquement le mode en "veille")
CO2PC	- Permet la validation anticipée des transactions mais aussi la validation synchronisée avec la validation global - Permet la déconnexion de participants - 2 messages nécessaires par participant en utilisant des délais - Rénégotiation des délais - Préserve l'autonomie des sites (ne demande pas le transfert des journaux des modifications) - Permet l'hétérogénéité des protocoles de concurrence utilisés	- Utilisation de transactions de compensation - Relachement de l'atomicité en garantissant l'atomicité sémantique - N'assure pas la durabilité

FIG. 6.10 – *Caractéristiques de quelques protocoles de validation et CO2PC.*

La figure 6.10 montre également la ressemblance entre TCOT et CO2PC. Ils permettent la validation anticipée des transactions, assurent l'atomicité sémantique en utilisant des transactions de compensation et réduisent le nombre de messages à l'aide de délais. CO2PC, à la différence de TCOT, préserve l'autonomie des sites en ne demandant pas le transfert des journaux des modifications. Ceci limite la taille et le nombre de messages nécessaires avant la validation. CO2PC supporte aussi la déconnexion des participants à n'importe quel moment du processus de validation.

6.4.2 Adaptation d'OTM

Dans cette section, nous introduisons la technique utilisée par TransMobi afin d'assurer la sérialisabilité globale des alternatives d'exécution (AE_k) concurrentes (cf. section 4.1.2.2). Les AE_k concurrentes appartiennent à différentes T_{AMT} car seule une AE_k par T_{AMT} doit être active. Dans la section 6.4.2.1, nous commençons par parler des conditions à respecter afin d'assurer un ordonnancement correct. Ensuite, dans la section 6.4.2.2, nous proposons une adaptation de la méthode OTM.

6.4.2.1 Conditions à respecter

Pour fournir la sérialisabilité globale, deux conditions sont nécessaires : (1) les SGBD sous-jacents doivent produire des ordonnancements sérialisables et recouvrables et (2) chaque SGBD maintient l'ordre relatif des sous-transactions perçu au niveau global.

Dans notre travail, grâce à que nous supposons que les SGBD sous-jacents préservent les propriétés ACID des transactions composantes, la première condition est assurée. Pour garantir la deuxième condition, la méthode utilisée doit assurer un ordre d'exécution globalement sérialisable. Nous considérons en plus, qu'afin d'être applicable aux systèmes mobiles, la méthode utilisée doit :

1. préserver l'autonomie des SGBD sous-jacents ;[4]
   - autonomie d'exécution : les SGBD peuvent valider ou annuler librement les transactions à sa charge,
   - autonomie de communication : les SGBD ne partagent pas d'information concernant leurs ordonnancements locaux,
2. permettre l'hétérogénéité des mécanismes de contrôle de concurrence utilisés dans les SGBD sous-jacents.

Il faut également respecter les conditions citées au début de la section 6.4. En particulier : utiliser des approches optimistes plutôt que pessimistes afin d'éviter le blocage du système.

Utilisation homogène des techniques traditionnelles. L'utilisation homogène par les SGBD sous-jacents de protocoles de contrôle de concurrence traditionnels (2PL [BHG87, EGLT76], estampillage [BHG87], certification [KR81], etc.) ne garantit pas la sérialisabilité globale. L'utilisation de 2PL rigoureux [BHG87] est une exception. Cependant, cette approche ne permet pas la libération anticipée des ressources (validation locale optimiste) ce qui la rend indésirable pour les environnements mobiles. De plus, cette approche est contradictoire avec l'hétérogénéité des sites.

4. A cause des caractéristiques de l'environnement mobile, nous considérons qu'il est important de préserver l'autonomie des sites et de permettre leur hétérogénéité (cf. section 2.1.1).

Exécution séquentielle. L'exécution séquentielle des transactions globales préserve la sérialisabilité globale seulement lorsqu'elle est combinée avec l'estampillage. Cette approche n'est pas convenable car elle réduit la concurrence et appliquée à l'environnement mobile elle peut entraîner des périodes d'attente indéfinis. Cette approche viole, également, l'hétérogénéité des sites.

Dans la section 2.2.1, nous avons présenté plusieurs techniques utilisées dans les systèmes multibases de données. Ces techniques sont plus appropriées pour l'environnement mobile. Parmi elles, nous proposons d'adapter la méthode OTM (*Optimistic Ticket Method*) (cf. section 2.2.1.1), qui est basée sur l'utilisation de tickets. Cette méthode respecte les conditions citées ci-dessus. Néanmoins, elle n'a pas été proposée pour l'environnement mobile. Elle ne respecte donc pas les conditions de la section 6.4. Dans la section qui suit, nous proposons une adaptation de la méthode OTM afin de l'appliquer à l'environnement mobile.

6.4.2.2 Fonctionnement d'OTM adapté

TransMobi combine la technique OTM avec CO2PC. Le fait d'utiliser OTM avec CO2PC représente la principale adaptation de la méthode OTM.

Traditionnellement, OTM est utilisée avec le protocole de validation 2PC. L'objectif est de valider les sous-transactions en même temps que la transactions globale. En effet, la validation unilatérale des sous-transactions n'est pas autorisée et les ressources restent bloquées jusqu'à la validation globale. Dans CO2PC (cf. section 6.4.1), l'un des objectifs est précisément de permettre la validation unilatérale des sous-transactions (ou transactions composantes) et la libération anticipée des ressources.

L'idée principale dans OTM est de convertir les conflits indirects en conflits directs (cf. section 2.2.1). Sur chaque SGBD, il est nécessaire de stocker une donnée spéciale, appelée *ticket*. Chaque sous-transaction globale lit et incrémente le ticket de façon atomique dans le site où elle s'exécute – il existe un ticket par site. Le ticket doit être géré comme n'importe quelle autre donnée. L'accès au ticket entraîne un ordre d'exécution local qui peut être perçu au niveau global.

Dans la suite, nous montrons l'utilisation d'OTM dans TransMobi. La figure 6.11 montre deux alternatives d'exécution concurrentes (AE_i et AE_j), chacune avec deux transactions composantes qui ont accès aux BD A et B. Sur chaque BD il existe un ticket (ticket A et B) dont l'accès entraîne un ordre entre les alternatives : $t_{i1} < t_{j1}$ et $t_{j2} < t_{i2}$.

Au niveau global, un graphe de sérialisabilité globale (GSG) est maintenu. Les nœuds du graphe correspondent aux AE_k en cours de validation ainsi qu'à celles récemment validées. Les arcs représentent l'ordre d'accès aux tickets. Si t_{ii} et t_{ji} s'exécutent sur un même site et t_{ji} obtient un ticket plus grand que celui de t_{ii}, alors un arc orienté d'AE_i vers AE_j est inséré. Lorsqu'un cycle se produit l'une des AE_k doit être annulée.

La décision de l'annulation d'une des AE_k générant un cycle, peut être basée sur différents critères. Par exemple, on peut annuler l'AE_k la plus récemment déclenchée, celle qui a le plus de transactions composantes non-validées (ou non-terminées), celle qui a la plus faible priorité (dans le cas ou des priorités peuvent être assignées aux AE_k), etc.

Un nœud peut être enlevé du GSG lorsque l'AE_k correspondante a validé et que toutes les AE_k qui ont un arc (qui arrive ou qui part) relié à ce nœud ont également validé.

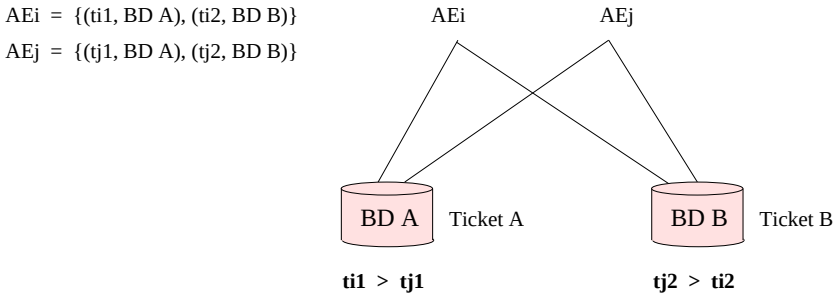

FIG. 6.11 – *Exemple de l'accès aux tickets dans OTM.*

Dans TransMobi, le GSG est maintenu par un gestionnaire global de sérialisabilité (GG) localisé sur un TMServeur-fixe. Pour l'instant, nous considérons que cette gestion est centralisée sur un seul TMServeur.

Le processus de sérialisabilité se déroule en même temps que celui de validation (cf. figure 6.12) :

- chaque transaction composante t_{ki} lit et incrémente la valeur du ticket ;
- lorsque t_{ki} termine son exécution, le TMServeur correspondant envoie le *vote* au coordinateur du CO2PC avec la nouvelle valeur du ticket attachée ;
- le coordinateur envoie la valeur du ticket – ainsi que l'information liée à t_{ki} – au gestionnaire du graphe global GG (tout coordinateur doit connaître le GG) ;
- le GG
  - crée un nœud AE_k dans le graphe, lorsqu'il s'agit de la première t_{ki} d'une alternative d'exécution ou
  - insère un arc entre AE_k et les autres AE_j du graphe si leurs transactions composantes ont eu accès au ticket sur la même BD, la direction de l'arc doit suivre l'ordre d'accès du ticket ;
- aussitôt qu'un cycle est détecté, le GG averti le coordinateur qui demande l'annulation de l'alternative correspondante ;
- si aucun cycle n'est détecté après l'insertion de tous les arcs, le GG donne le feu vert au coordinateur pour la validation de l'alternative.

La figure 6.12 montre le déroulement de la gestion du graphe de sérialisabilité globale dès la création des nœuds jusqu'à la détection d'un cycle. Un cycle est généré à cause de l'ordre d'accès contradictoire des transactions composantes des alternatives d'exécution dans les deux SGBD sous-jacents. La figure montre également les messages nécessaires :

- chaque participant transmet la valeur de la mise à jour du ticket. Cette valeur est ajoutée au message de *vote* envoyé au coordinateur de CO2PC qui transmet la dite valeur au GG ;

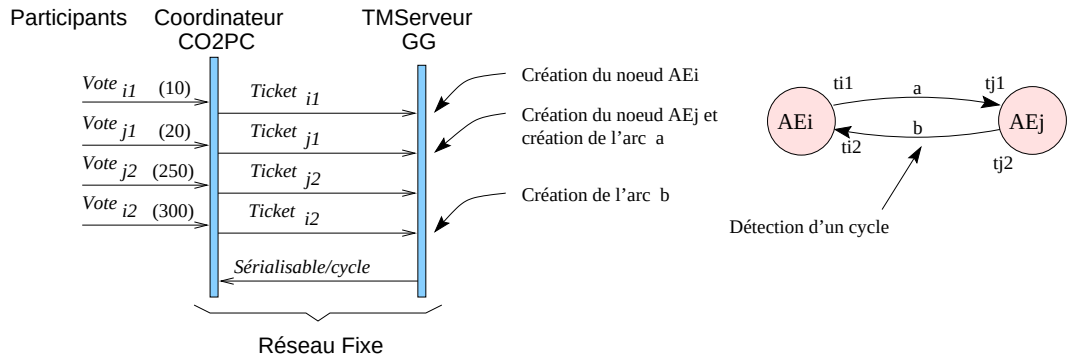

FIG. 6.12 – *Construction du graphe de sérialisabilité globale.*

- un message de notification sérialisable/cycle que le GG envoie au coordinateur de CO2PC.

La construction du GSG ne génère pas de nouveaux messages sur le réseau sans fil, grâce au fait que les participants attachent au message de vote de CO2PC, la nouvelle valeur du ticket. En effet, les nouveaux messages sont transmis sur le réseau fixe entre le coordinateur de CO2PC et le gestionnaire du graphe. Le nombre de messages est : un par participant (envoie de la valeur du ticket) plus un pour la notification sérialisable/cycle.

Les avantages. L'adaptation de la méthode OTM offre plusieurs avantages :

- c'est une méthode simple ;
- elle génère un ordre d'exécution sérialisable des alternatives d'exécution et par conséquence des transactions T_{AMT} ;
- elle préserve l'autonomie des SGBD sous-jacents ;
- elle autorise l'hétérogénéité des protocoles de contrôle de concurrence ;
- elle ne génère pas de nouveaux messages sur le réseau sans fil ;
- elle permet la déconnexion des sites mobiles (cf. section 6.4.3) ;
- grâce à l'utilisation de CO2PC les transactions composantes ont la possibilité de faire une validation unilatérale et les ressources peuvent être libérées de manière anticipée.

La désavantage d'OTM est que dans certains cas, elle génère des conflits là où possiblement il n'en avait pas. En effet, OTM peut entraîner l'annulation de certaines alternatives d'exécution à cause des conflits générés par l'accès au ticket. Une solution à ce problème peut être l'utilisation d'information sémantique pour permettre certains cycles dans l'exécution globale.

Par ailleurs, le fait de demander aux SGBD d'introduire et de gérer les tickets peut être perçu comme une violation de leur autonomie. Cependant, nous pensons que cette gestion est une charge minimale pour les SGBD.

6.4.3 Gestion des déconnexions

Une propriété clé d'un système mobile est la considération de la déconnexion comme un état normal du système. TransMobi gère la déconnexion au niveau transactionnel. C'est-à-dire, pendant l'exécution d'une T_{AMT}, les unités mobiles peuvent subir des déconnexions. Il existe deux types de déconnexions, la déconnexion désirable et l'indésirable. La première est prévue et souhaitée par l'utilisateur. La deuxième est une conséquence d'une variation d'état de l'environnement mobile. Par exemple, l'unité mobile est sortie de l'espace géographique couvert par le réseau sans fil ou la batterie de l'unité mobile est épuisée. Les protocoles de TransMobi tolèrent les deux types de déconnexions.

Dans CO2PC, les participants peuvent se déconnecter avant et après avoir voté :

Avant le vote. Le délai assigné aux transactions composantes permet aux unités mobiles de se déconnecter avant d'avoir voté, pourvue que le *vote* soit envoyé au coordinateur avant l'expiration du délai.

Après le vote. Une unité mobile peut se déconnecter temporairement après que le *vote* soit envoyé. Dans ce cas, la décision du coordinateur est stockée dans le TMAgent et elle est redirigée vers l'unité mobile lors de la prochaine reconnexion.

Les participants optimistes, aussi bien que les non-optimistes, peuvent se déconnecter (de manière prévue ou imprévue). La différence est que pour les non-optimistes, les ressources restent bloquées jusqu'à la prochaine reconnexion. Ceci est dû au fait que la partie 2PC de CO2PC finit lorsque la décision globale arrive au participant.

En ce qui concerne le protocole OTM, les déconnexions ne sont pas un problème car l'ordre de sérialisabilité est défini par l'accès aux tickets. En effet, l'incrémentation du ticket est une opération complètement indépendante de l'état de connexion des UM. L'unique contrainte est l'envoie de la valeur du ticket au coordinateur de CO2PC. Néanmoins, la durée de la déconnexion a une conséquence importante. Le GSG est construit dans l'ordre dans lequel les T_{AMT} valident. Ainsi, s'il existe un conflit, l'alternative qui essaie d'être sérialisée sera annulée. Cela veut dire que la probabilité d'annulation/validation est directement liée au temps qu'une transaction prend pour être sérialisée.

Un processus de déconnexion/reconnexion prévu peut être fait entre un tiers mobile désirant se déconnecter et le TMAgent auquel il est attaché.

Déconnexion d'un TMClient-mobile. Lorsqu'un TMClient-mobile désire se déconnecter, il transmet à son TMAgent le journal des T_{AMT} actives. Ainsi, le TMAgent est capable de stocker tous les messages concernant les T_{AMT} des TMClients-mobiles déconnectés.

Déconnexion d'un TMServeur-mobile. Lorsqu'un TMServeur-mobile désire se déconnecter, il transmet à son TMAgent le journal des transactions composantes actives. Il peut renégocier ses délais s'il connaît la date de la prochaine reconnexion. Ainsi, le TMAgent est ca-

pable de stocker tous les messages concernant les transactions composantes des TMServeurs-mobiles déconnectés.

Reconnexion d'un TMClient-mobile. Lors de la reconnexion d'un TMClient-Mobile, le TMAgent lui transmet tous ses messages en attente. Le TMClient-mobile peut ainsi prendre connaissance, par exemple, des T_{AMT} validées pendant son absence.

Reconnexion d'un TMServeur-mobile. Lors de la reconnexion d'un TMServeur-mobile, son TMAgent lui transmet tous ses messages en attente. Le TMServeur prend ainsi connaissance, par exemple, des décisions du coordinateur de la validation globale.

6.5 Décomposition fonctionnelle de TransMobi

La vue globale des interfaces du système est montrée dans la figure 6.13. Les applications sont conçues en utilisant les interfaces : ITMClient de TransMobi, Tx de X/Open et/ou SQL, OQL ou autres.

L'application définit des transactions T_{AMT} en utilisant l'interface ITMClient qui permet de demander une T_{AMT}. Les transactions composant les alternatives d'exécution sont délimitées par l'intermédiaire des interfaces classiques comme *begin, commit, abort.* La manipulation des données se fait par l'intermédiaire de procédures stockées et/ou des opérations SQL, OQL, etc.

L'interface ITMClient est implémentée par le composant TMClient de TransMobi (cf. section 6.5.1). Les autres interfaces sont fournies par les SGBD sous-jacents. Nous considérons que les SGBD possèdent un gestionnaire de transactions (GT) et un gestionnaire de ressources (GR) implantant les interfaces Tx et/ou SQL.

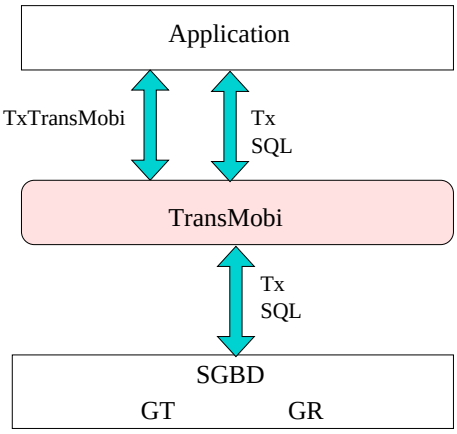

FIG. 6.13 – *Vue globale des interfaces utilisées par TransMobi.*

Dans la suite de cette section, nous montrons les différents composants fonctionnels de TransMobi. Dans un premier temps, nous introduisons le composant TMClient (section 6.5.1) dont l'interface est utilisée dans le code applicatif. Nous continuons avec le composant TMAgent (section 6.5.2) qui sert de représentant des tiers mobiles et dont l'interface est utilisée par les TMClients ou TMServeurs. Enfin, nous introduisons le composant TMServeur (section 6.5.3) dont l'interface est utilisée par les TMClient et TMAgents.

6.5.1 Composant TMClient

Le composant TMClient est installé sur des sites mobiles ou fixes. Les applications utilisent l'interface du `TMClient` pour demander les T_{AMT}. Quelques-unes des fonctions fournies par cette interface sont montrées dans la figure 6.14.

- `requestAMT(transamt amt)` est utilisée pour demander l'exécution d'une transaction mobile (T_{AMT}). Son paramètre est un objet de type `transamt` qui définit une transaction T_{AMT} (cf. figure 4.2). Cette fonction analyse la T_{AMT} demandée et choisit l'alternative d'exécution à déclencher.
- `abortAMT(transamt amt)` est utilisée pour annuler une T_{AMT} en cours d'exécution.
- `connect(identifiant idum)` permet de demander une connexion. Le paramètre de cette fonction est l'identifiant de l'unité mobile qui désire se connecter au réseau.
- `disconnect(identifiant idum)` permet de demander une déconnexion. Le paramètre de cette fonction est l'identifiant de l'unité mobile qui désire se déconnecter du réseau.

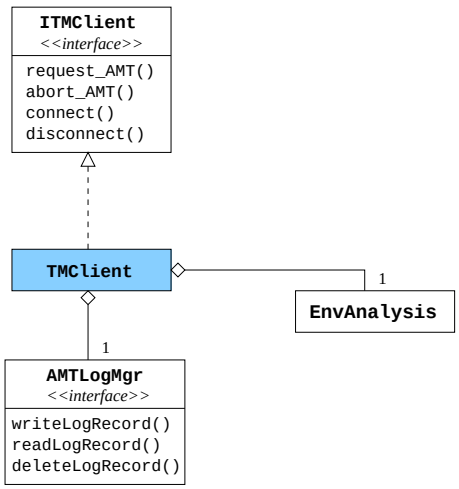

FIG. 6.14 – *Diagramme de classes du TMClient.*

Les modules de `TMClient` sont `AMTLogMgr` et `EnvAnalysis`. `AMTLogMgr` s'occupe de la journalisation pour la gestion des T_{AMT} actives. `EnvAnalysis` décider de la façon d'exécuter la T_{AMT}.

6.5.2 Composant TMAgent

Le composant `TMAgent` est installé sur des stations bases ou sur des unités fixes pouvant communiquer avec les unités mobiles. Il sert à faciliter la communication entre les TMClients et TMServeurs lorsque l'un d'entre eux est localisé sur une unité mobile. Les principales fonctions de son interface sont :

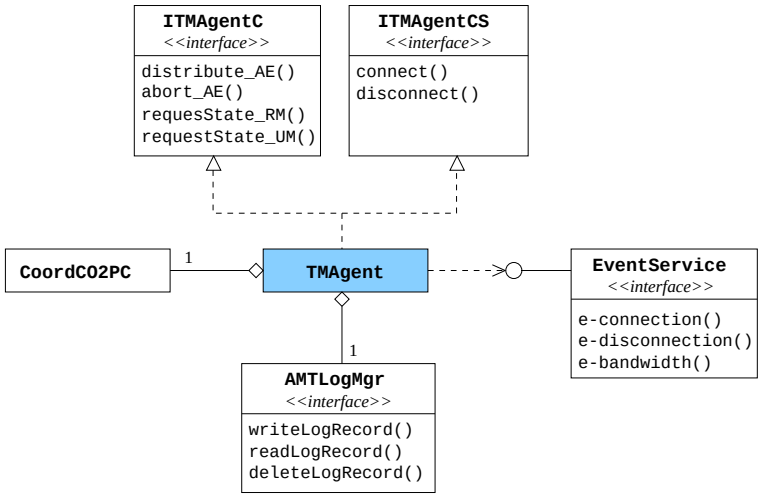

FIG. 6.15 – *Diagramme de classes du TMAgent.*

- `distribute_AE(alternative ae)` distribue les transactions composantes d'une AE_k. Son paramètre est l'alternative d'exécution à distribuer.
- `abort_AE(alternative ae)` annule les transactions composantes d'une AE_k. Son paramètre est l'alternative d'exécution à distribuer.
- `requestState_RM(caracteristic c)` fournit l'état du réseau mobile. Cette fonction a comme paramètre les caractéristiques du réseau mobile dont on veut connaître l'état (débit-bande-passante, prix-communication, etc.).
- `requestState_UM(identifiant idum, caracteristic c)` fournit l'état de l'unité mobile qui possède un SGBD. Cette fonction a comme paramètre les caractéristiques de l'unité mobile dont on veut connaître l'état (batterie-disponible, capacity-calcul, etc.).
- `connect(identifiant idum)` permet de demander une connexion. Le paramètre de cette fonction est l'identifiant de l'unité mobile qui désire se connecter au réseau.

– `disconnect(identifiant idum)` permet de demander une déconnexion. Le paramètre de cette fonction est l'identifiant de l'unité mobile qui désire se déconnecter du réseau.

Dans la figure 6.15 nous pouvons voir que l'interface ITMAgentC est utilisée par les TMClients et l'ITMAgentCS par les TMClients et TMServeurs.

Le `TMAgent` est composé d'un gestionnaire de journal qui gère et stocke de l'information nécessaire pour assurer la permanence des unités mobiles dans le système – `AMTLogMgr`. Le `TMAgent` a la capacité de percevoir l'état de la communication sans fil. Pour cela, il utilise un service événements – `EventService`. Le `TMAgent` peut également connaître l'état des sites où se localisent les `TMServeurs`. Ainsi, à travers le `TMAgent`, les `TMClients` peuvent connaître l'état de l'environnement mobile.

Du fait que les unités mobiles ne peuvent pas jouer le rôle de coordinateur dans le protocole de validation, `TMAgent` implante ce rôle – `CoordC02PC`. Le `TMAgent` joue le rôle de coordinateur lorsque dans l'exécution d'une T_{AMT} aucun des participants n'est localisé sur une UF.

6.5.3 Composant TMServeur

Le composant `TMServeur` peut être localisé sur des sites mobiles ou fixes. Son but principal est de communiquer avec les SGBD sous-jacents. Il fournit une interface par laquelle les `TMClients` et `TMAgents` peuvent demander l'exécution des transactions composantes. Les principales fonctions de son interface sont :

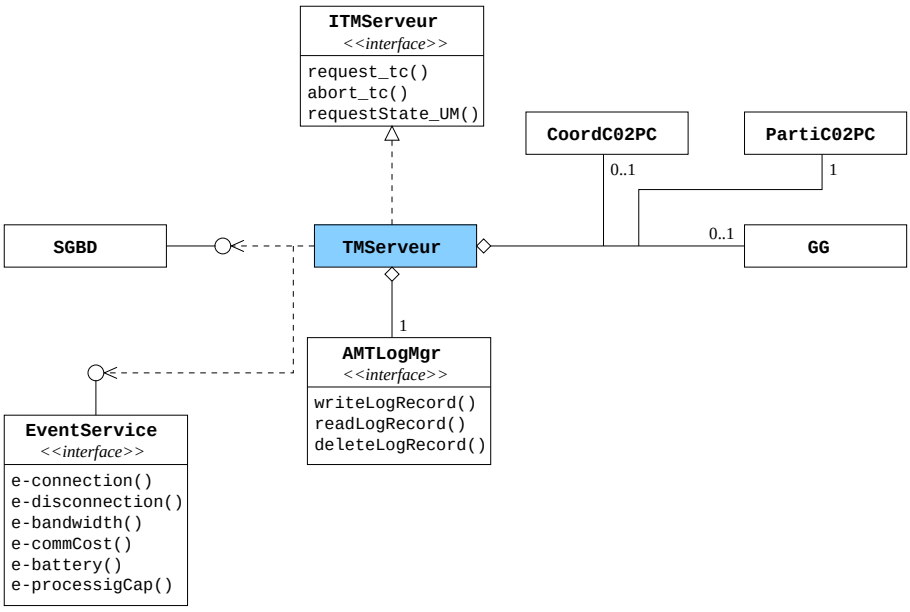

FIG. 6.16 – *Diagramme de classes du TMServeur.*

- `request_tc(transaction tc)` permet de demander l'exécution d'une transaction composante. Le paramètre de cette fonction est la transaction composante à exécuter.
- `annule_tc(transaction tc)` permet de demander l'annulation d'une transaction composante. Le paramètre de cette fonction est la transaction composante à annuler.
- `requestState_UM(identifiant idum, caracteristic c)` fournit l'état de l'unité mobile. Le paramètre de cette fonction contient les caractéristiques de l'unité mobile dont on veut connaître l'état.

Le `TMServeur` communique avec le SGBD sous-jacent par l'intermédiaire des interfaces Tx et SQL standards. Il peut fournir l'état des ressources de l'unité mobile à l'aide d'un service d'événements implanté par `EventService`.

Afin d'assurer une validation globale, le `TMServeur` participe à l'exécution du protocole CO2PC. Pour cela, il implante le module `PartiCO2PC`. Si le `TMServeur` est localisé sur une unité fixe, il peut coordonner le protocole CO2PC. C'est pour cela qu'il peut ou non implanter le module `CoordCO2PC`.

Un `TMServeur` peut implanter le module qui gère le graphe de sérialisabilité globale (GG) comme il a été défini dans la section 6.4.2.2.

6.5.4 Interaction des trois composants

La figure 6.17 illustre l'interaction des trois tiers. Nous pouvons voir que l'application utilise l'interface du `TMClient` pour exécuter/annuler des T_{AMT} . Le `TMClient` utilise celle des `TMAgent` et `TMServeur` pour exécuter/annuler les AE_k (et t_{ki}) ainsi que pour connaître l'état de l'environnement mobile. Lorsque le `TMServeur` est sur une unité mobile, il utilise l'interface du `TMAgent` pour se déconnecter/connecter du réseau. Finalement, le `TMAgent` utilise l'interface du `TMServeur` pour exécuter/annuler les transactions composantes demandées par le `TMClient`.

Le processus de demande d'exécution d'une T_{AMT} peut se dérouler comme suit. L'application appelle la fonction `request_AMT(transamt amt)`. Le `TMClient` vérifie l'état de l'environnement. Pour cela, il utilise les fonctions `requestState_UM(identifiant idum, caracteristic c)` et `requestState_RM(caracteristic c)`. Afin de connaître l'état des unités mobiles contenant les SGBD sous-jacents, il peut communiquer avec son `TMAgent` ou directement avec les `TMServeurs`. Une fois que le `TMClient` connaît l'état courant de l'environnement, il le compare avec les descripteurs d'environnement des AE_k composant la T_{AMT} demandée. Lorsqu'il choisit l'AE_k à exécuter, il s'adresse à son `TMAgent` pour lui demander la distribution des transactions composantes (`distribute_AE(alternative ae)`) ou il fait directement la distribution aux `TMServeurs` (`request_tc(transaction tc)`).

Le processus d'annulation d'une T_{AMT} peut se dérouler comme suit. L'application appelle la fonction `abort_AMT(transamt amt)`. Cette annulation entraîne celle de l'AE_k et des t_{ki}. Le `TMClient` peut demander directement l'annulation des t_{ki} (`abort_tc(transaction tc)`) ou le faire par l'intermédiaire du TMAgent en demandant l'annulation de l'AE_k correspondante (`abort_AE(alternative ae)`). Si l'unité mobile est déconnectée, la demande d'annulation sera faite lors de la prochaine reconnexion.

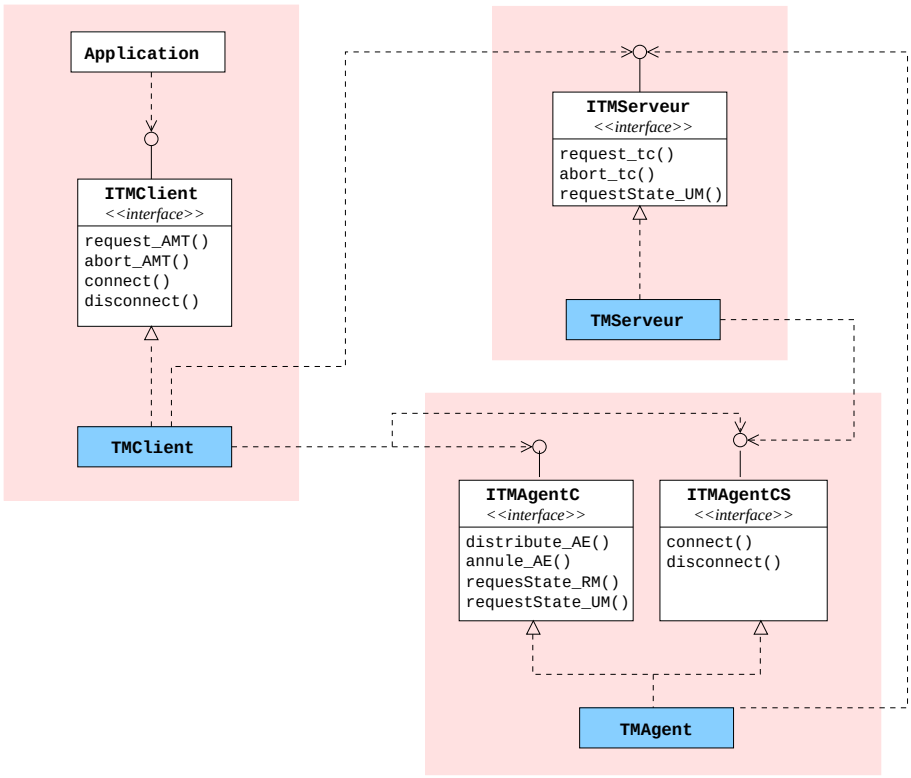

FIG. 6.17 – *Diagramme de classes des trois tiers.*

6.6 Le prototype TransMobi

Nous avons développé un prototype de l'intergiciel TransMobi. Ce prototype implante quelques-uns des composants décrits dans la section précédente.

6.6.1 Environnement matériel

Les unités mobiles utilisées sont les suivantes. Un ordinateur portable Compaq Armada 1700, avec un processeur Pentium II et 96 MB de RAM. Le système d'exploitation utilisé est Windows 98. Un PDA de type IPAQ H3850 de Compaq avec un processeur ARM SA1110 et 64 MB de RAM. Le système d'exploitation est Pocket PC version 3.0.

Nous avons utilisé également un serveur Linux installé sur un PC sur le réseau fixe.

Le réseau sans fil utilisé est un *Wireless Local Area Network* (WLAN) IEEE 802.11b appelé aussi WI-FI (*Wireless Fidelity*). L'environnement réseau comprend quatre stations base (*access point*) de type Orinoco AP-500. Le nombre de clients supportés dans une même cellule peut être de 50. La capacité de transmission peut atteindre 11 MB/s et la zone géographique couverte peut être de 550 m. La capacité de transmission des clients dépend

du nombre d'utilisateurs dans une même cellule, du type de zone couverte (à l'air libre, dans un bâtiment, etc.) et de la distance du client à la station base.

6.6.2 Environnement logiciel

Le développement a été en PersonalJava [Per00].[5] PersonalJava a été conçu pour les dispositifs comme les *set-top boxes* ou les téléphones mobiles. Il s'agit d'une version légère dérivée du JDK 1. Notre choix a été motivé par le fait que PersonalJava est gratuit et indépendant de la plate-forme utilisée.

En ce qui concerne l'environnement d'exécution, au début nous avons utilisé celui de PersonalJava pour Windows CE. Les besoins de mémoire sont d'environ 2.5 MB. Cependant nous avons dû utiliser une version bêta compatible avec les microprocesseurs ARM car la version normale supporte seulement les processeurs MIPS et SH3. Plus tard nous avons utilisé l'environnement d'exécution Jeode*Runtime* [Jeo], une mise en œuvre de PersonalJava qui prends 3.5 MB environ de mémoire pour exécuter les applications correctement. Jeode*Runtime* est compatible avec un grand nombre de processeurs : ARM, StrongARM, XScale, x86, MIPS, PowerPC et SuperH. Il s'est avéré plus stable et performant que la version bêta de PersonalJava.

Comme système gestionnaire de bases de données nous avons utilisée une version de test de PointBase (cf. section 3.1.13). Notre choix a été motivé par les caractéristiques de PointBase :

- il est développé complètement en Java et peut tourner sous n'importe quel système d'exploitation qui supporte une machine virtuelle Java ;
- il a une empreinte légère, entre 45 KB et 1 MB ;
- il est directement embarqué (*embedded*) dans les applications, pas besoin d'installation ;
- il supporte plusieurs connexions JDBC d'une même application tournant sur la même machine virtuelle ;
- il supporte des transactions réparties dans la version PointBase *embedded*, notamment, l'interface de 2PC est implantée ;
- il fournit les quatre niveaux d'isolation de l'ANSI SQL. Nous avons utilisé le niveau sérialisable.

Nous avons implanté le noyau minimal des trois tiers de TransMobi. Notre but a été de montrer leur faisabilité. Afin de montrer l'utilisation de Transmobi, une petite application d'achat électronique a été mise en œuvre. Les cinq modèles d'exécution on été implantés. Nous nous sommes concentrés surtout sur des aspects de communication. Ni l'adaptation d'OTM ni le protocole CO2PC ont été implantés.

Un stock de produits à vendre a été embarqué sur chacun des sites (portable, PDA et serveur Linux). Chaque unité mobile peut se connecter au serveur Linux pour faire une vente (modèle d'exécution 1) mais aussi, chacune des unités mobiles peut vendre de produits de

5. PersonalJava est maintenant remplacé par la famille Java 2 Micro Edition (J2ME) [J2M].

manière indépendante (modèle d'exécution 2). Les trois sites peuvent également participer à une vente collective (modèles 3, 4 et 5).

6.7 Conclusion

Dans ce chapitre, nous avons présenté TransMobi, un intergiciel pour la gestion de transactions mobiles T_{AMT}. TransMobi suit une architecture à trois tiers (client-agent-serveur). Il facilite l'interaction des SGBD faiblement couplés localisés sur des UM ou des unités fixes. Il est général grâce au fait qu'il préserve l'autonomie des sites et permet leur hétérogénéité. Un aspect très intéressant de TransMobi est le fait que les SGBD sous-jacents n'ont pas besoin d'être modifiés ni de fournir une interface particulière pour participer à l'exécution d'une transaction mobile. Il suffit de fournir des interfaces transactionnelles classiques comme *begin, abort, commit*. Les principales fonctions de TransMobi sont :

- l'adaptation des transactions T_{AMT} selon le contexte ;
- la garantie des propriétés d'atomicité sémantique des alternatives d'exécution (AE_k), de sérialisabilité globale des AE_k et de semi-atomicité des T_{AMT} ;
- la gestion de la déconnexion des UM participant dans une T_{AMT} ;

L'adaptation des T_{AMT} est possible grâce à la perception du contexte qui est faite par un service d'événements. Les différentes caractéristiques de l'environnement mobile (bande passante, prix de communication, etc.) sont surveillées afin de produire des événements qui sont notifiés à TransMobi. La production et le type de notification des événements sont faits de manière *ad hoc* selon le contexte applicatif.

Pour assurer l'atomicité sémantique des AE_k, nous avons proposé le protocole CO2PC (Combinaison d'une approche Optimiste et du protocole 2PC) qui offre une grande flexibilité lors de la validation des alternatives d'exécution. Il permet la validation optimiste (unilatérale) ou non-optimiste des transactions composantes et il limite le nombre de messages nécessaires à 2 messages par participant. La validation optimiste permet la libération des ressources avant la validation globale. Ceci est un grand avantage dans un contexte mobile. En cas d'annulation globale après une validation optimiste, des transactions de compensation sont exécutées sans entraîner des annulations en cascade.

Pour assurer un ordre d'exécution sérialisable des AE_k, nous avons adapté la méthode OTM. Malgré l'ordre assuré, une cohérence sémantique est offerte à cause de l'utilisation de transactions de compensation. Grâce à l'utilisation d'OTM en combinaison avec CO2PC, il y a pas génération de nouveaux messages sur le réseau sans fil.

Les deux protocoles (OTM et CO2PC) préservent l'autonomie des sites car ils ne demandent aucune information particulière aux SGBD. Dans le cas d'une validation non-optimiste dans CO2PC, l'interface de "préparation" du protocole 2PC est requise. Cependant, comme il s'agit d'une interface largement implantée, nous considérons que son utilisation n'affecte ni l'autonomie, ni l'hétérogénéité des sites. Si un site ne fournit pas l'état préparation, il peut participer dans la validation d'une manière optimiste. La contrainte posée est qu'en cas d'annulation globale après une validation optimiste, des transactions

de compensation doivent être exécutées. Par ailleurs, l'hétérogénéité est également préservée. Ni OTM ni CO2PC demandent l'utilisation d'un protocole de contrôle de concurrence particulier ou une manière de gérer la tolérance aux pannes (journalisation), etc.

Aussi bien CO2PC que l'adaptation d'OTM supportent les déconnexions pendant leur déroulement.

La surveillance du contexte et la gestion des événements peuvent entraîner un coût supplémentaire : complexité, dépense de batterie, besoin de calcul, de communication, etc. Nous n'avons pas fait d'analyse à ce sujet, cependant, nous estimons que le coût de surveillance des différentes caractéristiques de l'environnement peut être très variable et des compromis peuvent être faits afin d'économiser les ressources. En effet, il faut faire un compromis entre le gain obtenu par l'exécution des T_{AMT} et le coût généré par la perception de l'environnement.

CHAPITRE 7

Conclusion et perspectives

Dans ce dernier chapitre nous faisons un bilan de nos principales contributions (section 7.1) et nous introduisons quelques perspectives de recherche (section 7.2).

7.1 Bilan des contributions

A l'heure actuelle, la recherche dans le domaine de l'informatique mobile connaît une activité très importante, en particulier, sur les aspects liés aux bases de données [BBOB+03]. Nous considérons qu'un système mobile comprend une application avec des utilisateurs mobiles ou fixes qui accèdent aux données localisées sur des sites également mobiles ou fixes. A cause des caractéristiques de l'environnement mobile, les techniques traditionnelles de duplication, de gestion de cache, de requêtes, de transactions, etc. deviennent inadéquates telles quelles.

Dans notre travail, nous nous sommes intéressés particulièrement à la gestion des transactions. Du fait des limitations des unités mobiles et des variations d'état du réseau sans fil, les transactions sont sujettes à des annulations fréquentes et à des coûts d'exécution imprévus fréquemment élevés. Le coût étant représenté par le temps d'exécution, la dépense d'énergie, le prix de communication, etc.

Afin de limiter les répercussions du contexte mobile sur les transactions, nous pensons qu'il est nécessaire d'adapter l'exécution des transactions aux variations de l'environnement. Pour ce faire, il faut d'une part connaître le contexte et ses différentes variations et d'autre part, exécuter les transactions de manière convenable au contexte. Nous avons identifié cinq modèles d'exécution pour les transactions mobiles (cf. section 2.1.2). Chacun d'entre eux a besoin d'un contexte particulier. Nous croyons intéressant de choisir l'exécution de l'un des cinq modèles selon le contexte courant.

Dans la suite, nous rappelons notre étude de l'état de l'art (section 7.1.1) qui a motivé nos contributions du point de vue de modèle (section 7.1.2) mais aussi du point de vue de la gestion transactionnelle (sections 7.1.3 et 7.1.4).

7.1.1 Transactions multibases et mobiles

Dans notre étude de l'état de l'art, premièrement, nous avons analysé si les techniques transactionnelles multibases sont applicables à l'environnement mobile. Deuxièmement, nous avons analysé diverses propositions sur les transactions mobiles.

Nous avons fait un rapprochement des systèmes mobiles avec les multibases car les systèmes mobiles ont besoin d'autonomie, du fait des limitations de communication. En effet, lorsqu'une unité mobile se déconnecte, elle a besoin de continuer à travailler de manière indépendante. Egalement, la faiblesse de la bande passante et le prix élevé des communications amènent à restreindre les échanges d'information et ainsi le partage entre les composants du système. Par ailleurs, l'hétérogénéité est présente du fait de la diversité des unités mobiles et de la découverte de nouveaux serveurs de données au fur et à mesure des déplacements.

Après avoir analysée certaines techniques transactionnelles multibases, nous concluons qu'elles peuvent être applicables aux systèmes mobiles lorsqu'elles :

- relâchent les propriétés ACID ;
- préservent l'autonomie et permettent l'hétérogénéité des SGBD participant dans le système.

Cependant, les techniques considérées ne peuvent pas être directement appliquées. Il faut faire quelques adaptations afin de gérer de manière convenable les variations de l'environnement mobile.

Cette nécessité a motivé de nombreuses propositions sur les transactions mobiles. Ces propositions sont souvent orientées vers des contextes spécifiques. Les variations de l'environnement mobile ne sont pas suffisamment prises en compte puisque, généralement, seules les déconnexions des unités mobiles sont supportées. Par ailleurs, dans ces travaux, la possibilité d'exécuter les transactions selon le contexte disponible est très limitée. Certains travaux proposent d'exécuter les transactions soit sur une unité mobile soit sur des unités fixes selon l'état de la connexion (connecté, déconnecté). L'exécution répartie entre unités fixes et mobiles ou entre plusieurs unités mobiles est quasiment ignorée.

Au vue de l'état de l'art, nos contributions portent sur une approche générale qui permet l'exécution des transactions selon le contexte disponible et qui ne cible pas d'application particulière. D'une part, nous proposons un modèle de transactions mobiles adaptables (AMT) permettant de définir des transactions qui seront exécutées de la manière la plus adaptée au contexte courant de l'environnement. D'autre part, nous proposons un intergiciel (TransMobi) qui met en œuvre l'adaptation et implante les protocoles adéquats pour l'exécution des transactions de type AMT. Parmi les protocoles utilisés, nous proposons le protocole de validation CO2PC.

7.1.2 Modèle AMT

Le modèle transactionnel AMT est très flexible. Il offre la possibilité d'adapter les transactions au contexte et de limiter le coût d'exécution des transactions selon les critères de qualité acceptables pour l'application. Un critère de qualité peut être un compromis entre la qualité de résultats acceptée et le coût d'exécution estimé.

Une transaction T_{AMT} est composée d'un ensemble **d'alternatives d'exécution** sémantiquement équivalentes. Chacune des alternatives est composée d'un ensemble de **transac-**

tions composantes et d'un **descripteur d'environnement** qui décrit le contexte dans lequel l'alternative doit être déclenchée. Le contexte peut comprendre le débit et le prix de communication, l'état de la batterie de l'unité mobile, etc. Selon l'état courant de l'environnement, une seule alternative, la plus adéquate, est déclenchée ou alors la T_{AMT} est mise en attente. Les transactions composantes sont exécutées par les SGBD sous-jacents sur des sites fixes ou mobiles.

Afin de faciliter l'adaptation, le modèle permet de définir des alternatives en utilisant les cinq modèles d'exécution. Ainsi, selon le contexte, différents modèles d'exécution peuvent être utilisés. Par exemple, si la bande passante est faible alors une exécution complète sur l'unité mobile est préférable à une exécution répartie entre l'unité mobile et des unités fixes.

Le modèle AMT relâche les propriétés ACID et s'inspire des transactions Sagas [GMS87], Flexibles [ELR90] et DOM [BOH+92]. Les propriétés d'une T_{AMT} sont les suivantes : (1) les transactions composantes ont les propriétés AID, (2) les alternatives d'exécution ont les propriétés d'atomicité sémantique et de sérialisabilité globale et (3) la T_{AMT} a la propriété de semi-atomicité. L'exécution des T_{AMT} offre une cohérence sémantique à cause de l'atomicité sémantique. Nous avons fait une spécification formelle du modèle AMT en utilisant le formalisme ACTA [CR94] qui nous a permis de spécifier sa structure et ses propriétés.

L'atomicité sémantique a été choisie car elle permet une validation optimiste (anticipée) de transactions composantes. Ceci est un grand avantage dans notre contexte. L'atomicité sémantique se base sur la compensation des transactions validées avant une annulation globale. L'utilisation des transactions de compensation comme moyen de récupération pourrait être remise en cause par sa complexité d'implantation/réalisation. Nous pensons que notre approche peut être modifiée afin de supporter d'autres méthodes de récupération (section 2.2.2) si elles sont plus convenables au contexte applicatif (comme la reprise de mises à jour (*redo*) [MRKS92b, MRB+01] ou la reprise totale (*retry*) [MRKS92a]).

Afin de limiter l'incohérence du système global, nous avons décidé d'offrir la sérialisabilité globale. Cependant, ceci pourrait être considéré comme une contrainte forte pour un environnement mobile. Nous considérons que le modèle AMT peut être étendu afin d'utiliser d'autres formes de cohérence plus souples (section 2.2.1), comme la "sérialisabilité locale" [MRKS91a], la "sérialisabilité à deux niveaux" [MRKS91b, MRKS98] ou "l'epsilon-sérialisabilité" [PL91, WYP92, PL92].

Une étude analytique du modèle AMT a été faite. Nos objectifs ont été de : (1) montrer le gain qu'est possible d'obtenir avec l'utilisation des T_{AMT} et de (2) donner une idée de la manière de concevoir les T_{AMT}. En effet, l'étude montre le gain qui peut être obtenu avec notre approche par rapport aux approches où l'adaptation au contexte n'est pas considérée. Par ailleurs, le modèle T_{AMT} ajoute une certaine complexité à la conception des transactions. La description du contexte d'exécution, demande une connaissance assez précise de l'environnement mobile et de ses variations. Afin de pallier à cette complexité, notre étude donne une idée de la manière de décrire le contexte d'exécution. Elle montre également, comment définir les alternatives d'une T_{AMT}.

7.1.3 Intergiciel TransMobi

TransMobi est un intergiciel facilitant l'interaction des SGBD faiblement couplés localisés sur des unités mobiles ou des unités fixes, à travers des transactions T_{AMT}. Il s'agit

d'une solution générale permettant de préserver l'autonomie des sites et autorisant leur hétérogénéité. En effet, Les SGBD participant dans le système mobile ne sont pas contraints à modifier leur mode de fonctionnement et aucune supposition est faite sur leur façon d'assurer les propriétés ACID. Un aspect intéressant de TransMobi est que les SGBD sous-jacents n'ont pas besoin d'une interface particulière pour participer à l'exécution d'une transaction mobile T_{AMT}. Il suffit de fournir des interfaces classiques pour la gestion de transactions (*begin, abort, commit*).

Les principales fonctions de TransMobi sont : adapter les transactions T_{AMT} selon le contexte d'exécution ; gérer la déconnexion des unités mobiles participant à une T_{AMT} ; assurer les propriétés d'atomicité sémantique et de sérialisabilité globale des alternatives d'exécution, ainsi que la propriété de semi-atomicité des T_{AMT}. Nous considérons que les SGBD sous-jacents assurent les propriétés ACID des transactions composantes.

Afin de rendre possible l'adaptation des T_{AMT}, TransMobi utilise un service d'événements qui surveille l'environnement et en produit des événements. La surveillance du contexte et la gestion des événements peuvent entraîner un coût supplémentaire pour l'exécution des applications : complexité, dépense de batterie, besoin de calcul et de communication, etc. Nous n'avons pas fait d'analyse à ce sujet. Cependant, nous estimons qu'il faut veiller à ce que la perception de l'environnement ne soit pas trop pénalisante. Nous considérons que des compromis peuvent être réalisés afin d'optimiser les ressources. En effet, il faut faire un compromis entre le gain obtenu par l'exécution des T_{AMT} et le coût généré par la perception de l'environnement.

Pour assurer un ordre d'exécution sérialisable des alternatives d'exécution, nous avons fait une adaptation à la méthode OTM qui est une méthode simple et facile à implanter. Elle préserve l'hétérogénéité des sites car elle n'impose pas un protocole de concurrence particulier. L'autonomie est préservée car aucune information liée à la gestion locale des sites n'est demandée. La méthode se base sur l'utilisation de "tickets". Notre adaptation a consistée à permettre la validation prématurée des transactions composantes.

7.1.4 Protocole de validation CO2PC

Nous avons proposé le protocole CO2PC (Combinaison d'une approche Optimiste et 2PC) qui assure l'atomicité sémantique. Cet apport est intéressant à souligner car il offre une grande flexibilité lors de la validation des alternatives d'exécution. Il permet la validation optimiste (anticipée) ou non-optimiste des transactions composantes d'une même alternative d'exécution et il limite le nombre de messages nécessaires à 2 par participant.

A la différence d'autres protocoles (TCOT [KPDS02], UCM [BPA00]) CO2PC préserve l'autonomie des sites car il ne demande aucune information particulière aux SGBD. Dans le cas d'une validation non-optimiste, l'interface de "préparation" du protocole 2PC [Gra78] est requise. Comme il s'agit d'une interface largement implantée, nous considérons que son utilisation n'affecte ni l'autonomie ni l'hétérogénéité des sites. Si un site ne fournit pas l'état préparation, il peut participer à la validation d'une manière optimiste. La contrainte posée est qu'en cas d'annulation globale après une validation optimiste, des transactions de compensation doivent être prévues et exécutées. Par ailleurs, l'hétérogénéité est également préservée.

Afin d'assouplir les contraintes imposées aux systèmes, CO2PC relâche les propriétés d'atomicité et de durabilité. En effet, le fait de permettre une validation anticipée nous oblige à relâcher l'atomicité et fournir l'atomicité sémantique. CO2PC ne s'occupe pas de la durabilité. Il offre aux SGBD la possibilité de décider de la manière d'assurer la durabilité des transactions validées. Un SGBD localisé sur une unité mobile est libre de décider de faire le transfert des modifications sur un support stable (unité fixe) : après la validation de chacune des transactions composantes, après la validation d'un nombre défini d'entre elles, lorsqu'il bénéficie d'une bonne connexion, etc.

Nous n'avons pas inclu de performances de CO2PC dans cette thèse. Cependant, nous avons commencé à analyser la comparaison de CO2PC avec d'autres protocoles similaires comme TCOT et UCM, mais aussi avec le protocole le plus utilisé, 2PC. Nous nous intéressons aux propriétés offertes, aux contraintes imposées, aux besoins en terme de communication, à leur comportement en présence de variations du contexte, etc.

7.2 Perspectives

La suite de notre travail de recherche porte sur plusieurs points. Dans cette section, nous introduisons ceux qui nous semblent être les plus importants.

Notre approche en présence de la duplication

Nous voulons analyser l'intégration de TransMobi avec des services NODS (cf. section 1.3), notamment avec celui de duplication [Dra03]. Dans l'état actuel de notre proposition, nous considérons que les sources de données composant le système mobile sont faiblement couplées. Notre approche peut être étendue afin de prendre en compte la duplication et éventuellement la synchronisation de données. Pour cela, il faut se concentrer sur des aspects de cohérence "locale et globale" [Dra03]. Le but est de: (1) re-analyser le modèle AMT afin de relâcher éventuellement certaines des ces propriétés et (2) faire coopérer TransMobi avec un service de duplication qui offre un protocole de cohérence locale adéquat pour l'environnement mobile.

Outils d'aide à la conception des T_{AMT}

La surveillance du contexte mobile peut faciliter la (re)configuration des transactions T_{AMT}. La perception de l'environnement, fournie par un service d'événements, peut être utilisée pour profiler le contexte d'exécution. Nous pensons qu'une fois l'environnement profilé, un calcul analytique (comme celui présenté dans ce travail) peut être fait afin d'optimiser les T_{AMT} selon le critère de qualité requis par l'application. En effet, selon les variations dans le patron de comportement du contexte, on pourrait concevoir des outils qui suggérent la (re)configuration des T_{AMT}.

Adaptation dynamique

A l'heure actuelle, notre travail se concentre sur ce que nous pouvons appeler "l'adaptation statique". L'exécution d'une transaction est adaptée au contexte au moment de son lancement. Cependant, des variations de contexte peuvent survenir après le démarrage de la transaction. Afin de s'adapter à ces variations, nous voulons étendre notre étude vers "l'adaptation dynamique". Pour cela, il est nécessaire d'approfondir les aspects liés à la réutilisation du travail déjà fait par une transaction. L'objectif est d'analyser la compatibilité des alternatives d'exécution pour ne pas annuler complètement le travail effectué par l'une d'entre elles, mais de le réutiliser de manière adéquate dans une autre alternative.

Application de notre approche à d'autres contextes

La motivation de notre approche a été l'environnement mobile de type cellulaire. Cependant, d'autres contextes également très variables comme les réseaux *ad hoc* ou les contextes pair à pair (*peer to peer*) peuvent être envisagés pour nos propositions. Ces contextes sont semblables à celui que nous considérons dans notre travail. La différence majeure est le fait que les sites peuvent établir une communication sans utiliser des stations base comme intermédiaires. Nous considérons que le modèle transactionnel AMT peut être utilisé sans aucune adaptation. Cependant, pour TransMobi, il est nécessaire de faire quelques modifications afin de délocaliser ou de remplacer d'une manière "sûre" les fonctions propres au tiers agent (support des déconnexions, représentation des unités mobiles lors des déconnexions, etc.) mais aussi des fonctions comme la coordination de la validation ou la gestion du graphe de sérialisabilité globale.

Bibliographie

[AAFZ96] S. Acharya, R. Alonso, M. Franklin et S. Zdonik. Prefetching from a Broadcast Disk. Dans *Int. Conf. on Data Engineering (ICDE)*, New Orleans, USA, February 1996.

[ABP03] N. Anciaux, L. Bouganim et P. Pucheral. Memory Requirements for Query Execution in Highly Constrained Devices. Dans *Int. Conf. on Very Large Databases (VLDB)*, Berlin, Germany, September 2003.

[Adi00] M. Adiba. Transactions, Concepts Généraux, Concurrence et Reprise, Transactions Longues, Travail Coopératif, Formalisation. Notes de Cours DEA - Université de Grenoble, 2000.

[AGMS87] R. Alonso, H. Garcia-Molina et K. Salem. Concurrency Control and Recovery for Global Procedures in Federated Database Systems. *IEEE Data Engineering Bulletin*, 10(3), 1987.

[BBG+95] H. Berenson, P. A. Bernstein, J. Gray, J. Melton, E. J. O'Neil et P. E. O'Neil. A Critique of ANSI SQL Isolation Levels. Dans *ACM SIGMOD Conference*, San Jose, USA, May 1995.

[BBOB+03] G. Bernard, J. Ben-Othman, L. Bouganim, G. Canals, B. Defude, J. Ferrié, S. Gançarski, R. Guerraoui, P. Molli, P. Pucheral, C. Roncancio, P. Serrano-Alvarado et P. Valduriez. Mobilité et Bases de Données : Etat de l'Art et Perspectives (Partie I et II). *Chronique Technique et science informatiques (TSI)*, 22(3 and 4), 2003.

[BBPV02] C. Bobineau, L. Bouganim, P. Pucheral et P. Valduriez. PicoDBMS: Scaling Down Database Techniques for the Smartcard. Dans *Int. Conf. on Very Large Databases (VLDB)*, Cairo, Egipt, September 2002.

[BCF+97] J. Besancenot, M. Cart, J. Ferrié, R. Guerraoui, P. Pucheral et B. Traverson. *Les Systèmes Transactionnels : Concepts, Normes et Produits*. Editions Hermès, 1997.

[BG84] P. A. Bernstein et N. Goodman. An Algorithm for Concurrency Control and Recovery in Replicated Distributed Databases. *ACM Transactions on Database Systems (TODS)*, 9(4), 1984.

[BGMS92] Y. Breitbart, H. Garcia-Molina et A. Silberschatz. Overview of Multidatabase Transaction Management. *Very Large Databases (VLDB) Journal*, 1(2), 1992.

[BHG87] P. A. Bernstein, V. Hadzilacos et N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley Publisher, 1987.

[BOH+92] A. Buchmann, M. Tamer Özsu, M. Hornick, D. Georgakopoulos et F. A. Manola. *Database Transaction Models for Advanced Applications*, chapitre 5, A Transaction Model for Active Distributed Object Systems. Morgan Kaufmann Publisher, 1992.

[BP98] S. Balasubramaniam et B. C. Pierce. What is a file synchronizer? Dans *Int. Conf. on Mobile Computing and Networking (MobiCom)*, Dallas, USA, October 1998.

[BPA00] C. Bobineau, P. Pucheral et M. Abdallah. A Unilateral Commit Protocol for Mobile and Disconnected Computing. Dans *Int. Conf. Parallel and Distributed Computing Systems (PDCS)*, Las Vegas, USA, August 2000.

[BS88] Y. Breitbart et A. Silberschatz. Multidatabase Update Issues. Dans *ACM SIGMOD Conference*, Chicago, USA, September 1988.

[Chr91] P. K. Chrysanthis. *ACTA, A Framework for Modeling and Reasoning about Extended Transactions.* PhD thesis, University of Massachusetts, Amherst, USA, 1991.

[Chr93] P. K. Chrysanthis. Transaction Processing in a Mobile Computing Environment. Dans *IEEE Workshop on Advances in Parallel and Distributed Systems (APADS)*, Princeton, USA, October 1993.

[CL91] M. J. Carey et M. Livny. Conflict Detection Tradeoffs for Replicate Data. *ACM Transactions on Database Systems (TODS)*, 16(4), 1991.

[Col00] C. Collet. The NODS project : Networked Open Database Services. Dans *ECOOP*, Sophia Antipolis, France, June 2000.

[CR90] P. K. Chrysanthis et K. Ramamritham. ACTA: A framework for Specifying and Reasoning about Transaction Structure and Behavior. Dans *ACM SIGMOD Conference*, Atlantic City, USA, May 1990.

[CR91] P. K. Chrysanthis et K. Ramamritham. A Formalism for Extended Transaction Model. Dans *Int. Conf. on Very Large Databases (VLDB)*, Barcelona, Spain, September 1991.

[CR92] P. K. Chrysanthis et K. Ramamritham. *Database Transaction Models for Advanced Applications*, chapitre 10, ACTA: The SAGA Continues. Morgan Kaufmann Publisher, 1992.

[CR94] P. K. Chrysanthis et K. Ramamritham. Synthesis of Extended Transaction Models Using ACTA. *ACM Transactions on Database Systems (TODS)*, 19(3), 1994.

[DE89] W. Du et A. K. Elmagarmid. Quasi serializability: A Correctness Criterion for Global Concurrency Control in InterBase. Dans *Int. Conf. on Very Large Databases (VLDB)*, Amsterdam, 1989.

[DG98] R. A. Dirckze et L. Gruenwald. A Toggle Transaction Management Technique for Mobile Multidatabases. Dans *Int. Conf. on Information and Knowledge Management (CIKM)*, Bethesda, USA, November 1998.

[DG00] R. A. Dirckze et L. Gruenwald. A Pre-Serialization Transaction Management Technique for Mobile Multidatabases. *Mobile Networks and Applications (MONET)*, 5(4), 2000.

[DH95] M. H. Dunham et Abdelsalam Helal. Mobile Computing and Databases: Anything New? *ACM SIGMOD Record*, 4(4), 1995.

[DHB97] M. H. Dunham, A. Helal et S. Balakrishnan. A Mobile Transaction Model that Captures Both the Data and the Movement Behavior. *ACM/Baltzer Journal on special topics in mobile networks and applications*, 2(2), 1997.

[DK98a] M. H. Dunham et V. Kumar. Defining Location Data Dependency, Transaction Mobility and Commitment. Rapport de recherche 98-CSE-01, Southern Methodist University, Dallas, USA, February 1998.

[DK98b] M. H. Dunham et V. Kumar. Location Dependent Data and its Management in Mobile Databases. Dans *Int. DEXA Workshop on Mobility in Databases and Distributed Systems*, Vienna, Austria, August 1998.

[DK99] M. H. Dunham et V. Kumar. Impact of Mobility on Transaction Management. Dans *Int. Workshop on Data Engineering for Wireless and Mobile Access (MobiDE)*, Seattle, USA, August 1999.

[Dra03] S. Drapeau. *RS2.7: un Canevas Adaptable de Services de Duplication.* PhD thesis, Institut National Politecnique de Grenoble, June 2003. In French.

[Duo03] Phuong-Quynh Duong. *La Tolérance aux Fautes pour les Systèmes à Composants.* PhD thesis, Institut National Politecnique de Grenoble, May 2003. In French.

[DVCK99] A. Datta, D. E. VanderMeer, A. Celik et V. Kumar. Broadcast Protocols to Support Efficient Retrieval from Databases by Mobile Users. *ACM Transactions on Database Systems (TODS)*, 24(1), 1999.

[EGLT76] K. P. Eswarn, J. Gray, R. A. Lorie et I. L. Triger. The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM (CACM)*, 19(11), 1976.

[Elm92] A. K. Elmagarmid, éditeur. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann Publishers, 1992.

[ELR90] A. K. Elmagarmid, Y. Leu et M. Rusinkiewics. A Multidatabase Transaction Model for INTERBASE. Dans *Int. Conf. on Very Large Databases (VLDB)*, Brisbane, Australia, August 1990.

[Fas02] FastObjects by Poet. FastObjects j2 http://www.fastobjects.com/, 2002.

[FJL00] M. Frodigh, P. Johansson et P. Larsson. Wireless Ad hoc Networking - The art of networking without a network. *Ericsson Review*, 4, 2000.

[FS99] J. Flinn et M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. Dans *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, New Orleans, USA, February 1999.

[FZ96] M. Franklin et S. Zdonik. Dissemination-Based Information Systems. *IEEE Data Engineering Bulletin*, 19(3), 1996.

[GB03] L. García-Bañuelos. *PERSEUS: Un Canevas Logiciel pour la Construction des Gestionnaires d'Objets Persistants.* PhD thesis, Institut National Politecnique de Grenoble, May 2003.

[GBE+00] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, et M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems (TODS)*, 25(1), 2000.

[GHPS96] J. Gray, P. Helland, P.O'Neil et D. Shasha. The Dangers of Replication and a Solution. Dans *ACM SIGMOD Conference*, Montreal, Canada, June 1996.

[GM83] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems (TODS)*, 8(2), 1983.

[GMS87] H. Garcia-Molina et K. Salem. Sagas. Dans *ACM SIGMOD Conference*, San Francisco, USA, May 1987.

[GN95] M. Gallersdörder et M. Nicola. Improving Performance in Replicated Databases Through Relaxed Coherency. Dans *Int. Conf. on Very Large Databases (VLDB)*, Zürich, Switzerland, 1995.

[GR93] J. Gray et A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publisher, 1993.

[Gra78] J. Gray. Notes on Database Operating Systems. Dans *Advanced Course: Operating Systems*, number 60 in LNCS, 1978.

[GRS91] D. Georgakopoulos, M. Rusinkiewicz et A.P. Sheth. On Serializability of Multidatabase Transactions Through Forced Local Conflicts. Dans *Int. Conf. on Data Engineering (ICDE)*, Kobe, Japan, April 1991.

[GRS94] D. Georgakopoulos, M. Rusinkiewicz et A.P. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 6(1), 1994.

[HH94] L. B. Huston et P. Honeyman. Peephole log Optimization. Dans *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, USA, December 1994.

[HR83] T. Härder et A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys*, 15(4), 1983.

[IB94] T. Imielinski et B. R. Badrinath. Wireless Mobile Computing: Challenges in Data Management. *Communications of the ACM (CACM)*, 37(10), 1994.

[IBM02a] IBM Software Products. DB2 Everyplace http://www-3.ibm.com/software/data/db2/everyplace/, 2002.

[IBM02b] IBM Software Products. IBM Cloudscape http://www-3.ibm.com/software/data/cloudscape/, 2002.

[IBM02c] IBM Software Products. WebSphere Everyplace Access http://www-3.ibm.-com/software/pervasive/products/mobile_apps/ws_everyplace_access.shtml, 2002.

[IDM91] J. Ioannidis, D. Duchamp et G. Q. Maguire. IP-Based Protocols for Mobile Internetworking. Dans *SIGCOMM*, Switzerland, 1991.

[IET] IETF. Mobile Ad-hoc Networks (manet) http://www.ietf.org/html.charters/-manet-charter.html.

[J2M] J2ME Personal Profile. http://java.sun.com/products/personalprofile/.

[JBE95] J. Jing, O. Bukhres et A. K. Elmagarmid. Distributed Lock Management for Mobile Transactions. Dans *Int. Conf. on Distributed Computing Systems (ICDCS)*, Vancouver, Canada, May 1995.

[Jeo] Jeode PDA Edition for Microsoft Pocket PC 2002. http://www.esmertec.com.

[JHE99] J. Jing, A.S. Helal et A. K. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2), 1999.

[KK00] K. Ku et Y. Kim. Moflex Transaction Model for Mobile Heterogeneous Multidatabase Systems. Dans *IEEE Workshop on Research Issues in Data Engineering*, San Diego, USA, February 2000.

[KLS90] H. F. Korth, E. Levy et A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. Dans *Int. Conf. on Very Large Databases (VLDB)*, Brisbane, Australia, August 1990.

[KPDS02] V. Kumar, N. Prabhu, M. H. Dunham et A. Y. Seydim. TCOT- A Timeout-Based Mobile Transaction Commitment Protocol. *IEEE Transactions on Computers*, 51(10), 2002.

[KPE92] E. Kühn, F. Puntigam et A. K. Elmagarmid. *Database Transaction Models for Advanced Applications*, chapitre 9, Multidatabase Transaction and Query Processing in Logic. Morgan Kaufmann Publisher, 1992.

[KR79] H. T. Kung et J. T. Robinson. On Optimistic Methods for Concurrency Control. Dans *Int. Conf. on Very Large Databases (VLDB)*, Rio de Janeiro, Brazil, October 1979.

[KR81] H. T. Kung et J. T. Robinson. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems (TODS)*, 6(2), 1981.

[KS87] R. Alonso K. Salem, H. Garcia-Molina. Altruistic Locking: A Strategy for Coping with Long Lived Transactions. Dans *Int. Workshop on High Performance Transaction Systems (HPTS)*, number 359 in LNCS, September 1987.

[KS92] J. J. Kistler et M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems (TOCS)*, 10(1), 1992.

[Kum00] V. Kumar. A Timeout-based Mobile Transaction Commitment Protocol. Dans *ADBIS-DASFAA Symp. on Advances in Databases and Information Systems*, volume 1884 of *LNCS*, Prague, Czech Republic, September 2000.

[LH02] M. Lee et S. Helal. HiCoMo: High Commit Mobile Transactions. *Kluwer Academic Publishers Distributed and Parallel Databases (DAPD)*, 11(1), 2002.

[LKS91] E. Levy, H. F. Korth et A. Silberschatz. A theory of Relaxed Atomicity. Dans *ACM SIGACT-SIGOPS Symp. on principles of Distributed Computing (PODC)*, Montreal, Canada, August 1991.

[LS94] Q. Lu et M. Satynarayanan. Isolation-Only Transactions for Mobile Computing. *ACM Operating Systems Review*, 28(2), 1994.

[LS95] Q. Lu et M. Satynarayanan. Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions. Dans *IEEE HotOS Topics Workshop*, Orcas Island, USA, May 1995.

[MB01] S. K. Madria et B. Bhargava. A Transaction Model for Improving Data Availability in Mobile Computing. *Kluwer Academic Publishers Distributed and Parallel Databases (DAPD)*, 10(2), 2001.

[MCFP96] C. Morpain, M. Cart, J. Ferrié et JF. Pons. Maintaining Database Consistency in Presence of Value Dependencies in Multidatabase Systems. Dans *ACM SIGMOD Conference*, Montreal, Canada, June 1996.

[McO02] McObject Solutions. eXtremeDb http://www.mcobject.com/extremedb.htm, 2002.

[Mor96] C. Morpain. *Interopérabilité des Systèmes de Bases de Données Fédérés: Critères de Correction des Exécutions.* PhD thesis, Université Montpellier II, 1996.

[Mos81] J. E. B. Moss. *Nested Transactions: An approach to Reliable Computing.* PhD thesis, Massachusetts Institute of Technology, Massachusetts, USA, 1981.

[MR91] P. Muth et T. C. Rakow. Atomic Commitment for Integrated Database Systems. Dans *Int. Conf. on Data Engineering*, Kobe, Japan, April 1991.

[MRB+92] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth et A. Silberschatz. The Concurrency Control Problem in Multidatabases: Characteristics and Solutions. Dans *ACM SIGMOD Conference*, San Diego, June 1992.

[MRB+01] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth et A. Silberschatz. Overcoming Heterogeneity and Autonomy in Multidatabase Systems. Dans *Information and Computation*, volume 167, 2001.

[MRKS91a] S. Mehrotra, R. Rastogi, H. F. Korth et A. Silberschatz. Maintaining Database Consistency in Heterogeneous Distributed Database Systems. Rapport de recherche TR-91-04, University of Texas at Austin, 1991.

[MRKS91b] S. Mehrotra, R. Rastogi, H. F. Korth et A. Silberschatz. Non-serializable Executions in Heterogeneous Distributed Database Systems. Dans *Int. Conf. on Parallel and Distributed Systems (PDIS)*, Miami, USA, 1991.

[MRKS92a] S. Mehrotra, R. Rastogi, H. F. Korth et A. Silberschatz. A Transaction Model for Mutidatabase Systems. Dans *Int. Conf. on Distributed Computing Systems (ICDCS)*, Yokohama, Japan, June 1992.

[MRKS92b] S. Mehrotra, R. Rastogi, H. F. Korth et A. Silberschatz. Ensuring Transaction Atomicity in Multidatabase Systems. Dans *ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, San Diego, USA, June 1992.

[MRKS98] S. Mehrotra, R. Rastogi, H. F. Korth et A. Silberschatz. Ensuring Consistency in Multidatabases by Preserving Two-Level Serializability. *ACM Transactions on Database Systems (TODS)*, 23(2), 1998.

[MV00] K. A. Momin et K. Vidyasankar. Flexible Integration of Optimistic and Pessimistic Concurrency Control in Mobile Environments. Dans *ADBIS-DASFAA Symp. on Advances in Databases and Information Systems*, volume 1884 of *LNCS*, Prague, Czech Republic, September 2000.

[Ora02a] Oracle Corporation. Oracle9i Lite: The Internet Platform For Mobile Computing http://otn.oracle.com/products/lite/, 2002.

[Ora02b] Oracle Corporation. Oracle9iAS Wireless http://otn.oracle.com/products/-iaswe/, 2002.

[OV99] T. Özsu et P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 2nd édition, 1999.

[PA02] A. Popovici et G. Alonso. Ad-hoc Transactions for Mobile Services. Dans *VLDB Workshop on Technologies for E-Services*, Hong-Kong, China, August 2002.

[PB95] E. Pitoura et B. Bhargava. Maintaining Consistency of Data in Mobile Distributed Environment. Dans *Int. Conf. on Distributed Computing Systems (ICDCS)*, Vancouver Canada, May 1995.

[PB99] E. Pitoura et B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11(6), 1999.

[PBVB01] P. Pucheral, L. Bouganim, P. Valduriez et C. Bobineau. Picodbms: Scaling down database techniques for the smartcard. *Very Large Databases (VLDB) Journal*, 10(2-3), 2001.

[PC99] E. Pitoura et P. K. Chrysanthis. Exploiting Versions for Handling Updates in Broadcast Disks. Dans *Int. Conf. on Very Large Databases (VLDB)*, Edinburgh, Scotland, UK, September 1999.

[Per00] PersonalJava Application Environment Specification v1.2 (Final). http://java.sun.com/products/personaljava/, November 2000.

[PKN88] C. Pu, G. Kaiser et N.Hutchinson. Split Transactions for Open-Ended Activities. Dans *Int. Conf. on Very Large Databases (VLDB)*, Los Angeles, USA, September 1988.

[PL91] C. Pu et A. Leff. Replica Control in Distributed Systems: An Asynchonous Approach. Dans *ACM SIGMOD Conference*, Denver, USA, May 1991.

[PL92] C. Pu et A. Leff. Autonomous Transaction Execution with Epsilon Serializability. Dans *Int. RIDE Workshop on Transaction and Query Processing*, Phoenix, USA, February 1992.

[Poi02] PointBase Java Databases. PointBase http://www.pointbase.com, 2002.

[PS98] E. Pitoura et G. Samaras. *Data Management for Mobile Computing.* Kluwer Academic Publishers, 1998.

[Pu88] C. Pu. Superdatabases for Composition of Heterogeneous Databases. Dans *Ont. Conf. on Data Engineering (ICDE)*, Los Angeles, USA, February 1988.

[RC96a] K. Ramamritham et P. K. Chrysanthis. A Taxonomy of Correctness Criteria in Database Applications. *Very Large Databases (VLDB) Journal*, 5(1), 1996.

[RC96b] K. Ramamritham et P. K. Chrysanthis. *Advances in Concurrency Control and Transaction Processing.* IEEE Computer Society Press, 1996.

[RN02] H. Ramampiaro et M. Nygård. CAGIS Trans: Providing Adaptable Transactional Support for Cooperative Work - Extended Version. *ITM: Information Technology and Management Journal, Kluwer Academic*, 2002.

[SA02] P. Serrano-Alvarado. Defining an Adaptable Mobile Transaction Service. Dans *Int. EDBT Workshop on Young Researchers Workshop*, number 2490 in LNCS, Prague, Czech Republic, March 2002.

[SA03] P. Serrano-Alvarado. Transacciones Adaptables en Ambientes Móviles. Dans *Consorcio Doctoral del Encuentro Internacional de Ciencias de la Computación (ENC)*, Tlaxcala, México, September 2003. In Spanish.

[SARA01a] P. Serrano-Alvarado, C. Roncancio et M. Adiba. Issues on Mobile Transactions for DBMS. Dans *Encuentro Nacional de Computación (ENC)*, Aguascalientes, México, September 2001.

[SARA01b] P. Serrano-Alvarado, C. Roncancio et M. Adiba. Mobile Transaction Supports for DBMS. Dans *Journées de Bases de Données Avancées (BDA)*, Maroc, October 2001.

[SARA01c] P. Serrano-Alvarado, C. L. Roncancio et M. Adiba. Analyzing Mobile Transactions Support for DBMS. Dans *Int. DEXA Workshop on Mobility in Databases and Distributed Systems (MDDS)*, Munich, Germany, September 2001.

[SARA02] P. Serrano-Alvarado, C. Roncancio et M. Adiba. TransMobi : Intergiciel pour la Gestion de Transactions Mobiles. Dans *Colloque sur l'informatique Mobile, Assises Nationales GdR I3*, Nancy, France, December 2002. In French.

[SARA04] P. Serrano-Alvarado, C. Roncancio et M. Adiba. A Survey of Mobile Transactions. *Kluwer Academic Publishers Distributed and Parallel Databases (DAPD)*, 2004. To appear.

[SARAL03] P. Serrano-Alvarado, C. Roncancio, M. Adiba et C. Labbé. Environment Awareness in Adaptable Mobile Transactions. Dans *Journées de Bases de Données Avancées (BDA)*, Lyon, France, October 2003.

[SARAL04] P. Serrano-Alvarado, C. Roncancio, M. Adiba et C. Labbé. Context Aware Mobile Transactions. Dans *Int. Conf. on Mobile Data Management (MDM)*, Berkeley, USA, January 2004. Poster and long abstract.

[SKK+90] M. Satynarayanan, J. Kistler, P. Kumar, E. Okasaki, H. Siegel et C. Steere. Coda: A Highly Available File System for Distributed Workstation Environment. *IEEE Transactions on Computers*, 39(4), 1990.

[SWL+02] CF. Sørensen, Alf Inge Wang, Hien Nam Le, H. Ramampiaro, M. Nygård et Reidar Conradi. The MOWAHS Characterisation Framework for Mobile Work. Dans *Symposium on Software Engineering, Databases and Applications*, Innsbruck, Austria, 2002.

[Syb02] Sybase Inc. Mobile and Wireless http://www.sybase.com/products/mobile-wireless/, 2002.

[VCFS00] N. Vidot, M. Cart, J. Ferrié et M. Suleiman. Copies convergence in a distributed real-time collaborative environment. Dans *Int. Conf. on Computer Supported Cooperative Work (CSCW)*, Philadelphia, USA, December 2000.

[Vin02] R. Vingralek. GnatDb: A Small-Footprint, Secure Database System. Dans *Int. Conf. on Very Large Databases (VLDB)*, Hong Kong, China, August 2002.

[VS00] G Vargas-Solar. *Service d'Evénements Flexible Pour l'Intégration d'Applications Bases de Données Réparties.* PhD thesis, Université Joseph Fourier, Granoble, France, December 2000. In French.

[WC95] G. D. Walborn et Panos K. Chrysanthis. Supporting Semantics-Based Transaction Processing in Mobile Database Applications. Dans *Symp. on Reliable Distributed Systems (SRDS)*, Bad Neuenahr, Germany, September 1995.

[WC97] G. D. Walborn et P. K. Chrysanthis. PRO-MOTION: Management of Mobile Transactions. Dans *ACM Symp. on Applied Computing*, San Jose, USA, March 1997.

[WC99] G. D. Walborn et P. K. Chrysanthis. Transaction Processing in PRO-MOTION. Dans *ACM Symp. on Applied Computing*, San Antonio, USA, February 1999.

[WXCJ98] O. Wolfson, B. Xu, S. Chamberlain et L. Jiang. Moving Objects Databases: Issues and Solutions. Dans *Int. Conf on Statistical and Scientific Database Management (SSDBM)*, Capri, Italy, July 1998.

[WYP92] K.L. Wu, P.S. Yu et C. Pu. Divergency Control for Epsilon Serializability. Dans *Int. Conf. on Data Engineering (ICDE)*, Phoenix, USA, February 1992.

[YZ94] L. H. Yeo et A. Zaslavsky. Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment. Dans *Int. Conf. on Distributed Computing Systems (ICDCS)*, Poznan, Poland, June 1994.

[ZNB01] A. Zhang, M. H. Nodine et B. K. Bhargava. Global Scheduling for Flexible Transactions in Heterogeneous Distributed Database Systems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(3), 2001.

[ZNBB94] Aidong Zhang, Marian Nodine, Bharat Bhargava et Omran Bukhres. Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems. Dans *ACM SIGMOD Conference*, Minneapolis, USA, May 1994.

ANNEXE A

Rappel de la logique du premier ordre

L'objectif de cet annexe est d'aider à la compréhension de la spécification formelle du modèle AMT faite dans le chapitre 4. Nous présentons d'abord un petit rappel des lois de la logique du premier ordre (section A.1). Ensuite, nous rappelons quelques règles de déduction naturelle (section A.2). Enfin (section A.3) nous décrivons les déroulement des déductions faites dans la section 4.2.

A.1 Lois logiques

- Double négation : $\neg(\neg A) \Leftrightarrow A$
- Transitivité : $((A \Rightarrow B) \wedge (B \Rightarrow C)) \Rightarrow (A \Rightarrow C)$
- Commutativité : $(A \wedge B) \Leftrightarrow (B \wedge A)$
  $(A \vee B) \Leftrightarrow (B \vee A)$
- Associativité : $((A \wedge B) \wedge C) \Leftrightarrow (A \wedge (B \wedge C))$
  $((A \vee B) \vee C) \Leftrightarrow (A \vee (B \vee C))$
- Distributivité : $(A \wedge (B \vee C)) \Leftrightarrow ((A \wedge B) \vee (A \wedge C))$
  $(A \vee (B \wedge C)) \Leftrightarrow ((A \vee B) \wedge (A \vee C))$
- Lois de Morgan : $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$
  $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$
- Contraposition : $(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$
- L'implication : $(A \Rightarrow B) \Leftrightarrow (\neg A \vee B)$

- La négation d'une implication : $\neg(A \Rightarrow B) \Leftrightarrow (A \wedge \neg B)$
- Double implication : $(A \Leftrightarrow B) \Leftrightarrow ((A \Rightarrow B) \wedge (B \Rightarrow A))$

A.2 Règles de déduction naturelle

Les règles de déduction naturelle peuvent être d'introduction ou de réduction.

- A B on en déduit $A \wedge B$
- $(A \Rightarrow B, A)$ on en déduit B (Modus ponens)
- $(A \Rightarrow B, \neg B)$ on en déduit $\neg A$ (Modus tollens)
- $(A \Rightarrow B)$ $(B \Rightarrow A)$ on en déduit $(A \Leftrightarrow B)$

A.3 Quelques déductions faites

Failure atomicity, définition B.5

Transactions atomiques Axiome 9 (définition B.8)
$B = (commit_t[p_t[ob]] \in H)$
$A = (commit_t \in H)$
$B \Rightarrow A$

Transactions atomiques Axiome 10 (définition B.8)
$C = (p_t[ob] \in H)$
$A \Rightarrow (C \Rightarrow B)$

Preuve
$(B \Rightarrow A)$ $(A \Rightarrow (C \Rightarrow B))$
$(B \Rightarrow A) \wedge (A \Rightarrow (C \Rightarrow B))$
$B \Rightarrow (C \Rightarrow B)$ Par transitivité
Donc, dans la définition B.5 (1) **Failure atomicity** $B \Rightarrow (C \Rightarrow B)$
De façon similaire, nous pouvons déduire des Axiomes 11 et 12 (de la définition B.8) la définition B.5 (2).

L'atomicité sémantique, définition 4.4 (2)

$A = (abort_{AE_k})$
$B = (commit_{t_{ki}} \rightarrow abort_{AE_k})$

$C = (abort_{t_{ki}})$
$D = (commit_{t_{ki}} \rightarrow commit_{tc_{ki}})$

$A \Rightarrow (\neg B \Rightarrow C) \wedge (B \Rightarrow D)$

$(\neg B \Rightarrow C) \Leftrightarrow (\neg C \Rightarrow B)$
$\neg C \rightarrow B, \neg B / C$ modus tollens

$B \Rightarrow D, B / D$ modus ponens $A \Rightarrow (C \wedge D)$

La validation d'une T_{AMT}, Lemme 4.7(1)

$A = (abort_{AE_k} \in H)$
$B = (abort_{T_{AMT}} \in H)$

$A \Rightarrow B$
$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$ Par contraposition

La validation d'une AE, Lemme 4.3

$A = (commit_{AE_k} \in H)$
$B = (commit_{t_{ki}} \in H)$

$A \Rightarrow B$
$(A \Rightarrow B) \Leftrightarrow (\neg B \Rightarrow \neg A)$ Par contraposition

La validation complète d'une T_{AMT}, Lemme 4.8

Du Lemme 4.7(3)
$A = (commit_{T_{AMT}} \in H)$
$B = (commit_{AE_k} \in H)$

$A \Rightarrow B$

Du Lemme 4.3
$C = (commit_{t_{ki}} \in H)$
$B \Rightarrow C$
Donc dans le Lemme 4.8
$A \Rightarrow B \quad B \Rightarrow C$
$(A \Rightarrow B) \wedge (B \Rightarrow C)$
$A \Rightarrow C$ Par transitivité.
Cette même analyse est faite pour l'annulation complète d'une T_{AMT}, Lemme 4.10

L'annulation d'une AE, Lemme 4.4 Cas 2(3)

$A = (abort_{AE_k} \in H)$
$B = (commit_{tc_{ki}} \in H)$
$C = (commit_{t_{ki}} \rightarrow commit_{tc_{ki}})$

$(A \Rightarrow B) \wedge (B \Rightarrow C)$
$A \Rightarrow C$ Par transitivité

L'annulation d'une AE, Lemme 4.4 Cas 3

$A = (abort_{AE_k} \in H)$
$B = (abort_{ti} \in H)$
$C = (commit_{t_{ki}} \rightarrow commit_{tc_{ki}})$

$(A \Rightarrow B) \wedge (A \Rightarrow C)$
$(\neg A \vee B) \wedge (\neg A \vee C)$ loi de Morgan$(\neg A \vee (B \wedge C))$ associativité $A \Rightarrow (B \wedge C)$

ANNEXE B

Le formalisme ACTA

ACTA [CR90, Chr91, CR91, CR92, CR94] est un cadre formel et extensible conçu pour la spécification de transactions. Il est basé sur la logique du premier ordre.[1] ACTA permet de spécifier et de raisonner sur les interactions entre les transactions. Il considère différents types d'interaction entre les transactions ainsi que les effets des transactions sur les objets. ACTA organise l'expression des effets des transactions (sur d'autres transactions et sur des objets) en cinq dimensions : l'historique, les dépendances entre transactions, la vue d'une transaction, l'ensemble des conflits d'une transactions et la délégation (voir figure B.1).

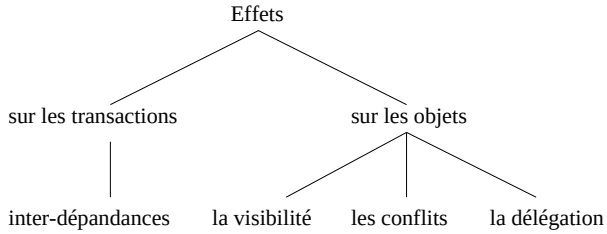

FIG. B.1 – *Les dimensions du canevas ACTA*

L'interdépendance des transactions permet de définir la structure du modèle transactionnel ainsi que les aspects liés à l'*atomicité*. La visibilité et les conflits sur les objets concernent les caractéristiques liées à l'*isolation* et à la correction des données manipulées (*sérialisabilité*). La propriété de *cohérence* ne peut pas être exprimée par le formalisme ACTA. Cette restriction n'a pas de grandes répercutions sur la spécification correcte des modèles transactionnels car la vérification des contraintes d'intégrité n'intervient pas dans la définition des modèles.

1. Dans l'annexe A, nous présentons un rappel de quelques lois de la logique du premier ordre.

Cette section introduit le formalisme ACTA. Nous présentons les notions de base nécessaires à la définition du modèle AMT (cf. chapitre 4). Pour une présentation plus complete voir [CR90, Chr91, CR91, CR92, CR94], ainsi que [Adi00]. La section B.5 présente quelques spécifications indispensables comme la sérialisabilité ou l'atomicité.

B.1 Les objets, les événements

Les transactions accèdent aux objets et les manipulent en demandant des opérations (par exemple, *lire, écrire, incrémenter*). Ces opérations sont considérées comme des événements atomiques. $p_t[ob]$ dénote l'événement qui correspond à la demande de l'opération p sur l'objet ob par la transaction t. OE_t dénote l'ensemble des événements qui peuvent être demandées par t, ainsi, $p_t[ob] \in OE_t$.

L'*état* d'un objet est représenté par son contenu. Les opérations produisent toujours un résultat. Le résultat d'une opération sur un objet dépend de l'état de l'objet. Pour un objet dans un état donné s, $return(s, p)$ exprime le résultat retourné par une opération p, et $état(s, p)$ dénote l'état de l'objet après l'exécution de p.

Les effets des opérations sur les objets ne sont pas permanents pendant l'exécution. Les opérations doivent être *validées* (*commit*) ou *annulées* (*abort*) de manière explicite. Si une opération n'est pas validée ou annulée, alors elle est en cours d'exécution (*in progress*). Une opération est validée (ou annulée) lorsque la transaction qui l'a demandée valide (ou annule). Ainsi, les effets d'une opération p demandée par une transaction t sur un objet ob sont rendus permanents sur la base de données lorsque $p_t[ob]$ est validée.

En dehors de la demande des opérations sur les objets, les transactions demandent des primitives de gestion – début (*begin*), validation, annulation. La définition des primitives et leur sémantique dépendent du modèle de transactions. De ce fait, la demande d'une primitive de transaction est un événement significatif. ACTA ne propose pas un ensemble particulier d'événements significatifs mais il fournit un moyen pour le spécifier.

- ES_t dénote l'ensemble des Événements Significatifs pour la transaction t.
- EI_t dénote les Événements d'Initiation qui peuvent être demandés pour initier l'exécution de la transaction t. $EI_t \subset ES_t$.
- ET_t dénote les Événements de Terminaison qui peuvent être demandés pour terminer l'exécution de la transaction t. $ET_t \subset ES_t$.
- Les événements correspondant aux opérations sur les objets sont différents des événements de gestion des transactions $OE_t \neq ES_t$.

B.2 Les historiques et les conditions sur l'occurrence des événements

La notion d'*historique* est fondamentale dans ACTA. Un historique représente l'exécution concurrente d'un ensemble de transactions. ACTA capture les effets des transactions

sur d'autres transactions, ainsi que les effets sur les objets, au travers de *contraintes* sur les historiques. Ceci conduit à la définition de modèles de transactions en terme d'un ensemble d'*axiomes*. Ces axiomes sont des assertions invariables concernant les historiques ou des *pré-conditions/post-conditions* des opérations ou encore des primitives de gestion des transactions. La correction des différents modèles de transactions peut également être exprimée en terme de propriétés des historiques générés.

Définition B.1 *: **L'exécution d'une transaction***

L'exécution d'une transaction t est un ordre partiel sur l'ensemble d'événements E_t avec une relation d'ordre $<_t$ où:

1. $E_t \subseteq (EO_t \cup ES_t)$ *; et*

2. $<_t$ *dénote l'ordre temporel dans lequel les événements demandés par t arrivent.*

Autrement dit, E_t contient les événements sur les objets qui peuvent être demandés par t et les événements significatifs liés à t.

Définition B.2 *: **Historique H** Un historique H de l'exécution concurrente d'un ensemble de transactions T, contient tous les événements associés à T et indique l'ordre (partiel) de l'occurrence de ces événements. H_{ct} dénote l'historique des événements produits avant un instant donné (historique courant).*

La place d'apparition d'un événement dans l'historique peut être affecté de trois manières différentes : (1) un événement e peut être contraint de se produire *après* un autre événement e' ; (2) un événement e peut apparaître *seulement si* une condition c est vraie ; (3) une condition c peut *demander* l'occurrence d'un événement e.

- $e \rightarrow e'$: le prédicat est vrai si l'événement e précède l'événement e' dans H.
- $(e \in H) \Rightarrow condition_H$: si e appartient à H, alors la condition sur H est vrai. Autrement dit, $condition_H$ est nécessaire pour que e soit dans H.
- $condition_H \Rightarrow (e \in H)$: si la condition sur H est vrai, alors e appartient à H. Autrement dit, $condition_H$ est suffisante pour que e soit dans H.

Par ailleurs, la projection d'un historique H selon un critère donné x est exprimée par $projection(H, x)$. Par exemple, $projection(H, t)$ est la projection de l'historique H sur une transaction t. Cette projection notée H^t, donne l'ordre des événements liés à t. Nous pouvons également considérer $projection(H, ob)$, qui est la projection de l'historique H sur un objet spécifique ob. Cette projection notée $H^{(ob)}$ donne l'historique de la demande des opérations sur l'objet ob.

B.3 Dépendances des transactions

ACTA utilise des *dépendances* comme un moyen de spécifier et de raisonner sur le comportement des transactions concurrentes. Les dépendances sont exprimées en terme d'événements significatifs associés aux transactions. Ainsi, l'ensemble de dépendances inter-transactions, qui se produisent pendant l'exécution concurrente des transactions est noté $DepSet$. Cet ensemble est relatif à l'historique H. $DepSet_{ct}$ dénote l'ensemble de dépendances jusqu'à un instant donné (ct), où $DepSet_{ct}$ concerne H_{ct}.

Types de dépendances

La liste de dépendances présentée ici n'est pas exhaustive. Grâce au caractère extensible d'ACTA, d'autres dépendances peuvent être définies.

Soit t_i et t_j deux transactions et H un historique fini qui contient tous les événements appartenant à t_i et t_j.

Dépendance de validation (Commit Dependency) $(t_j\ \mathcal{CD}\ t_i)$: si les deux transactions t_i et t_j valident, la validation de t_i doit précéder celle de t_j :
$(commit_{t_j} \in H) \Rightarrow ((commit_{t_i} \in H) \Rightarrow (commit_{t_i} \rightarrow commit_{t_j}))$.

Dépendance de validation forte (Strong-Commit Dependency) $(t_j\ \mathcal{SCD}\ t_i)$: si t_i valide, alors t_j valide aussi :
$(commit_{t_i} \in H) \Rightarrow (commit_{t_j} \in H)$.

Dépendance d'annulation (Abort Dependency) $(t_j\ \mathcal{AD}\ t_i)$: si t_i annule, alors t_j aussi :
$(abort_{t_i} \in H) \Rightarrow (abort_{t_j} \in H)$.

Dépendance d'annulation faible (Weak-Abort Dependency) $(t_j\ \mathcal{WD}\ t_i)$: si t_i annule et t_j n'a pas encore validé, t_j doit annuler aussi. En d'autres termes, si t_j valide et t_i annule, la validation de t_j doit précéder l'annulation de t_i dans l'historique :
$(abort_{t_i} \in H) \Rightarrow (\neg(commit_{t_j} \rightarrow abort_{t_i}) \Rightarrow (abort_{t_j} \in H))$.

Dépendance de terminaison (Termination Dependency) $(t_j\ \mathcal{TD}\ t_i)$: t_j ne peut pas valider ou annuler tant que t_i n'a pas validé ou annulé :
$(\epsilon' \in H) \Rightarrow (\epsilon \rightarrow \epsilon')$ où $\epsilon \in \{commit_{t_i}, abort_{t_i}\}$, et $\epsilon' \in \{commit_{t_j}, abort_{t_j}\}$.

Dépendance d'exclusion (Exclusion Dependency) $(t_j\ \mathcal{ED}\ t_i)$: si t_i valide et si t_j a démarré son exécution, alors t_j annule (tous deux, t_i et t_j, ne peuvent pas valider) :
$(commit_{t_i} \in H) \Rightarrow ((begin_{t_j} \in H) \Rightarrow (abort_{t_j} \in H))$.

Dépendance de compensation (Force-Commit-on-Abort Dependency) $(t_j\ \mathcal{CMD}\ t_i)$: si t_i annule, t_j doit valider :
$(abort_{t_i} \in H) \Rightarrow (commit_{t_j} \in H)$.

Dépendance de début (Begin Dependency) $(t_j\ \mathcal{BD}\ t_i)$: t_j ne peut pas démarrer si t_i n'a pas démarré :
$(begin_{t_j} \in H) \Rightarrow (begin_{t_i} \rightarrow begin_{t_j})$.

Dépendance séquentielle (Serial Dependency) $(t_j\ \mathcal{SD}\ t_i)$: t_j ne peut pas démarrer tant que t_i n'a pas validé ou annulé :
$(begin_{t_j} \in H) \Rightarrow (\epsilon \rightarrow begin_{t_j})$ où $\epsilon \in \{commit_{t_i}, abort_{t_i}\}$.

Dépendance début sur validation (Begin-on-Commit Dependency) $(t_j\ \mathcal{BCD}\ t_i)$: t_j ne peut pas démarrer jusqu'à ce que t_i valide :
$(begin_{t_j} \in H) \Rightarrow (commit_{t_i} \rightarrow begin_{t_j})$.

Dépendance début sur annulation (Begin-on-Abort Dependency) $(t_j\ \mathcal{BAD}\ t_i)$: t_j ne peut pas démarrer jusqu'à ce que t_i annule :
$(begin_{t_j} \in H) \Rightarrow (abort_{t_i} \rightarrow begin_{t_j})$.

Dépendance début sur validation faible (Weak-begin-on-Commit Dependency) $(t_j\ \mathcal{WCD}\ t_i)$: si t_i valide, t_j peut démarrer après que t_i valide :
$(begin_{t_j} \in H) \Rightarrow ((commit_{t_i} \in H) \Rightarrow (commit_{t_i} \rightarrow begin_{t_j}))$.

B.3.1 Sources de dépendances

Les dépendances entre les transactions peuvent provenir soit directement des propriétés structurelles des transactions – *dépendances de structure* – soit indirectement par le fait que les transactions manipulent les mêmes objets – *dépendances de comportement.*

Dépendances de structure

La structure d'une transaction étendue définit ses transactions composantes et les relations entre elles. En effet, les dépendances peuvent spécifier des liens dans la structure. Par exemple, dans une *Sagas* (cf. section 2.3.3), la relation avec ses composantes est établie au démarrage de la transaction composante. Ceci peut être exprimé par la dépendance d'annulation faible de la transaction composante t_i vis-à-vis de la Saga S, $(t_i\ \mathcal{WD}\ S)$, et par la dépendance d'annulation de la Saga vis-à-vis de la transaction composante $(S\ \mathcal{AD}\ t_i)$. Comme la relation s'établie au démarrage de la transaction composante t_i, cette dépendance s'exprime par une *post* condition :

$$post(begin_{t_i}) \Rightarrow (((t_i\ \mathcal{WD}\ S) \in DepSet_{ct}) \wedge ((S\ \mathcal{AD}\ t_i) \in DepSet_{ct}))$$

La dépendance d'annulation de la Saga sur ses transactions composantes assure l'annulation de la Saga lorsqu'une transaction composante annule. La dépendance d'annulation faible garantit l'annulation d'une transaction composante non-validée si la Saga annule. Il faut remarquer que cette dépendance n'interdit pas à la transaction composante de valider et de rendre ses effets visibles.

Dépendances de comportement

Les dépendances ($\mathcal{D}$) formées par l'interaction de transactions sur un objet partagé sont déterminées par les propriétés de synchronisation sur cet objet. Deux opérations sont en conflit si l'ordre de leur exécution est important. Par exemple, une relation de conflit ($\mathcal{D}$) peut apparaître de la manière suivante :

$(p_{t_i}[ob] \rightarrow q_{t_j}[ob]) \Rightarrow (t_j \ \mathcal{D} \ t_i))$

Ceci veut dire que si la transaction t_i demande une opération p et qu'ultérieurement une transaction t_j demande une opération q sur le même objet, alors t_j génère une dépendance de type $\mathcal{D}$ sur t_i.

B.4 Effets des transactions sur les objets

La correction de l'exécution concurrente de transactions dépend des dépendances entre elles et de la manière dont elles accèdent aux objets. Plus concrètement, la correction dépend de l'effet des événements significatifs associés aux transactions ainsi que des opérations demandées. La section B.3.1 a abordé le premier point. Cette section aborde le second. Tout d'abord, nous introduisons les effets des opérations sur les transactions et les inter-relations induites par ces effets. Ensuite, nous expliquons la visibilité des effets des opérations d'une transaction sur une autre transaction.

Conflits entre les opérations

D'une façon générale, deux opérations p et q sont en conflit si elles accèdent au même objet et au moins une d'entre elles est une écriture ($conflit(H^{(ob)}, p, q)$). C'est-à-dire, deux opérations sont en conflit si leurs effets sur l'état d'un objet ou leurs valeurs de retour ne sont pas indépendants de leur ordre d'exécution :

$return - value - dependent(H^{(ob)}, p, q)$ est vrai si ($conflit(H^{(ob)}, p, q)$) est vrai et $return(H^{(ob)} \circ p, q) \neq return(H^{(ob)} q)$.

Dans ACTA le contrôle de la concurrence sur un objet est exprimé en terme de *relations de conflit* :

$(p_{t_i}[ob] \rightarrow q_{t_j}[ob]) \Rightarrow condition_H$

Où $condition_H$ est une dépendance de relation qui inclut les transactions t_i et t_j qui demandent les opérations conflictuelles p et q sur l'objet ob.

La $condition_H$ dans une relation de conflit peut inclure d'autres événements significatifs définis par les différents modèles transactionnels. En outre, la généralité qui porte la relation de conflit montrée ici, permet la spécification de différents types de contrôle de concurrence.

Visibilité d'une transaction

La *visibilité* d'une transaction est sa capacité de voir les effets d'autres transactions pendant son exécution. Dans ACTA le contrôle sur la visibilité des objets est défini par deux entités, *Visibilité* et *ConflictSet*. *Visibilité* est l'ensemble des objets visibles à un instant

donné. Ainsi, lorsque la visibilité d'une transaction porte sur l'historique courant des objets de la base de données, sans aucune restriction, la définition est :

$Visibilité_t = H_{ct}$

La restriction de la visibilité peut être exprimée par la projection d'un prédicat sur l'historique courant :

$Visibilité_t = projection(H_{ct}, prédicat(t, H_{ct}, DepSet_{ct}))$.

Le prédicat dépend de t, des événements dans H_{ct} et des dépendances inter-transactions $DepSet_{ct}$.

Ensemble des conflits d'une transaction

L'ensemble des conflits d'une transaction t, dénoté par $ConflictSet_t$, contient toutes les opérations en cours avec lesquelles des conflits doivent être déterminés. La composition d'un $ConflictSet_t$ est aussi définie par des prédicats qui peuvent inclure des événements demandés par t, par d'autres transactions t_i, par des événements de H_{ct}, ainsi que par des dépendances dans $DepSet_t$:

$ConflictSet_t = \{p_{t_i}[ob] \mid predicat(t, t_i, H_{ct}, DepSet_{ct})\}$

Cela dit, l'ensemble des conflits d'une transaction t est composé par toutes les opérations en cours faites par d'autres transactions :

$ConflictSet_t = \{p_{t'}[ob] \mid t' \neq t, Inprogress(p_{t'}[ob])\}$.

B.5 Quelques spécifications en utilisant ACTA

Dans cette section, nous montrons la définition de sérialisabilité, de *failure atomicity*, de *setwise failure atomicity* et des transactions atomiques. Ces concepts sont indispensables pour faciliter l'explication et compréhension de la spécification de l'AMT, que nous introduisons au chapitre 4.

B.5.1 Notions de correction

Sérialisabilité

On définit la sérialisabilité de la manière suivante grâce au formalisme ACTA:
Soit T un ensemble de transactions.
Soit $\mathcal{C}$ une relation binaire sur les transactions de T.
Soit T_{commit} le sous-ensemble de T qui contient les transactions qui ont validé.
Soit H_{commit} l'historique des événements liés aux transactions dans T_{commit}.

Définition B.3 *: Sérialisabilité*

$\forall t_i, t_j \in T_{commit}, t_i \neq t_j$

$(t_i \; \mathcal{C} \; t_j)$ *si* $\exists ob \; \exists p, q \; (conflit(p_{t_i}[ob], q_{t_j}[ob]) \wedge (p_{t_i}[ob] \rightarrow q_{t_j}[ob]))$

Soit $\mathcal{C}^*$ *la fermeture transitive de* $\mathcal{C}$ *:*

$(t_i \; \mathcal{C}^* \; t_k)$ *si* $[(t_i \; \mathcal{C} \; t_k) \vee \exists t_j (t_i \; \mathcal{C} \; t_j \wedge t_j \; \mathcal{C}^* \; t_k)]$.

H_{commit} *est (conflit)* ***sérialisable*** *si* $\forall t \in T_{commit} \neg(t \; \mathcal{C}^* \; t)$.

Autrement dit, deux transactions ont une relation d'ordre ($\mathcal{C}$), lors qu'elles sont en conflit et lorsque la fermeture transitive est acyclique. Cette définition de sérialisabilité est en réalité celle de "sérialisabilité conflictuelle" où l'ordre d'exécution des transactions qui ne sont pas en conflit n'affecte pas la correction des objets.

Les objets atomiques

La sérialisabilité porte sur un ensemble de transactions validées. Cependant, il est aussi important d'assurer la sérialisabilité sur les transactions en cours d'exécution. En effet, il faut assurer que l'annulation d'une opération entraîne l'annulation des opérations pour lesquelles une dépendance de valeur existe. La définition qui suit montre que pour qu'un objet ait un comportement *correct*, il doit assurer que lorsqu'une opération (p_{t_i}) annule, toutes les opérations (q_{t_j}) dont leur valeurs dépendent de (p_{t_i}), doivent aussi annuler. Un objet a aussi un comportement *sérialisable* si la relation des transactions validées qui lui ont accédé est acyclique où $\mathcal{C}_{ob}$ considère seulement les relations concernant ob ($\mathcal{C}_{ob} \subseteq \mathcal{C}$).

Définition B.4 *: Correction d'un objet*

Un objet ob a un comportement **correct** *si :*

$\forall t_i, t_j, t_i \neq t_j, \; \forall p, q$

$(return\text{-}value\text{-}dependent(p, q) \wedge (p_{ti}[ob] \rightarrow q_{tj}[ob]) \wedge$

$\neg((commit[p_{t_i}[ob]] \rightarrow q_{t_j}[ob]) \vee (abort[p_{t_i}[ob]] \rightarrow q_{t_j}[ob])) \Rightarrow$

$((abort[p_{t_i}[ob]] \in H^{(ob)}) \Rightarrow (abort[q_{t_j}[ob]] \in H^{(ob)}))$.

$\forall t_i, t_j \in T_{commit}, t_i \neq t_j$

$(t_i \; \mathcal{C}_{ob} \; t_j)$ *si* $\exists p, q \; (conflit(p_{t_i}[ob], q_{t_j}[ob]) \wedge (p_{t_i}[ob] \rightarrow q_{tj}[ob]))$

Un objet ob a un comportement **sérialisable** *si :*

$\forall t \in T_{commit} \neg(t \; \mathcal{C}^*_{ob} \; t)$.

Un objet ob est **atomique** *s'il a un comportement* **correct** *et* **sérialisable**.

B.5.2 Différents types d'atomicité

Failure atomicity

L'atomicité, ou plus particulièrement la *failure atomicity* comme indiqué dans [Chr91], implique la validation de toutes les opérations d'une transaction ou d'aucune d'entre elles ("tout ou rien"). Dans la définition qui suit, le premier point exprime la clause "tout" tandis que le second exprime la clause "rien".

Définition B.5 *: **Failure atomicity***

Une transaction t est failure atomic si :

1. $\exists ob\, \exists p\, (commit[p_t[ob]] \in H) \Rightarrow$

   $\forall ob'\, \forall q\, ((q_t[ob'] \in H) \Rightarrow (commit[q_t[ob'] \in H))$

2. $\exists ob\, \exists p\, (abort[p_t[ob]] \in H) \Rightarrow$

   $\forall ob'\, \forall q\, ((q_t[ob'] \in H) \Rightarrow (abort[p_t[ob']] \in H))$

Voir Annexe A.

Setwise failure atomicity

Setwise failure atomicity est une généralisation de *failure atomicity* pour des transactions composées de plusieurs transactions. De façon similaire à *failure atomicity*, la définition qui suit indique dans sa première condition que si une opération demandée par une transaction t_i appartenant à T est validée, alors, toutes les opérations demandées par les transactions dans T sont validées. La seconde condition indique que si une opération demandée par t_i qui appartient à T est annulée, alors, toutes les opérations demandées par les transactions de T sont annulées.

Définition B.6 *: **Setwise failure atomicity***

Un ensemble de transactions T est setwise failure atomic si :

1. $\exists t_i \in T\, \exists ob\, \exists p\, (commit_{t_i}[p_{t_i}[ob]] \in H) \Rightarrow$

   $\forall t_j \in T\, \forall ob'\, \forall q\, ((q_{t_j}[ob'] \in H) \Rightarrow (commit_{t_j}[q_{t_j}[ob']] \in H))$,

2. $\exists t_i \in T\, \exists ob\, \exists p\, (abort_{t_i}[p_{t_i}[ob]] \in H) \Rightarrow$

   $\forall t_j \in T\, \forall ob'\, \forall q\, ((q_{t_j}[ob'] \in H) \Rightarrow (abort_{t_j}[q_{t_j}[ob']] \in H))$

Failure atomicity ainsi que *setwise failure atomicity* assurent l'**atomicité** d'une ou de plusieurs transactions.

B.5.3 Transactions atomiques

Cette section introduit, en premier lieu, les axiomes fondamentaux des transactions (définition B.7), puis les axiomes des *transactions atomiques* (définition B.8).

Les axiomes fondamentaux des transactions

Chaque modèle de transaction définit un ensemble d'événements significatifs que les transactions de ce modèle peuvent demander en plus des opérations sur les objets. En effet, une transaction t est toujours associée à un ensemble d'événements d'initiation (EI_t) et à un ensemble d'événements de terminaison (TE_t). Ici, nous montrons un ensemble d'axiomes fondamentaux qui sont applicables à tout modèle de transactions. Ces axiomes spécifient la relation entre les événements significatifs de type identique ou différent, ainsi qu'entre les événements significatifs et les opérations sur les objets.

Définition B.7 *: Les axiomes fondamentaux des transactions*

I $\forall \alpha \in EI_t \ (\alpha \in H^t) \Rightarrow \nexists \beta \in EI_t \ (\alpha \rightarrow \beta)$
Une transactions ne peut pas être initiée par deux événements différents.

II $\forall \delta \in ET_t \ \exists \alpha \in EI_t \ (\delta \in H^t) \Rightarrow (\alpha \rightarrow \delta)$
Si une transaction est terminée, elle a dû auparavant être initiée.

III $\forall \gamma \in ET_t \ (\gamma \in H^t) \Rightarrow \nexists \delta \in ET_t \ (\gamma \rightarrow \delta)$
Une transaction ne peut pas être terminée par deux événements différents.

IV $\forall ob \ \forall p \ (p_t[ob] \in H) \Rightarrow ((\exists \alpha \in EI_t(\alpha \rightarrow p_t[ob])) \wedge (\exists \gamma \in ET_t(p_t[ob] \rightarrow \gamma)))$
Seules les transactions en cours peuvent demander des opérations sur les objets. Cet axiome énonce simultanément qu'une transaction doit toujours être initiée et terminée.

Les transactions atomiques

Nous considérons les transactions atomiques comme des transactions *plates* avec les propriétés AID. Elles sont *sérialisables* et *atomiques*. Autrement dit, elles sont exécutées sans interférence, comme si elles respectaient un ordre séquentiel. De plus, soit toutes les opérations d'une transaction sont exécutées soit aucune d'entre elles. Dans la définition qui suit, les axiomes 8, 9 et 10, définissent la sémantique de l'événement *commit* en terme de l'opération *commit* définie sur les objets. Les axiomes 11 et 12 définissent la sémantique de l'événement *abort* en terme de l'opération *abort* définie sur les objets.

Définition B.8 *: Définition axiomatique des transactions atomiques*

1. $ES_t = \{\text{begin, commit, abort}\}$

2. $EI_t = \{\text{begin}\}$
3. $ET_t = \{\text{commit, abort}\}$
4. t satisfait les axiomes fondamentaux I à IV
5. $Visibilité_t = H_{ct}$

   Une transaction perçoit l'état courant de tous les objets dans la base de données.
6. $ConflitSet_t = \{p_{t'}[ob] \mid t' \neq t, Inprogress(p_{t'}[ob])\}$

   Les effets des conflits concernent toutes les opérations en cours réalisées par d'autres transactions.
7. $\forall ob\, \exists p\ (p_t[ob] \in H) \Rightarrow (ob$ est atomique)
   Tous les objets sur lesquels une transaction demande une opération sont atomiques (ob est atomique s'il est *correct* et *sérialisable*).
8. $(commit_t \in H) \Rightarrow \neg(t\ \mathcal{C}^*\ t)$
   Une transaction atomique peut valider seulement si elle ne fait pas partie d'un cycle dans une relation $\mathcal{C}$ (cf. définition B.3).
9. $\exists ob\, \exists p\ (commit_t[p_t[ob]] \in H) \Rightarrow (commit_t \in H)$
   Si une opération sur un objet valide, la transaction qui en fait la demande doit valider.
10. $(commit_t \in H) \Rightarrow \forall ob\, \forall p\ ((p_t[ob] \in H) \Rightarrow (commit_t[p_t[ob]] \in H))$
    Si une transaction valide, toutes les opérations demandées par la transaction valident aussi.
11. $\exists ob\, \exists p\ (abort_t[p_t[ob]] \in H) \Rightarrow (abort_t \in H)$
    Si une opération sur un objet annule, la transaction qui en fait la demande doit annuler.
12. $(abort_t \in H) \Rightarrow \forall ob\, \forall p\ ((p_t[ob] \in H) \Rightarrow (abort_t[p_t[ob]] \in H))$
    Si une transaction annule, toutes les opérations demandées par la transaction sont aussi annulées.

Théorème B.1 *: Propriétés des transactions atomiques :*

1. Si t est une transaction atomique, t est **failure atomic**,
2. Un ensemble T de transactions atomiques validées est **sérialisable**.

Preuve La preuve est donnée par [Chr91]. Nous tenons juste à souligner que la condition 2 est dérivée de la relation $\mathcal{C}$ établie entre les transactions avec des opérations en conflit (Axiome 7) ainsi que du critère de sérialisabilité établie dans l'Axiome 8. C'est-à-dire, les transactions respectent un ordre sérialisable si les opérations en conflit sont sérialisées et s'il n'existe pas de cycle entre les transactions.

Nous considérons qu'une transaction atomique satisfait les propriétés AID. La propriété de *durabilité* est assurée lors de la validation des opérations sur les objets (cf. section B.1).

B.5.4 Transactions réparties et emboîtées

Le deux théorèmes qui suivent présentent les propriétés des transactions réparties et emboîtées fermées (cf. section 2.3.1), nous en ferons référence dans le chapitre 4.

Théorème B.2 *: Propriétés des transactions réparties :*

1. Si t_d est une transactions répartie alors t_d est *setwise failure atomic*,
2. Un ensemble de transactions réparties validées S est sérialisable.

Preuve La preuve est donnée par [Chr91].

Théorème B.3 *: Propriétés des transactions emboîtées :*

1. Un ensemble de transactions emboîtées T est sérialisable,
2. Les opérations sont validées seulement par la transaction racine :
   $\forall ob\ \forall p\ \forall t\ ((p_t[ob] \in H) \wedge (commit_{t_r}[p_t[ob]] \in H)) \Rightarrow (t_r$ est une transaction racine),
3. Si une transaction t_0 annule, toutes les opérations réalisées par t_0 et ses descendantes annulent :
   $(abort_{t_0} \in H) \Rightarrow$
   $\forall t,\ (t = t_0\ \vee t \in Descendants(t_0))\ \forall ob\ \forall p\ ((p_t[ob] \in H) \Rightarrow (abort[p_t[ob]] \in H))$

Preuve La preuve est donnée par [Chr91].

Index

Transactions Adaptables pour les Environnements Mobiles

Résumé

Dans cette thèse, nous nous intéressons aux transactions dans les environnements mobiles. La problématique visée est l'influence du contexte sur l'exécution des transactions. En effet, les environnements mobiles se caractérisent par une grande variabilité au niveau réseau sans fil et par des unités mobiles à ressources limitées. Ces caractéristiques affectent la gestion de données, en particulier, elles entraînent un nombre important de défaillances transactionnelles et des coûts d'exécution imprévus.

Nous proposons un modèle de Transactions Mobiles Adaptables (AMT) permettant de définir des transactions avec plusieurs *alternatives d'exécution*. Ce modèle offre la possibilité d'adapter les transactions au contexte et de maîtriser le coût de leur exécution selon des critères de qualité acceptables pour l'application. Le modèle AMT a été spécifié avec le formalisme ACTA. Nous proposons une étude analytique qui montre que (1) les transactions de type AMT augmentent la probabilité de validation, et que (2) les coûts deviennent prévisibles grâce à la possibilité de choisir le type d'exécution en fonction du contexte. Pour valider notre approche, nous avons conçu l'intergiciel TransMobi qui gère la perception de l'environnement et qui implante le modèle AMT à l'aide de protocoles appropriés. En particulier, nous proposons le protocole de validation CO2PC qui assure l'atomicité sémantique. Nous avons développé un prototype de TransMobi en PersonalJava qui utilise des bases de données relationnelles, un réseau WLAN et des assistants personnels (PDA).

Mots clefs : Transactions, environnement mobile, atomicité, cohérence, isolation, durabilité, perception de l'environnement, adaptation.

Adaptable Transactions for Mobile Environments

Abstract

In this dissertation, we are interested in transactions in mobile environments. The focussed problem is the context influence on transaction execution. Mobile environments are characterized by highly variable wireless networks and by limited mobile host resources. Such characteristics affect data management and in particular, they generate high rates of transactional failures and unpredictable execution costs.

We propose an Adaptable Mobile Transaction model (AMT) that allows defining transactions with several *execution alternatives* associated to a particular context. This model offers the possibility of adapting transactions to the context and controlling execution costs with respect to application correctness criteria. The AMT model has been specified by using the formalism ACTA. We propose an analytical study which shows that (1) using AMTs increases commit rates, and (2) costs become predictable thanks to the possibility of choosing transaction execution types according to the context. To validate our approach, we defined the middleware TransMobi that manages environment awareness and implements the AMT model with suitable protocols. Among these protocols, we propose CO2PC, a commit protocol to ensure semantic atomicity. We developed a TransMobi prototype in PersonalJava that uses a relational databases, a WLAN and personal assistants (PDA).

Keywords: Transactions, mobile environment, atomicity, coherency, isolation, durability, consistency, context awareness, adaptation.