THÈSE DE DOCTORAT DE L'UNIVERSITÉ PARIS VI

RÉSUMÉ ÉTENDU[1]

Spécialité:

Informatique

présentée par

Claudia LEON LUNA

Pour obtenir le grade de

DOCTEUR de L'UNIVERSITÉ PARIS VI

Sujet de la thèse :

Contraintes d'intégrité et transactions imbriquées

Integrity constraints and nested transactions

Soutenue le 19 juillet 2001

devant le jury composé de :

Mme. Judith BARRIOS	Examinateur
Mme. Anne DOUCET	Directrice
Mme. Claudia RONCANCIO	Rapporteur
Mme. Marta RUKOZ	Co-directrice
Mme. Maria Esther VIDAL	Examinateur
M. José AGUILAR	Rapporteur
M. Stéphane GANÇARKI	Encadrant

1. La version complète de cette thèse, effectuée en co-tutelle entre l'Université Pierre et Marie Curie et l'Université Centrale du Vénézuela, est rédigée en espagnol. Elle peut être obtenue en s'adressant directement à l'auteur (cleon@strix.ciens.ucv.ve) ou bien à Stephane.Gancarski@lip6.fr.

Résumé

Cette thèse de doctorat propose de nouveaux développements dans le contexte des transactions imbriquées. Parmi les modèles de transactions avancées, les transactions imbriquées constituent sans doute le plus connu et le plus souvent objet de mise en œuvre. Ce modèle permet de décomposer une transaction en une hiérarchie de sous-transactions pouvant s'exécuter en parallèle, chacune constituant une unité de reprise après défaillance. Cette hiérarchie est représentable par une structure d'arbre.

La première partie consiste en l'évaluation d'un système qui supporte des transactions imbriquées, dans le but d'étudier son comportement dans les systèmes reparties (chapitre 2).

La seconde partie étudie le problème de la vérification de contraintes pour des transactions imbriquées (chapitre 3 à 5). Cette étude est faite dans un premier temps d'un point de vue conceptuel, c'est-à-dire en considérant une base de données centralisée. Dans ce cadre, nous avons proposé une solution où l'idée principale est d'associer le contrôle d'une contrainte au plus petit ancêtre commun des transactions feuilles qui touchent la contrainte. Cette solution intègre le contrôle d'exécution des transactions imbriquées avec la capacité de vérifier des contraintes d'intégrité aussitôt que possible pendant l'exécution d'une transaction. Cette solution a été mise en œuvre comme un prototype, avec l'objectif de valider et d'étudier le comportement de l'algorithme proposé. Dans un second temps nous avons adapté cette approche à un environnement multibase : à l'aide d'une typologie, nous avons établi des stratégies différentes de vérification en fonction de la nature des contraintes et de la structure des transactions. La solution proposeée dans cette thèse ne perturbe pas le contrôle d'exécution des transactions imbriquées, ce qui le rend très flexible. La transparence est fournie puisque les utilisateurs n'ont pas besoin d'ajouter de code de contrôle dans les définitions des contraintes ou des transactions.

Mots clés : Transactions imbriquées, contraintes d'intégrité, base de données orientées objet, systèmes distribués.

Table des matières

Introduction

Les systèmes répartis sont actuellement la norme pour structurer de nombreux systèmes informatiques. Ils sont utilisés pour l'implantation de systèmes interactifs généraux, ainsi que pour une grande variété d'applications commerciales [13]. Les applications réparties supportent de nombreux utilisateurs qui accèdent de manière concurrente aux mêmes informations stockées sur ces systèmes répartis. Développer des applications réparties est une tâche difficile, car celles-ci doivent offrir à l'utilisateur la transparence d'architecture et de composants, tant matériels que logiciels, du système réparti sous-jacent. En d'autres termes, l'utilisateur dispose d'une application intégrée et ne doit pas s'occuper de détails tels que la localisation des informations et des ressources, l'exécution de processus sur des ordinateurs distants, ou la synchronisation et la communication entre ces processus.

Le concept de transaction constitue une manière de structurer les applications réparties dans des unités de travail pouvant être vues comme des actions atomiques. Le modèle de transaction le plus simple, et le plus utilisé, est celui des transactions plates (Flat Transactions). Ces transactions ne possèdent qu'un seul niveau de contrôle, c'est-à-dire que soit les effets de toutes ses actions sont reflétés à la fin de la transaction (*la transaction est validée*) soit aucun effet n'est reflété (*la transaction est abandonnée*). Les transactions plates sont supposées satisfaire les quatre conditions suivantes, connues comme *les propriétés ACID* [27] :

L'Atomicité, ou propriété de *"tout ou rien"*, fait référence au fait que toutes les opérations d'une transaction doivent être traitées comme une seule unité, donc soit toutes les opérations sont exécutées, soit aucune.

La Cohérence demande qu'une transaction soit correcte, c'est-à-dire que si elle est exécutée toute seule, la transaction amène la base de données d'un état cohérent à un autre état cohérent. Classiquement, la cohérence est assurée par la définition de contraintes d'intégrité, qui sont des assertions définies sur la base de données qui doivent être vérifiées à la fin de chaque transaction. Un état d'une base de données est cohérent si toutes les contraintes d'intégrité sont satisfaites.

L'Isolation exige que chaque transaction utilise une base de données cohérente, c'est- à-dire, les effets intermédiaires d'une transaction sont invisibles aux transactions concurrentes.

La Durabilité impose que les résultats d'une transaction qui a déjà validée soient permanents dans la base de données malgré d'éventuelles défaillances.

Le modèle de **transactions plates** fut développé à l'origine pour des applications de bases de données bancaires, où les transactions sont généralement courtes et leur atomicité est de grande importance. L'extension des bases de données dans les environnements décentralisés a fait que les transactions soient utilisées maintenant dans de nouveaux domaines. Elles sont utilisées dans des applications qui vont des processus du contrôle industriel au travail coopératif. Pour de telles applications, le modèle des transactions plates n'est pas adapté, compte tenu des délais d'exécution et des restrictions qu'il impose. Pour répondre aux besoins de ces nouvelles applications, un certain nombre de modèles de transactions avancées ont été développés (voir description dans [28, 21, 37]).

La plupart des modèles de transactions avancées reflètent la vue individuelle de leurs auteurs sur les exigences et conditions du domaine des applications qui seront la cible du modèle de transaction respectif. Par conséquent, un modèle des transactions peut être convenable pour certains domaines d'application, alors qu'il peut être peu approprié pour d'autres. Entre les différents modèles de transactions avancées, on distingue celui des **transactions imbriquées** qui a été proposé par Moss au début des années 80 [45], mais qui est persistant encore aujourd'hui. Les avantages principaux des transactions imbriquées sont le support de modularité, la reprise après défaillance et le parallélisme intra-transaction.

Une **transaction imbriquée (TI)** peut contenir un nombre arbitraire de sous-transactions, et chacune à son tour, peut contenir également plusieurs sous-transactions. La transaction entière forme par conséquent un arbre, appelé *arbre de transactions*, qui possède ainsi une profondeur arbitraire. Le sommet de l'arbre est la *transaction racine*. Les transactions ayant des sous-transactions sont appelées *pères*, leurs sous-transactions sont leurs *filles*. Les transactions situées entre une transaction donnée et la racine de l'arbre sont appelées ses *ancêtres*.

Chaque sous-transaction d'une TI est exécutée indépendamment, et éventuellement en parallèle. Cela signifie qu'elle peut décider de valider ou d'abandonner indépendamment d'autres sous-transactions. Si une sous-transaction décide de valider, cette validation n'est pas définitive : ses mises à jour seront effectuées seulement si tous ses ancêtres décident de valider, c.-à-d. lorsque la transaction racine termine. Si une sous-transaction abandonne, ses sous-transactions doivent également abandonner, mais pas forcément sa transaction père. Cela oblige à redéfinir la notion d'atomicité d'une transaction (-tout ou rien-) pour les TI. Lorsque la transaction racine est validée, il faut garantir que seront effectuées les mises à jour de toutes les sous-transactions qui ont décidé de valider et dont aucun ancêtre n'a abandonné. Une TI peut donc se terminer, en conservant la cohérence, même si certaines de ses sous-transactions ont abandonné. Cela signifie que toutes les transactions d'une TI (y compris la racine) doivent respecter cette nouvelle définition d'atomicité ainsi que la propriété classique d'isolement. Cependant, seule la racine d'une TI se doit de conserver la cohérence des données et leur durabilité.

Les transactions imbriquées ont été l'objet d'études approfondis. Le verrouillage à deux phases classique a été adapté aux transactions imbriquées par Moss [45], et sa correction a été prouvée par Lynch [42]. D'autres algorithmes ont été proposés pour le contrôle de la concurrence [33, 43], la détection d'interblocages [50] et la reprise après des défaillances [46]. Différentes variantes de ce modèle peuvent être trouvées dans [21]. Plusieurs systèmes supportant des transactions imbriquées ont été développés, comme Argus [41], Camelot [23], ou le SGBD objet Versant [6]. Le travail décrit dans [8] présente une architecture à trois niveaux pour supporter des transactions imbriquées sur un SGBD commercial. Récemment, avec la commercialisation d'Encina, un moniteur transactionnel qui fournit des transactions imbriquées, beaucoup d'applications industrielles et bancaires ont été développées en utilisant ce modèle de transactions [1]. Des travaux montrent aussi que le modèle de transactions imbriquées est approprié pour les environnements multibases [9], pour les bases de données fedérées [58] ou les bases de données réparties [45]. Dans ces environnements dans lesquels les données sont stockées sur des sites différents, les mises à jour sur différents sites peuvent être exécutées par différentes sous-transactions d'une transaction imbriquée.

Malgré les innombrables travaux dans ce domaine pendant les quinze dernierés années, il existe encore au moins deux questions que l'on peut se poser par rapport aux transactions imbriquées :

1. Quel est le comportement réel des TI sur les systèmes répartis?
2. Comment peut-on gérer les contraintes d'intégrité dans les transactions imbriquées pour en assurer la propriété de cohérence?

C'est à ces deux questions que nous tentons de répondre dans cette thèse en cotutelle. Pour cela, nous nous basons d'une part sur l'expérience acquise au C.C.P.D. de l'Université Centrale du Venezuela dans le domaine des transactions imbriquées [50, 39, 40] et d'autre part sur celle acquise au LIP6 de l'Université P. et M. Curie Paris 6 dans le domaine du maintien de la cohérence dans les bases de données [4, 49, 17, 44].

Sujet de la thèse

L'objectif de cette thèse est d'une part d'étudier et évaluer des systèmes qui supportent des transactions imbriquées réparties, et d'autre part de proposer et valider une solution pour la vérification des contraintes d'intégrité dans les systèmes de bases de donnés qui supportent des transactions imbriquées.

L'émergence de systèmes supportant les transactions imbriquées [1, 51, 8] nous a amenés à poursuivre le premier but : proposer un modèle pour évaluer ces systèmes, en tenant compte que les critères d'évaluation ne peuvent jamais être généraux parce qu'ils dépendent des caractéristiques particulières de chaque système. Dans cette optique, nous avons étudié un cas particulier, le système SIMA : un outil qui supporte des transactions imbriquées sur un réseau hétérogène [51]. Ce système fut développé au C.C.P.D., de l'Université Centrale du Venezuela pour la gestion d'applications de traitement d'images médicales. Nous avons réalisé l'évaluation de SIMA pour extraire le coût de calcul que la gestion des transactions imbriquées ajoute au temps d'exécution d'une application.

D'autre part, la complexité croissante d'applications actuelles nécessite le développement de systèmes qui garantissent la cohérence de la base de données, c'est-à-dire que la base de données doit être sémantiquement correcte. Ceci est classiquement assuré par la définition de contraintes d'intégrité qui sont des assertions définies sur la base de données, et qui doivent être vérifiées à la fin de chaque transaction. Beaucoup de travail a été consacré à ce problème [47, 11, 32, 10, 30] et beaucoup de SGBD fournissent maintenant ces fonctionnalités [35, 56, 22]. Une synthèse des différentes approches est présentée dans [29] et [24].

Dans la plupart des cas, les systèmes de vérification de contraintes d'intégrité sont conçus pour le modèle simple et classique des transactions plates et très peu de solutions ont été proposées pour la gestion des contraintes d'intégrité dans le contexte des transactions imbriquées. Bien qu'il apparaisse naturel que les contraintes d'une transaction imbriquée soient vérifiées à chaque sous-transaction, la solution ne s'avère pas des plus efficaces. En effet, le système peut se trouver face à un problème de redondance des vérifications si plusieurs sous-transactions *touchent* une même contrainte. Par ailleurs, le système peut détecter une incohérence en fin de sous-transaction que l'on peut qualifier de temporaire dans la mesure où une autre sous-transaction qui touche la même contrainte et n'a pas encore fini, peut corriger cette incohérence. Il est important de rappeler que dans le contexte des transactions imbriquées la propriété de cohérence, doit être satisfaite uniquement par la transaction racine [45].

Dans [38], les auteurs présentent un modèle de transactions, nommé NT/PV (Nested Transactions with Predicats and Versions), qui permet la concurrence pour les transactions de longue durée sans sérialisation. Dans le modèle NT/PV, chaque transaction a une pré-condition et une post-condition. La pré-condition de chaque transaction décrit l'état de la base de données qui est exigé pour exécuter correctement la transaction. La post-condition décrit l'état de la base de données qui existerait après l'exécution isolée d'une transaction sur un état de la base de données qui satisfait sa pré-condition. Dans cette approche, les contraintes d'intégrité sont intégrées dans la post-condition de la transaction racine seulement. Cela ne permet de vérifier les contraintes d'intégrité qu'à la fin de la transaction racine.

La vérification de contraintes d'intégrité dans le contexte de transactions imbriquées est également abordée dans [16]. Les contraintes d'intégrité sont définies au niveau des méthodes, à l'aide de pré-et post-conditions associées à un mécanisme de gestion d'exceptions. Une exception est levée lorsqu'une pré- ou post-condition est violée. Dans cette approche, le programmeur doit lui-même décrire, de façon procédurale, les actions à effectuer lorsqu'une contrainte est violée. Il doit également déterminer à quel niveau dans les transactions imbriquées ces actions sont définies et exécutées.

A notre connaissance, l'ensemble de ces travaux ne tient pas compte des transactions imbriquées réparties.

Nous nous sommes intéressés dans cette thèse à la recherche de solutions à la vérification de contraintes en présence de transactions imbriquées, visant à :

- S'appliquer à des TI réparties,

- Vérifier les contraintes le plus tôt possible au cours de l'exécution de la transaction imbriquée, de manière à défaire le moins possible de la transaction en cas de violation de la cohérence.

- Ne pas imposer à l'utilisateur de spécifier la stratégie de vérification des contraintes, mais que celle-ci soit déterminée automatiquement par le système.

- Interférer le moins possible avec les mécanismes sous-jacents de contrôle de l'exécution et de contrôle de la concurrence, de manière à obtenir l'approche la plus générale et portable possible.

- Spécifier les contraintes d'intégrité de façon globale et déclarative. Une analyse syntaxique des contraintes et des transactions permet de réduire l'ensemble des contraintes à vérifier, en déterminant un ensemble de contraintes risquant d'être violées par une transaction.

Réalisation

Dans un premier temps, nous nous sommes consacrés à l'étude du comportement des systèmes qui supportent les transactions imbriquées réparties. Pour cela nous avons réalisé l'évaluation de SIMA: un système qui supporte des transactions imbriquées sur un réseau hétérogène. SIMA fut développé pour la gestion des applications de traitement des images médicales basées sur le modèle de transactions imbriquées.

Dans un deuxième temps, nous avons étudié le problème de la vérification de contraintes pour des transactions imbriquées d'un point de vue conceptuel, c'est-à-dire en considérant une base de données centralisée. Dans ce cadre, nous avons proposé une solution où l'idée principale est d'associer le contrôle d'une contrainte au plus petit ancêtre commun des transactions feuilles qui touchent la contrainte. Cette solution a été mise en œuvre dans l'Universidad Central de Venezuela comme un prototype, avec l'objectif de valider et d'étudier le comportement de l'algorithme proposé.

Ensuite, la problématique a été étendue aux multibases, dans lesquelles nous avons pris en compte le coût lié au transfert de données entre les nœuds chargés de vérifier les différentes contraintes. Nous avons considéré certains types de contraintes pour lesquelles nous avons cherché la meilleure stratégie de vérification dans un environnement réparti, en fonction de la répartition des objets sur les différents sites. De cette manière, nous avons considéré des cas de difficulté croissante et nous avons obtenu une classification des contraintes en fonction de leur vérification dans un environnement réparti. Cette classification nous a permis ensuite de proposer une stratégie de vérification pour chaque classe de la typologie, lorsque les transactions sont imbriquées et réparties.

Plan de la thèse

Cette thèse est composée de six chapitres. Le premier chapitre rappelle les caractéristiques des différents modèles des transactions avancées. Nous présentons en détail celles du modèle de transactions imbriquées, qui est le sujet d'étude de cette thèse.

Dans le chapitre II nous présentons l'évaluation d'un système qui supporte les transactions imbriquées.

Le chapitre III constitue un état de l'art présentant les principales approches utilisées pour la gestion de la cohérence en bases de données.

Notre approche pour la vérification des contraintes d'intégrité dans les systèmes de bases des donnés qui supportent transactions imbriquées est présentée dans le chapitre IV.

Le chapitre V décrit la mise en œuvre de la méthode proposée dans le chapitre précédent, ainsi que une exemple d'application où notre approche sera bien adaptée.

Dans le dernier chapitre, nous présentons la conclusion de cette thèse, mettant en évidence tant les différents apports de notre contribution ainsi que les limites d'une telle approche.

Dans la suite de ce document nous présentons un résumé de chaque chapitre considéré dans ce plan de thèse.

Chapitre 1

Modèles des transactions avancées

En réponse aux besoins de nouvelles applications, un certain nombre de modèles de transactions avancées ont été développés (voir descriptions dans [28, 21, 37]). La plupart de ceux-ci ont été proposés pour applications dans une domaine particulière. Par conséquent, un modèle des transactions peut être convenable pour certains domaines d'application, alors qu'il peut être peu approprié pour d'autres et aucun modèle ne fournit la généralisation qu'on désire. Dans la suite du chapitre, nous présentons les caractéristiques de quelques modèles des transactions avancées. Nous commençons par les transactions imbriquées qui sont traitées plus en détail car elles sont le sujet d'étude dans cette thèse. Ensuite, nous décrivons brièvement d'autres modèles.

1.1 Transactions imbriquées

Les transactions imbriquées présentées par Moss conviennent bien aux systèmes réparties et aux transactions longues qui peuvent être découpées en plusieurs tâches logiquement indépendantes [1, 45, 8, 51]. En effet, une tâche complexe peut être décomposée en sous-tâches logiques pouvant s'exécuter indépendamment, qui, à leur tour, peuvent être divisées en d'autres sous-tâches logiques. Dans le contexte des transactions imbriquées, une tâche complexe correspond à la transaction racine et chaque tâche logique correspond à une sous-transaction.

Une Transaction Imbriquée (TI) est un arbre de transactions dont les composants sont ses sous-transactions. Dans ce modèle, une transaction peut contenir un nombre arbitraire de sous-transactions, chacune pouvant être composée de plusieurs sous-transactions. Un arbre de transactions est ainsi de profondeur arbitraire. Le sommet de l'arbre est la *transaction racine*. Les transactions ayant des sous-transactions sont appelées *pères*, leurs sous-transactions sont leurs *filles*.

Plus précisément, une TI est une transaction formée par :

- des opérations sur les objets de la base de données, et/ou
- d'autres transactions (des sous-transactions) qui peuvent aussi être des TI.

Chacune des sous-transactions d'une TI est exécutée indépendamment, et éventuellement en parallèle. Ceci signifie qu'elle peut décider de valider ou d'abandonner indépendamment des autres sous-transactions. Cette décision peut dépendre des résultats de l'exécution de ses sous-transactions. Si une sous-transaction abandonne, ses sous-transactions doivent également abandonner, mais pas forcément la transaction père. Si une sous-transaction décide de valider, cette validation n'est pas définitive : ses mises à jour seront exécutées seulement si toutes ses transactions ancêtres décident de valider, c'est-à-dire lorsque la transaction racine termine. Ceci oblige à *redéfinir la notion d'atomicité (-tout ou rien-)* pour les TI. Lorsque la transaction racine valide, il faut garantir que les mises à jour de toutes les sous-transactions qui ont décidé de valider et dont aucun ancêtre n'a abandonné, seront effectuées. Une TI peut donc terminer, en conservant la cohérence, même si certaines de ses sous-transactions ont abandonné. On parle alors *d'abandon partiel.*

Lorsque plusieurs transactions s'exécutent en concurrence, le système de gestion doit assurer la cohérence de leurs opérations respectives sur le système, en empêchant que ces opérations n'interfèrent. Un mécanisme de verrouillage a deux phases peut être utilisé pour assurer l'atomicité des transactions en milieu concurrent. Dans les transactions imbriquées il faut prendre en compte leur comportement pour la gestion de verrous afin d'éviter les abandons en chaîne de transactions et de garantir la seriabilité des actions effectuées par des transactions. La solution proposée dans [45] est la suivante : au lieu de relâcher les verrous lors qu'une sous-transaction valide, ils sont hérités par son père. Ainsi une transaction peut posséder des verrous soit par une demande soit par un héritage. Une transaction ne peut utiliser un objet que si elle obtient son verrou par demande. On établit ainsi les règles de gestion des verrous suivantes :

1. Une transaction ne peut obtenir un verrou que si celui-ci est libre ou que si toute transaction qui le possède par héritage est un de ses ancêtres.
2. Lorsqu'une transaction abandonne tous ses verrous sont libérés par cette transaction, mais pas par les transactions qui les possèdent par héritage.
3. Lorsqu'une transaction valide, ses verrous sont hérites par son père.
4. Lorsqu'une transaction ne peut pas obtenir un verrou, elle doit attendre jusqu'à ce qu'elle puisse l'obtenir.

5. Une fois qu'une transaction a obtenu un verrou, soit par demande soit par héritage, elle ne peut pas le libérer avant de finir (avant d'abandonner ou de valider).

Un verrou peut être possédé par plusieurs transactions, mais seulement une d'entre elles peut le posséder par demande, donc une seule peut l'utiliser. Il a été prouvé que ces règles garantissent la seriabilité des transactions imbriquées [42]. De plus, ces règles assurent que si une transaction abandonne, seule ses descendantes sont entraînées par cet abandon. Les objets utilisés par cette transaction n'ont été utilisés que par des transactions dans la sous-arborescence dont elle est racine.

Ainsi, le comportement des transactions imbriquées par rapport aux *propriétés ACID* est le suivant : toutes les transactions d'une TI (y compris la racine) doivent respecter cette nouvelle définition d'atomicité ainsi que la propriété classique d'isolement. Cependant, seulement la racine doit conserver la cohérence des données et leur durabilité.

Dans le cadre de cette thèse, nous supposons que seules les sous-transactions feuilles peuvent faire des mises à jour sur la base de données. Ce sont donc les seules transactions susceptibles de violer les contraintes d'intégrité. Ceci n'enlève rien à l'expressivité du modèle, comme cela est montré dans [45], mais permet de simplifier le processus de vérification des contraintes d'intégrité.

Nous supposons aussi, que la structure des TI peut être connue au moment de la compilation. Plus précisément, nous supposons qu'il est possible de connaître au moment de la compilation le plus petit ancêtre commun d'un ensemble de sous-transactions feuilles d'une TI.

1.2 Autres modèles de transactions avancées

Sphères de contrôle

Le premier modèle des transactions avancées est celui des sphères de contrôle (Spheres of Control) qui a été proposé dans les années 70 par Davies [14]. Ce modèle propose de maintenir toute l'histoire des processus afin d'avoir les dépendances qui existent entre les différentes valeurs. Lorsqu'on a besoin de changer une valeur donnée, on pourra aussi réaliser les changement sur les autres valeurs qui sont affectées. La complexité de sa mise en œuvre a fait que ce modèle n'a pas été réalisé en pratique.

Sagas

Les Sagas [26] sont des transactions avec deux niveaux de contrôle, où un contrôle optimiste de la concurrence permet que les objets mis à jour par une transaction T puissent être utilisés par d'autres transactions avant la validation de T. Le concept des Sagas est basé sur des transactions de compensation: *pour une transaction T, une transaction de compensation notée C n'est qu'une transaction qui peut annuler sémantiquement les effets de T, après que T a été validée.* Une Saga est une transaction longue qui consiste en une séquence de sous-transactions relativement indépendantes. Pour exécuter une Saga, le système doit garantir que soit toutes ses sous-transactions sont complétées soit toute exécution partielle est annulée grâce à des sous-transactions de compensation.

Transactions imbriquées ouvertes

Les transactions imbriquées ouvertes constituent des hiérarchies des sous-transactions sur différents niveaux d'abstraction et des relations de commutativité définissent à chaque niveau les conflits entre opérations prédéfinies [21, 58, 59]. La différence entre les transactions imbriquées et les transactions imbriquées ouvertes, est que dans le modèle ouvert, les sous-transactions peuvent valider (c.-à-d., leurs effets sont faits visible à d'autres transactions) indépendamment de la validation de la transaction racine. L'abandon d'une transaction doit être rendu effectif au moyen des opérations inverses de haut-niveau qui compensent les sous-transactions complétées.

Transactions multi-niveau

Une transaction multi-niveau est un arbre des transactions équilibré, c'est-à-dire dont toutes les feuilles sont à la même profondeur [59]. Le modèle des transactions multi-niveaux permet de tirer parti de la sémantique des objets typés, et de leur mode de construction hiérarchique, dans le but d'accroître la concurrence. Il impose que la hiérarchie d'objets soit construite suivant des niveaux d'abstraction successifs et que la décomposition de la transaction globale reflète cette construction hiérarchique des objets et de leurs opérations. La décomposition entraîne que toutes les opérations définies à un même niveau d'abstraction correspondent, à l'exécution, à des sous-transactions situées à la même profondeur dans l'arbre de chaque transaction globale. Cela permet d'exploiter la commutativité des opérations et l'indépendance des objets définis à chaque niveau, pour accroître la concurrence entre des transactions.

ConTract

Dans [57] un ConTract est défini comme un ensemble d'actions prédéfinies (appelés

Step) qui ont une spécification explicite du flux du contrôle entre elles. Le ConTract a été proposé pour définir et contrôler des calculs complexes et de long durée dans des applications non standard en bureautique, en CAO et le contrôle de processus industriels. L'exécution d'un ConTract doit être récupérable en avant. Cela signifie que lorsque l'exécution d'un ConTract est interrompue à cause d'une défaillance, elle doit être ré-instanciée et continuée à partir du point où elle a été interrompue. Pour cela, toute l'information de l'état (y compris l'état de la base de données, des variables du programme de chaque Step, et l'état global du ConTract) doit être récupérable. Un ConTract peut faire connaître ses résultats partiels avant que le ConTract entier soit complet. En cas de défaillance, les transactions de compensation sont utilisées pour annuler les résultats des Steps qui ne sont plus valables. Le modèle de ConTract utilise des invariants définis sur la base de données pour le contrôle de concurrence et permet de résoudre des conflits de manière flexible en spécifiant explicitement la marche à suivre quand les conflits se produisent.

Transactional Activity Model

Les TAM (Transactional Activity Model) sont une extension des transactions imbriquées qui permet de définir explicitement la portée de l'exécution d'une transaction au moyen de la définition des règles actives [15].

ACTA

Le formalisme ACTA a été proposé comme un outil de spécification formelle permettant de caractériser le comportement et les propriétés des modèles de transactions [12]. ACTA est basé sur la logique du premier ordre et ses règles permettent de caractériser les effets qu'une transaction produit sur les autres transactions ainsi que sur les objets qu'elle manipule.

Fermetures des transactions (transaction closures)

Les fermetures des transactions [52] étendent le formalisme ACTA en considérant des transactions ayant une structure plus générale. Elles considèrent des relations plus riches entre les transactions, par exemple que des transactions filles peuvent terminer après leur père et lexistence de dependences entre plus de deux transactions.

Chapitre 2

Evaluation d'un système qui supporte les transactions imbriquées

Dans ce chapitre nous présentons l'évaluation d'un système qui supporte les transactions imbriquées. Dans une première partie nous décrivons l'architecture du système sur lequel nous avons travaillé. Il s'agit de SIMA : System for distributed IMage Applications. Dans une deuxième partie, nous présentons le modèle d'évaluation expérimentale utilisé ainsi que les résultats que nous avons obtenus par rapport à la performance de SIMA.

2.1 Architecture de SIMA

SIMA est un système qui supporte la construction et le contrôle d'exécution des applications réparties pour le traitement des images. Ces applications sont basées sur le modèle de transactions imbriquées. Une application consiste en une composition de transactions qui peuvent être imbriquées. Ceci a pour résultat une hiérarchie, aussi grande que désirée, où la racine de l'arbre correspond à la transaction racine, les feuilles correspondent à opérations de base et les nœuds intermédiaires sont des opérateurs de contrôle. Le système inclut deux types d'opérateurs de contrôle : '//' indique l'exécution parallèle et ';' établit l'exécution en série. Ainsi, une TI présente un parallélisme interne défini explicitement.

Les images et les opérations sont distribuées sur le réseau comme suit:

1. Chaque image réside totalement sur un seul nœud.

2. Chaque opération est localisée et est exécutée sur un nœud seul.
3. Pour appliquer une opération sur une ou plusieurs images, il est nécessaire d'envoyer une copie des images au nœud où l'opération réside, si elle est sur un nœud différent.

Dans SIMA, l'utilisateur construit un arbre de transactions au moyen d'une interface graphique. Cet arbre définit l'application par composition des opérations de base. Après il demande au système l'exécution de cet arbre. Il est possible alors d'utiliser les ressources disponibles sur un réseau hétérogène pour le traitement des images. SIMA permet le développement d'applications dans un environnement coopératif. En effet, SIMA utilise les transactions imbriquées pour contrôler l'exécution d'applications distribuées et l'accès concurrent à l'ensemble d'images qui pourraient être traitées simultanément par des applications d'utilisateurs différents ou par des opérations d'un seul utilisateur.

La structure de SIMA a été basée sur des modules identiques dupliquées sur chaque ordinateur. Ces modules communiquent au moyen des messages pour accoupler les fonctions globales de l'outil. La communication entre les différents modules est basée sur les sockets JAVA. Dans la figure 2.1, l'architecture du système est présentée.

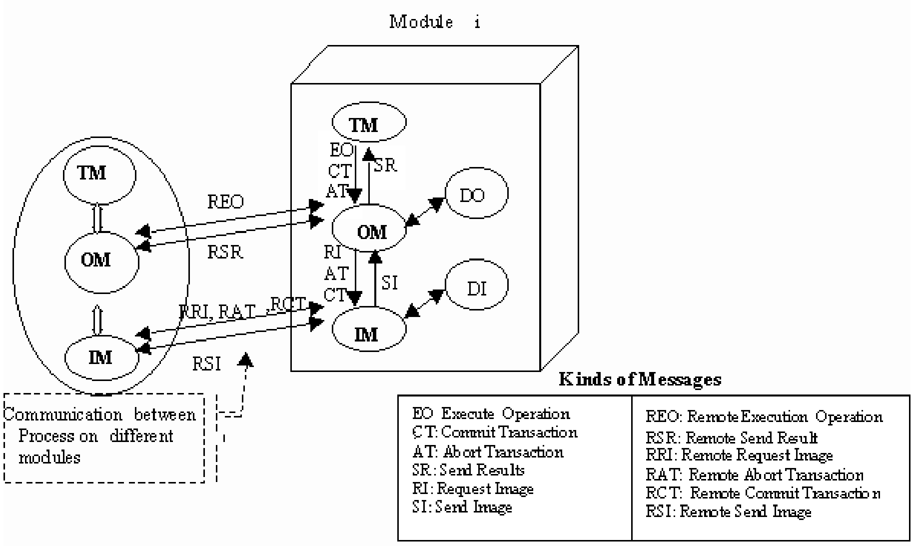

FIG. 2.1 – *Architecture de SIMA*

Chaque module a les fonctions suivantes:

1. Contrôle d'exécution des transactions imbriquées initiées dans l'ordinateur local, d'après le comportement des transactions exposé dans le chapitre 1.

2. Exécution des opérations de base qui sont localisées dans l'ordinateur local.

3. Contrôle des accès concurrents à des images qui résident dans l'ordinateur local. Pour cela, il applique l'algorithme de contrôle de concurrence et lorsque l'accès est accepté, il transmet une copie de l'image au site qui le demande.

Chaque module est composé de trois gestionnaires qui communiquent hiérarchiquement. Dans la figure 2.1, les gestionnaires composants chaque module et leurs relations sont présentées. Le gestionnaire des transactions (TM) est responsable de l'interface avec l'utilisateur et du contrôle d'exécution de l'arbre de l'application. Le gestionnaire des opérations (OM) demande les images impliquées dans une opération au gestionnaire des images (IM), exécute des opérations sur ces images et alors envoie les résultats au TM. Le gestionnaire des images (IM) est responsable des images (contrôle de la concurrence et persistance).

2.2 Evaluation de SIMA

Dans cette partie, nous présentons le modèle d'évaluation expérimentale utilisé ainsi que le résultat que nous avons obtenu par rapport à la performance de SIMA.

Cette évaluation consiste, dans une première étape, à caractériser les applications possibles qui peuvent être construites avec SIMA : principalement la taille des arbres de transactions, leur profondeur et la quantité de feuilles qui peuvent les composer. Cette caractérisation nous a permis de générer automatiquement des arbres de transactions de structure aléatoire qui peuvent être exécutées sur SIMA.

L'évaluation de la performance de SIMA a été réalisée avec l'objectif de mesurer le temps pris pour réaliser les tâches du contrôle et de la communication. En particulier, nous avons mesuré le **coût ajouté** par l'outil à l'exécution d'une application. Ce **coût ajouté** dépend du temps moyen que l'outil prend pour:

- le contrôle de l'exécution de l'arbre d'une application,
- la communication entre les modules de SIMA,
- le contrôle de la concurrence, et
- la transmission et gestion des images.

Pour déterminer ce temps, nous avons produit de façon aléatoire 800 arbres des transactions dont les dimensions varient entre 1 et 64 feuilles (quantité d'opérations de base). La localisation des opérations, la localisation des images, le nombre de sous-transactions pour chaque transaction et le type de chaque opérateur (en série ou parallèle) ont été produits en utilisant une distribution uniforme dans chaque cas. Nous avons modélisé les arrivés des demandes d'exécution des arbres en utilisant une distribution de Poisson.

Le **coût ajouté** par l'outil a été mesuré sans prendre en considération le temps d'exécution des opérations de base. Cela veut dire que nous avons supposé des *opérations nulles* (ayant un temps d'exécution égal à 0). Cependant, le temps de génération des processus *(threads)* et de gestion des images ont été pris en considération. Dans cette évaluation nous avons considéré des images de 512x512 pixels à 8-bits.

Avec le modèle décrit ci-dessus, le temps moyen d'exécution d'une transaction, aussi connu dans la littérature comme *temps de réponses*, était 46.47 secondes avec un niveau de confiance de 0.90 et une marge d'erreur de 2 secondes. Comme nous avons considéré des *opérations nulles*, ce temps de réponses correspond au **coût de contrôle ajouté par SIMA** à l'exécution d'une application.

Le coût ajouté par l'outil pour l'exécution d'une application (46.47 secondes) est negligeable comparé au temps que l'utilisateur gagne dans la construction des applications dans le domaine de traitement des images médicales.

Cette expérience sera publié prochainement dans la review *Parallel and Distributed Computing Practices, special issue on Distributed Object-Oriented Systems* [51].

Chapitre 3

Gestion de contraintes d'intégrité dans les bases de données

Ce chapitre présente les principales approches utilisées pour la gestion de la cohérence dans les bases de données, et particulièrement l'approche dans laquelle se situe cette thèse.

Tout utilisateur de bases de données doit impérativement travailler sur une base cohérente. On part ainsi du principe que toute base de données est cohérente et qu'une transaction, si elle est exécutée toute seule dans une base de données qui est cohérente, laissera la base de données dans un autre état cohérent. Cette fonctionnalité est généralement assurée par des contraintes d'intégrité (CI) qui sont des assertions logiques devant toujours être vérifiées par les données de la base. Une base de données est cohérente si et seulement si toutes les contraintes sont satisfaites[1].

Une transaction qui met à jour les objets de la base de données doit conserver les contraintes d'intégrité de la base de données. La vérification de l'ensemble de ces contraintes, à l'issue de toute mise à jour, est très coûteuse et réduit la disponibilité de la base de données. C'est pourquoi, entre autres, les premiers systèmes de gestion de bases de données ne supportaient pas la définition de contraintes d'intégrité trop coûteuses à vérifier. Traditionnellement on supposait que chaque utilisateur est complètement informé de toutes les contraintes et écrit seulement des programmes qui sont corrects.

Prenant en compte que l'hypothèse que tout programme est correct est irréaliste, de nombreux travaux ont abordé le problème de la gestion des contraintes d'intégrité [47, 11, 32, 10, 30]. Il existe plusieurs approches, présentées de façon synthétique dans [29, 24]. L'approche préventive consiste à prouver la correction des transactions de mise à jour par rapport aux contraintes d'intégrité avant de les exécuter [3]. Cer-

1. Dans cette thèse, on ne considère que des contraintes statiques

taines approches, telles que les règles actives et les déclencheurs (triggers) [60] sont bien adaptées aux transactions « on-line ». L'utilisateur doit alors déterminer lui-même, pour chaque contrainte, les événements déclenchant sa vérification. Les solutions consistant à vérifier les contraintes après l'exécution d'une transaction (**méthodes de détection**) utilisent généralement des techniques de compilation pour réduire le processus de vérification de contraintes lors de l'exécution [7, 34, 53, 36, 4, 17].

Notre travail se situe dans l'approche des **méthodes de détection**. La solution que nous proposons peut s'adapter à tout mécanisme de gestion de contraintes dans les SGBD objets utilisant les principes suivants :

1. Les contraintes d'intégrité sont déclarées de façon globale et déclarative.

2. Une analyse syntaxique des contraintes et des transactions permet de réduire l'ensemble de contraintes à vérifier, en déterminant l'ensemble de contraintes risquant d'être violées par une transaction.

3. La vérification des contraintes est faite automatiquement (par génération, à la compilation, de code vérificateur par exemple).

Les contraintes d'intégrité peuvent être définies au niveau des classes, ou de façon globale au niveau du schéma. Cette dernière approche est plus générale, et permet d'exprimer de manière uniforme les contraintes portant sur une seule ou plusieurs classes. L'expression sous forme déclarative des contraintes est simple, et s'intègre plus facilement à la déclaration du schéma.

L'analyse des contraintes et des transactions a pour objectif de réduire le nombre de contraintes à vérifier après l'exécution d'une transaction donnée. Dans cette thèse, nous avons considérons une analyse purement syntaxique, comme celle proposé dans Themis [4], mais notre approche est valable pour des méthodes d'analyse plus poussées, comme [5]. Le principe de base de l'analyse, purement syntaxique, consiste à détecter, pour chaque contrainte et chaque transaction, les structures impliquées. Le résultat de l'analyse syntaxique (d'une contrainte comme d'une transaction) est donc un ensemble de chemins de la base de données. Par la suite, on dira que l'analyse d'une contrainte détermine les objets impliqués par la contrainte, alors que l'analyse d'une transaction détermine les objets touchés par la transaction. Intuitivement, une transaction risque de violer une contrainte lorsqu'elle manipule des structures de données également manipulées par la contrainte, c'est-à-dire lorsque l'intersection des deux analyses syntaxiques n'est pas vide. Ces analyses permettent donc de déterminer, lors de la compilation d'une transaction, l'ensemble de contraintes à vérifier.

Chacune de ces contraintes est vérifiée de façon automatique, par génération de code de vérification en fin de transaction, ou par déclenchement de méthodes spécifiques. Pour optimiser le coût du processus de vérification, les algorithmes sont adaptés aux différents types de contraintes et sont instanciés pour chaque contrainte (seules les modifications portant sur les instances des chemins apparaissant dans l'analyse syntaxique de la contrainte sont prises en compte dans la vérification). Les algorithmes sont optimisés en fonction du type des contraintes. Il n'est pas nécessaire, par exemple, de vérifier les contraintes universelles sur tous les objets de la base impliqués par cette contrainte. Il est possible de limiter la vérification à un ensemble minimal d'objets. Cet ensemble, déterminé à l'exécution, comprend uniquement les objets dont les modifications par la transaction risquent de violer la contrainte.

Dans le chapitre suivant nous proposons des mécanismes de vérification de contraintes dans les transactions imbriquées, s'appuyant sur les principes décrits ci-dessus.

Chapitre 4

Vérification de contraintes d'intégrité aux transactions imbriquées

Notre approche pour la vérification de contraintes d'intégrité dans les systèmes de bases de données qui supportent les transactions imbriquées est présentée dans ce chapitre, qui comprend deux parties. Dans la première partie nous considérons le problème de la vérification de contraintes pour des transactions imbriquées en considérant une base de données centralisée. La deuxième partie étend la problématique aux multibases.

4.1 Vérification de contraintes dans les transactions imbriquées en bases de données centralisées

Dans cette partie nous considérons le problème de la vérification de contraintes pour des transactions imbriquées d'un point de vue conceptuel, c'est-à-dire en considérant une base de données centralisée. Dans cette optique, nous poursuivons un double but :

1. Vérifier les contraintes d'intégrité le plus tôt possible, c'est-à-dire aussitôt que les opérations susceptibles de violer la contrainte sont exécutées, et abandonner le moins possible de sous-transactions en cas de violation d'une contrainte.

2. Interférer le moins possible avec les mécanismes sous-jacents de contrôle de l'exécution et de contrôle de la concurrence, de manière à obtenir l'approche la plus générale et portable possible.

Dans [20], nous présentons une solution à ce problème. Celle-ci assure que si une transaction racine valide, alors la base de données est dans un état cohérent. Nous résumons dans la suite de cette section les principales caractéristiques de notre solution.

Soit une transaction imbriquée exécutée sur une base de données régie par un ensemble de contraintes. Les contraintes d'intégrité sont définies globalement, au niveau du schéma de la base de données et sont vérifiées par le système. Une analyse syntactique permet de savoir, à la compilation, les structures manipulées par les sous-transactions et les contraintes, et donc déterminer l'ensemble de contraintes qui peuvent être violées par la transaction imbriquée. Le code de vérification de la contrainte est inséré automatiquement dans la transaction.

L'idée forte de notre approche est de choisir, pour chaque contrainte, une sous-transaction qui sera responsable de sa vérification: le plus petit ancêtre des sous-transactions qui touchent la contrainte. On dit qu'une transaction touche une contrainte si elle est susceptible de la violer. A cet effet, nous utilisons les techniques d'analyse de l'approche «*des méthodes de détection*», décrite dans le chapitre précédent, pour déterminer à la compilation:

- Pour chaque sous-transaction feuille Tl_j d'une transaction imbriquée T, l'ensemble $C(Tl_j)$ de toutes les contraintes touchées par Tl_j.
- Pour une contrainte C_i, l'ensemble $T(C_i)$ des sous-transactions feuilles de T qui touchent C_i. On note $SCA(C_i)$ le plus petit ancêtre commun de toutes les sous-transactions incluses dans $T(C_i)$.

Dans la figure 4.1 l'arbre entier d'une transaction imbriquée T est représenté, chaque transaction est représentée par un nœud dans l'arbre. Au-dessous chaque transaction feuille Tl_j, l'ensemble des contraintes touchées par Tl_j, c'est-à-dire $C(Tl_j)$ est montré. Par exemple, $C(Tl_{211}) = \{C_2\}$. Nous pouvons voir que $T(C_1) = \{Tl_{212}, Tl_{222}\}$, c'est-à-dire la contrainte C_1 est touché par les transactions Tl_{212} et Tl_{222}. T_2 est le plus petit ancêtre commun de Tl_{212} et Tl_{222}, T_2 est donc le responsable pour le contrôle de C_1. La procédure du contrôle de chaque contrainte C_i est représenté par un rectangle et est associé avec la sous-transaction responsable pour l'exécuter. Par exemple, T_2 est associé avec la procédure du contrôle de C_1.

$SCA(C_i)$ est responsable de la maintenance de la contrainte C_i. Comme ce nœud contrôle l'exécution de son sous-arbre, il est possible d'assurer qu'en cas de violation de C_i, non seulement les sous-transactions touchant C_i vont être abandonnées, mais que les sous-transactions qui ont utilisé leur résultats vont être abandonnées en cascade. Par ailleurs, contrairement aux systèmes ne gérant que des transactions plates,

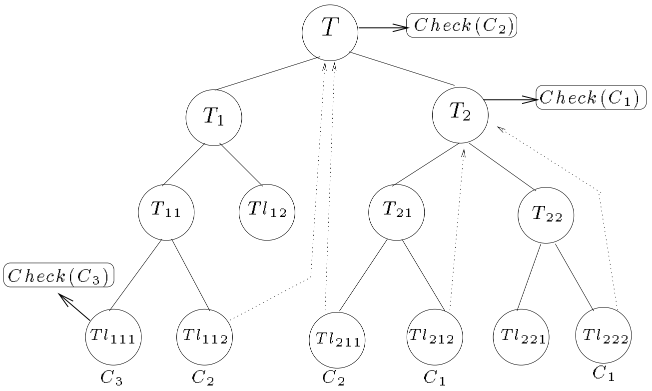

FIG. 4.1 – *Contraintes touchées par chaque sous-transaction d'une TI*

les autres sous-transactions qui ne sont pas concernées par C_i peuvent continuer leur exécution. La communication directe entre les sous-transactions feuilles touchant une contrainte et la sous-transaction responsable de sa vérification permet de lancer le processus de vérification aussitôt que toutes les feuilles touchant la contrainte ont terminé leur exécution. En cas de violation, la propagation vers le bas d'un message d'abandon permet de n'abandonner que ce qu'il est nécessaire pour restaurer la satisfaction de la contrainte. Nous prenons de plus en compte d'éventuelles re-vérifications d'une contrainte si une sous-transaction feuille qui la touche doit abandonner à cause d'une autre contrainte.

La sous-transaction $SCA(C_i)$ connaît l'identification des feuilles qui touchent C_i, et chacune d'entre elles connaît l'identification de $SCA(C_i)$. Lorsque ces dernières ont toutes fini leur exécution, $SCA(C_i)$ peut lancer le processus $Check(C_i)$ de vérification de C_i. Si C_i est satisfaite, rien d'autre n'est exécuté. Sinon, C_i est violée et un certain nombre d'actions sont exécutées au sein du sous-arbre de $SCA(C_i)$ en vue de rétablir la cohérence. Nous utilisons pour cela la notion *d'abandon partiel*, inhérente au modèle des transactions imbriquées, pour obtenir un état cohérent de la base de données sans abandonner entièrement la transaction imbriquée. Toutes les sous-transactions à abandonner (celles qui touchent C_i et celles qui ont déjà utilisé les résultats des ces premières) sont à l'intérieur du sous-arbre de $SCA(C_i)$. Dans le pire des cas, c'est le sous-arbre entier qui devra abandonner, mais les sous-transactions à l'extérieur de ce sous-arbre continuent leur exécution indépendamment de C_i, jusqu'à atteindre l'éventuelle validation de la transaction imbriquée.

Pour abandonner le moins de sous-transactions possibles dans le sous-arbre de $SCA(C_i)$, nous procédons comme suit: $SCA(C_i)$ envoie un message d'abandon à travers les branches de son sous-arbre qui contiennent des feuilles touchant C_i. Ce message est propagé vers le bas jusqu'à atteindre le plus petit ancêtre encore actif (c'est-à-dire n'ayant pas encore envoyé sa décision à son père) de chaque feuille incriminée. Ce nœud va prendre la décision d'abandonner, entraînant l'abandon de son sous-arbre. Cet abandon en cascade peut avoir des effets de bord sur d'autres contraintes qui doivent alors être re-vérifiées. En effet, une transaction feuille Tl_j ayant déjà fini son exécution, peut être forcée d'abandonner à cause de la violation de C_i (peu importe si Tl_j touche C_i ou non). Si Tl_j touche une autre contrainte C_k, il est possible que C_k ait déjà été vérifiée par rapport aux effets de toutes les feuilles qui la touchent. L'abandon de Tl_j implique donc de re-vérifier C_k par rapport aux feuilles qui la touchent à l'exception de Tl_j. Dans [20], nous montrons que, même si C_k doit être re-vérifiée plusieurs fois, la vérification effective (la dernière) intervient toujours au plus tôt, c'est-à-dire dès que l'état final de la base de données par rapport à C_k est produit. Nous montrons aussi qu'il est toujours possible de re-vérifier une contrainte, en d'autres termes qu'une situation ou une contrainte C_i devrait être re-vérifiée par une sous-transaction déjà validée (non active) est impossible.

Dans cette solution nous avons intégré le contrôle d'exécution de la transaction imbriquée avec la capacité de vérifier des contraintes d'intégrité aussitôt que possible pendant l'exécution d'une transaction. Ce mécanisme ne perturbe pas le contrôle d'exécution de la transaction imbriquée, ce qui le rend très flexible. La transparence est fournie puisque les utilisateurs n'ont besoin d'ajouter aucun code de contrôle dans les définitions des contraintes ou des transactions.

Notre approche fut présentée dans Transactions and Database Dynamics'99 à Dagstuhl, Allemagne en septembre 1999 et fut publiée dans le proceeding de la conférence [20].

4.2 Vérification de contraintes dans les transactions imbriquées en multibases

Dans les systèmes multibases [54] où existent plusieurs opérations sur plusieurs bases de données gérées par différents SGBD le maintien de la cohérence pose de nouveaux problèmes. En effet, dans de multibases, il faut déterminer où chaque contrainte sera stockée et vérifiée. Une telle décision peut dépendre du type de contrainte, du type de mise à jour et du site où la mise à jour est effectuée. Dans ce cadre, nous avons étudié l'adaptation de l'approche présentée dans la première partie, pour être exécutée dans les systèmes multibases. Notre but est de maintenir des contraintes d'intégrité globales. Ces contraintes touchent des données sur plusieurs sites. Nous avons donc étudié comment vérifier des contraintes d'intégrité globales pendant l'exécution d'une transaction imbriquée dans un systèmes multibase.

Comme méthodologie, nous avons considéré certains types de contraintes. Pour chacun, nous cherchons la meilleure stratégie de vérification dans un environnement multibase, en fonction de la répartition des objets sur les différents sites. De cette manière, nous avons considéré des cas de difficulté croissante et nous avons obtenu une classification de contraintes en fonction de leur vérification dans un environnement distribué. Cette classification nous a permis ensuite de proposer une stratégie de vérification pour chaque classe de la typologie, lorsque les transactions sont imbriquées et réparties.

4.2.1 Contraintes d'intégrité dans les multibases

Un système multibase supporte des opérations sur plusieurs bases de données composantes, chacune gérée par un SGBD composant. Différentes architecture et différents niveau d'intégration sont possibles entre les composants, correspondant à différents niveaux de services globaux. [54] classe les systèmes multibases en deux grandes catégories : les systèmes fédérés et les systèmes non-fédérés. Dans le premier cas, les composants sont autonomes et le système supporte aussi bien des transactions globales que des transactions locales à un SGBD composant. Les transactions locales sont gérées et contrôlées de manière autonome par les composants. Dans le second cas, la base de données apparaît aux utilisateurs comme une seule base de données répartie sur plusieurs sites. Il n'existe qu'un seul niveau de gestion et de contrôle, toutes les transactions sont globales, même si elle n'accèdent qu'à un seul site.

D'autres points de différence entre systèmes fédérés et non-fédérés existent et ont été décrits dans [54]. Cependant, seule la distinction entre transaction globale et transaction locale influe sur notre approche. En effet, notre but est de maintenir des contraintes d'intégrité globales sur un système multibase. Ces contraintes, aussi appelées contraintes fédérées ou contraintes réparties, touchent des données situées sur plusieurs sites. Dans ce cas, il est connu que la présence de transactions locales pose un problème très difficile : contrôler une contrainte touchant plusieurs sites alors que seul un site contrôle l'exécution de la transaction locale. C'est pourquoi, dans cette thèse , nous ne considérons pas les transactions locales. En dehors de ce point, nous considérons notre approche suffisament générale pour être applicable aussi bien à des systèmes fédérés que non-fédérés.

La figure 4.2 montre l'architecture (classique) du système multibase que nous prenons en considération. Les transactions globales sont adressées au gestionnaire de transactions multibases (GTG). Ce gestionnaire fournit une interface transparente et uniforme pour l'accès aux données réparties, en se basant sur un schéma global qui décrit toutes les données et leur localisation. Nous supposons de plus que l'architecture consi-

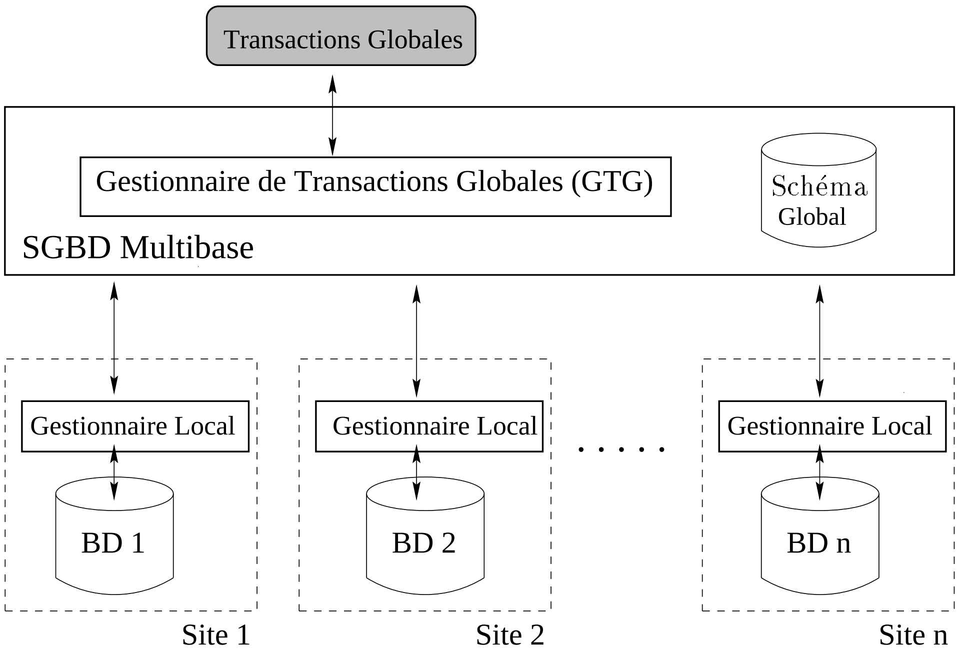

FIG. 4.2 – *Architecture du système multibase*

dérée possède les caractéristiques suivantes :

1. Toutes les transactions sont globales et sont gérées par le gestionnaire de transactions globales à l'aide d'un schéma global. Ce gestionnaire garantit notamment le contrôle de la concurrence entre transactions et entre sous-transactions d'une transaction (voir par exemple [58])

2. Comme déjà dit au chapitre 1 nous utilisons un modèle de transactions imbriquées où seules les feuilles accèdent aux objets. Chaque feuille est exécutée totalement sur un seul site et n'accède donc qu'aux données de ce site.

3. Nous ne prenons pas en compte d'éventuelles répliques d'une même donnée sur plusieurs sites.

L'affectation des sous-transactions d'une transaction aux différents sites est induit par la répartitions des données. Lorsqu'une transaction globale est initiée sur un site, ses sous-transactions sont alors lancées récursivement, jusqu'à atteindre un nœud dont le sous-arbre ne contient que des feuilles s'exécutant sur un seul site. Ce nœud est alors envoyé sur le site en question et son sous-arbre sera exécuté entièrement sur ce site. La figure 4.3 illustre ce fonctionnement dans le cas de la transaction globale T initiée sur le site S_i. Les sous-transactions T_1 et T_2 ayant des feuilles de leur sous-arbre sur plusieurs sites différents, elles sont exécutées sur le site initiateur S_i. En revanche, le nœud T_{11}, fils de T_1, a toutes les feuilles de son sous-arbre qui vont s'exécuter sur le site S_1. T_{11} est donc envoyé sur le site S_1 où s'exécutera tout son sous-arbre. On note sur la figure qu'un sous-arbre affecté à un site peut se réduire à une seule sous-transaction feuille, comme c'est le cas pour Tl_{12} et Tl_{21}.

La répartition des données et des sous-transactions sur plusieurs sites nous amène a étendre la solution présentée à la section 4.1. Le principe de base reste identique. Par une analyse des contraintes et des transactions et à l'aide du schéma global de la base, nous sommes capables de déterminer quelles sont les données impliquées par une contrainte et sur quel(s) site(s) elles sont localisées, aussi bien que la localisation des sous-transactions feuilles d'une transaction globale ainsi que l'ensemble des contraintes touchées par une sous-transaction feuille sur un site donné. Par conséquent, le contrôle de la vérification d'une contrainte C_i par le plus petit ancêtre commun $SCA(C_i)$ des feuilles touchant la contrainte n'a pas à être remis en cause. En effet, puisque que nous laissons s'exécuter la transaction en parallèle du processus de vérification, le contrôle ne peut être fait que par un nœud possédant le contrôle de l'exécution entre le moment où C_i est touchée par des transactions feuilles et le moment où un message d'abandon doit être envoyé à ces feuilles en cas de violation de C_i, soit un ancêtre commun à chaque feuille touchant C_i. Parmi ces ancêtres communs, $SCA(C_i)$ est le plus «proche» des feuilles puisqu'il est soit sur le site commun aux feuilles touchant C_i, soit sur le site initiateur de la transaction globale. Dans ce dernier cas, tous les ancêtres communs aux feuilles touchant C_i sont aussi sur le site initiateur.

En revanche la vérification effective par le processus $Check(C_i)$ de la contrainte C_i doit tenir compte des coûts de communication entre sites pour optimiser le temps de vérification. On peut distinguer deux cas :

1. La contrainte C_i est une contrainte *locale*, c'est-à-dire qu'elle implique des données stockées toutes sur le même site S_j. Dans ce cas, il est évident que la solution la plus efficace est d'effectuer $Check(C_i)$ sur le site S_j.

2. La contrainte C_i est *globale*, c'est-à-dire qu'elle implique des données situées sur plusieurs sites différents. Dans ce cas, il est nécessaire de choisir le ou les sites sur lesquel effectuer le processus de vérification de la contrainte, en essayant d'envoyer le moins de messages sur le réseau, et aussi en minimisant la taille de ces messages.

Pour effectuer une vérification efficace des contraintes globales lorsque le système multibase supporte des transactions imbriquées, il est donc nécessaire de considérer la nature des contraintes mais aussi la structure des transactions imbriquées, notamment les sites sur lesquels s'effectuent leurs sous-transactions feuilles. La figure 4.3 illustre cette problématique : les données impliquées par chaque contrainte C_1, C_2 ou C_3 sont dans le domaine délimité en pointillé, le nom de la contrainte figurant à l'intérieur de la contrainte. Lorsqu'une sous-transaction feuille touche une partie de ce domaine sur un site donné, elle est incluse dans le domaine. Par exemple, C_1 implique des données sur les trois site S_1, S_2 et S_3 et est touchée par Tl_{1112} sur S_1, Tl_{12} sur S_2 et Tl_{222} sur S_3. C_2 est une contrainte locale à S_1 et est touchée par Tl_{112} et Tl_{1111} sur ce site. Sur cet exemple on peut voir que la définition d'une stratégie de vérification est plus ou moins complexe selon les cas :

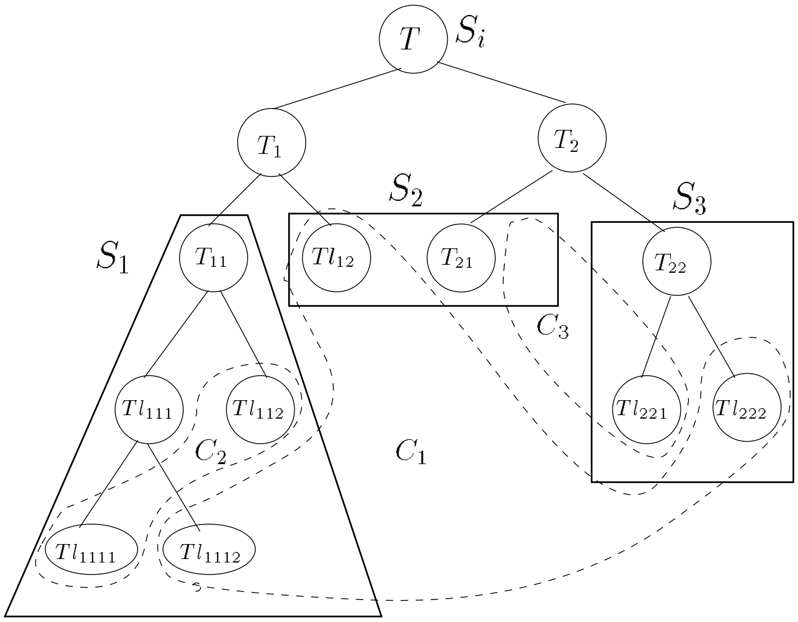

FIG. 4.3 – *Contraintes globales touchées par une transaction imbriquée.*

- Pour la contrainte C_2, il est clair que le processus de vérification peut se faire entièrement sur le site S_1, aucune donnée d'un autre site n'étant nécessaire à cette verification.
- Pour la contrainte C_3, bien que la seule feuille qui touche la contrainte soit exécutée sur le site S_3, il est probable que le processus soit effectué principalement sur le site S_2 si C_3 est une contrainte universelle. En effet, l'évaluation de C_3 nécessite

de comparer les données modifiées ou insérées sur S_3 par Tl_{221} avec les données qui leur sont associées au travers de la contrainte sur S_2. Evaluer la contrainte sur S_2 nécessite donc uniquement le transfert de ces données modifiées sur S_3 vers S_2, alors que l'évaluation de la contrainte sur S_3 nécessite le transfert de toutes les données de S_2 impliquées par la contrainte, y compris celles qui ne sont pas affectées directement ou indirectement par la transaction.

- Pour la contrainte C_1, la situation est encore plus complexe. L'évaluation de la contrainte peut être faite, selon la nature de la contrainte et les actions effectuées par Tl_{1112}, Tl_{12} et Tl_{222}, sur S_1, S_2, S_3 ou n'importe quelle combinaison de ces sites.

La définition d'une méthode optimisée de vérification de contraintes globales dans un environnement multibase supportant des transactions imbriquées apparaît être un challenge difficile par la diversité des situations à prendre en compte. Avant d'envisager une solution générale à ce problème, il est donc nécessaire d'en dégager les points facilement traitables et ceux qui nécessitent des développements ultérieurs plus poussés. Il est évident qu'une contrainte locale doit être vérifiée sur l'unique site qui contient les données qu'elle implique. Pour une contrainte globale, il faut étudier le(s) meilleur(s) endroit(s) où effectuer sa vérification, ce qui dépend du type de la contrainte. Dans la suite de cette section, nous essayons de dégager différentes catégories de contraintes globales et proposons, pour chaque catégorie, une stratégie de vérification adéquate. Dans un premier temps, nous traitons les cas les plus simples où une contrainte globale peut se réécrire entièrement à partir de contraintes locales, puis nous abordons le cas plus général où une contrainte lie des objets sur plusieurs sites.

4.2.2 Contraintes globales vérifiables localement

Dans cette sous-section, nous montrons comment évaluer des contraintes globales qui peuvent être évaluées entièrement à partir de processus locaux, c'est-à-dire des processus qui n'accèdent chacun qu'aux données d'un seul site. Nous étudions dans un premier temps le cas où les conditions à vérifier localement sont reliées par une conjonction, puis nous étudions le cas où elles sont liées par une disjonction.

4.2.2.1 Contraintes se ramenant à une conjonction de contraintes locales

Les contraintes globales se ramenant à une conjonction de contraintes locales sont celles qui peuvent être mises sous la forme

$$C_1 \wedge C_2 \ldots \wedge C_n$$

telle que, pour tout i, C_i est une contrainte locale à la base de données du site S_i.

Si les C_i sont complétement indépendantes entre elles (i.e si elles touchent des classes différentes et non liées entre elles), le remplacement de C par l'ensemble des C_i prises independamment doit être envisagé dès la phase de conception des contraintes. Par exemple, si C est de la forme $\forall o_1 \in Cl_1, \forall o_2 \in Cl_2, F(o_1, o_2)$, avec la classe Cl_1 (resp. Cl_2) sur le site S_1 (resp. S_2) et que $F(o_1, o_2)$ peut se réécrire en $F_1(o_1) \wedge F_2(o_2)$, alors on peut directement déclarer deux contraintes indépendantes C_1 : $(\forall o_1 \in Cl_1, F_1(o_1))$ sur le site S_1 et C_2 : $(\forall o_2 \in Cl_2, F_2(o_2))$ sur le site S_2.

Dans certains cas, la décomposition de C doit tenir compte de la fragmentation des données. Par exemple, si la contrainte C est de la forme $\forall o \in Cl, F(o)$ et que la classe Cl est fragmentée sur les sites $S_1, \ldots, S_n$. Si on appelle Cl_{S_i} le fragment de Cl localisé sur S_i, alors la contrainte C se ramène à $(\forall o_1 \in Cl_{S_1}, F(o_1)) \wedge (\forall o_2 \in Cl_{S_2}, F(o_2)) \ldots \wedge (\forall o_n \in Cl_{S_n}, F(o_n))$

Stratégie de vérification : La stratégie à suivre pour des contraintes de ce type est simple. A la compilation de la contrainte C, celle-ci est réécrite en $C_1, \ldots, C_n$. La satisfaction de C étant équivalente à la satisfaction de l'ensemble des C_i, il suffit pour maintenir C de maintenir chaque C_i, ce qui peut être fait sur le site correspondant.

4.2.2.2 Contraintes se ramenant à une disjonction de contraintes locales

Les contraintes globales se ramenant à une disjonction de contraintes locales sont celles qui peuvent être mises sous la forme :

$$C_1 \vee C_2 \ldots \vee C_n$$

telle que C_i est une contrainte locale à la base de données du site S_i.

De telles contraintes sont par exemple les contraintes existentielles sur une classe fragmentée. Ainsi, une contrainte de la forme $\exists o \in Cl, F(o)$ avec Cl fragmentée en $Cl_1, Cl_2, \ldots Cl_n$ respectivement sur les sites $S_1, S_2, \ldots S_n$ se ramène à $(\exists o_1 \in Cl_1, F(o_1)) \vee (\exists o_2 \in Cl_2, F(o_2)) \ldots \vee (\exists o_n \in Cl_n, F(o_n))$

Stratégie de vérification : Plusieurs stratégies peuvent être envisagées pour vérifier une disjonction de contraintes locales de la forme $C \equiv C_1 \vee C_2 \ldots \vee C_n$. Pour cela, il faut considérer, parmi les contraintes locales C_i, lesquelles sont touchées par les feuilles de la transaction imbriquée. Pour simplifier, on suppose que ce sont les contraintes locales $C_1, C_2 \cdots C_k$, avec $k \leq n$ (en d'autres termes, nous supposons que les contraintes $C_{k+1} \cdots C_n$ ne sont pas touchées par la transaction imbriquée).

Une première méthode pour effectuer la vérification de C est la suivante : effectuer la vérification des contraintes locales $C_{k+1} \cdots C_n$ en parallèle de l'exécution de la transaction (donc sans attendre la fin de l'exécution des transactions feuilles touchant la contrainte). Comme ces contraintes locales ne sont pas affectées par la transaction imbriquée, la satisfaction de l'une d'entre elle suffit à satisfaire la contrainte globale, quels que soient les effets de la transaction. Dans ce cas, il ne sera pas nécessaire de vérifier les contraintes locales $C_1 \cdots C_k$ après l'exécution des transactions feuilles. Si aucune des contraintes locales $C_{k+1} \cdots C_n$ n'est satisfaite, alors $SCA(C)$ doit attendre la fin des transactions feuilles qui touchent C, puis déclencher les processus de vérification des contraintes locales $C_{k+1} \cdots C_n$ jusqu'à en trouver qui soit satisfaite. Dans le cas contraire, le processus d'abandon partiel décrit à la section 4.1 est lancé. Cette méthode nécessite de modifier le comportement de $SCA(C)$ par rapport à la solution présentée à la section 4.1 : celui-ci n'attend plus la fin des transactions feuilles concernées par la contrainte pour lancer la vérification. Elle a pour inconvénient d'avoir à accéder à des sites $(S_{k+1}, \ldots, S_n)$ que la transaction ne touche pas elle-même. L'avantage est qu'elle augmente le parallélisme et peut éviter d'avoir à attendre la fin de l'exécution des transactions feuilles pour commencer le processus de vérification de la contrainte globale.

Une autre méthode possible est de procéder à l'inverse et attendre la fin des transactions qui touchent C pour faire la vérification. Cette méthode n'implique donc pas de modifier le comportement de $SCA(C)$ par rapport aux principes présentés en la section 4.1. La vérification est lancée dans un premier temps sur les sites touchés $(S_1, \ldots, S_k)$, puis, si aucune contrainte locale n'est satisfaite, $SCA(C)$ lance la vérification sur les sites impliqués mais non touchés $(S_{k+1}, \ldots, S_n)$ jusqu'à trouver une C_j qui soit satisfaite. Cette méthode a l'avantage de n'accéder à des sites non concernés par la transaction que si c'est nécessaire, mais réduit le parallélisme.

Dans le cas où les transactions feuilles d'un seul site touchent la contrainte C, (par exemple sur le site S_j) une troisième méthode est de vérifier C_j *avant* l'exécution de la transaction feuille sur S_j. En effet, si C_j n'est pas satisfaite, alors on est sûr qu'il existe un site S_i (différent de S_j), *non concerné par la transaction* sur lequel C_i est satisfaite, puisque C, qui est une disjonction de contraintes locales, est elle-même supposée satisfaite en début de transaction. Par conséquent, on a la certitude que C sera satisfaite quels que soient les effets de la transaction feuille sur S_j. Si C_j est satisfaite au début de la transaction, alors on se ramène à l'une des deux méthodes précédentes. Cette méthode peut se généraliser à des cas où C peut être touchée par des transactions feuilles sur différents sites, mais n'est rentable que si le nombre de sites impliqués par la contrainte et touchés par la transaction est très petit par rapport au nombre total de sites impliqués par la contrainte.

Notons enfin que ces trois méthodes peuvent être optimisées si le système maintient l'information sur la satisfaction des différentes composantes de la disjonction. Par exemple, si $SCA(C)$ sait qu'il existe un site S_j impliqué par C mais non touché par la transaction pour lequel C_j est satisfaite avant la transaction, alors il sait que C ne pourra pas être violée par la transaction imbriquée et qu'aucune vérification n'est nécessaire. Dans le cas contraire, la vérification ne sera nécessaire que sur les sites touchés par la transaction imbriquée.

4.2.3 Contraintes globales non vérifiables localement

Une contrainte dont la vérification ne peut être effectuée entièrement par des processus locaux indépendants va contenir des prédicats et des quantificateurs nécessitant une vérification simultanée sur plusieurs sites. Une telle contrainte peut avoir une complexité arbitraire. Elle peut être définie à partir de n'importe quel type de prédicat, de quantificateur et de connecteur, ce qui rend une stratégie de vérification générale impossible à établir. Par exemple, si la contrainte contient un prédicat dont une variable est quantifiée existentiellement sur une collection répartie sur plusieurs sites, l'évaluation de la contrainte va devoir accéder éventuellement à des objets non affectés par la transaction considérée. Ce problème se ramène donc à un problème d'évaluation de requête répartie sur une multibase à objet, problème réputé pour n'avoir pas de solution efficace générale et qui dépasse le cadre de cet article. De même, si la contrainte contient un prédicat formé à partir d'un agrégat global (agrégat calculable sur des données provenant de plusieurs sites), une vérification efficace de la contrainte soulève le problème de calcul efficace d'agrégats répartis, dont la solution est encore un sujet de recherche très actif dans le domaine des entrepôts de données.

Nous nous restreignons à une classe importante de contraintes, que nous appelons *contrainte globale conjonctive.* Cette classe est, à notre connaissance la seule dont la vérification ait été abordée dans la littérature. Une contrainte de cette classe peut se mettre sous la forme suivante:

$$\begin{aligned}\forall(o_1 \in Cl_1, o_2 \in Cl_2, \cdots, o_n \in Cl_n) F_1(\vec{x}_1) \wedge F_2(\vec{x}_2) \ldots \wedge F_j(\vec{x}_j) \\ \Rightarrow F_{j+1}(\vec{x}_{j+1}) \wedge F_{j+2}(\vec{x}_{j+2}) \ldots \wedge F_m(\vec{x}_m)\end{aligned}$$

où F_i est un prédicat quelconque et $\vec{x}_i$ un sous-ensemble quelconque des variables $o_1, o_2, \cdots, o_n$.

Ce type de contraintes apparaît généralement quand il existe des chaînes de relations entre objets de différentes BDs. C'est le cas dans la figure 4.4 où il existe une liaison entre la classe *Voiture* sur le site S_1 et la classe *Personne* sur le site S_2, au moyen de la référence *propriétaire*. Nous pouvons définir sur cette multibase une contrainte globale universelle $C_{ferrari}$ telle que:

$$\forall(v \in Voiture, p \in Personne), (marque(v) = Ferrari \wedge propriétaire(v) = p) \Rightarrow âge(p) \geq 33$$

La contrainte $C_{ferrari}$ établit que le *Propriétaire* de toute *Voiture* qui est une *Ferrari* doit avoir au moins 33 ans. Comme on l'a vu en chapitre 3, la principale optimisation

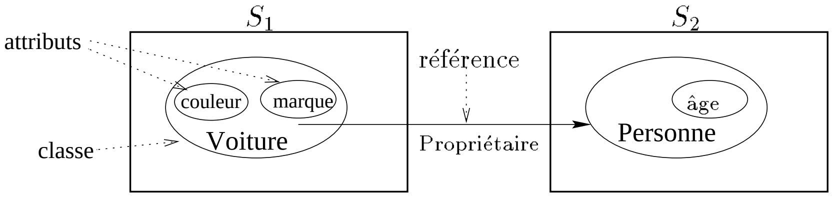

FIG. 4.4 – *Liaison du site S_1 au site S_2 par la réference propriétaire*

pour des contraintes universelles est de collecter à l'exécution les objets modifiés par une transaction et de ne vérifier la contrainte que par rapport à ces objets (et ceux qui leur sont liés par la contrainte). Dans le cas réparti, il est nécessaire, de plus, de minimiser le transfert des objets sur lesquels la contrainte est à vérifier. Pour cela, nous procédons à une analyse de chaque contrainte, dans un double but.

Le premier but s'inspire de [32]. On détermine, pour les sites où c'est possible, un prédicat localement évaluable sur ce site, dont la satisfaction par rapport aux objets modifiés par la transaction est suffisante pour garantir la satisfaction de la contrainte globale. Ce prédicat sera désigné par PL_i, dans la suite de cette section. Pour l'exemple de la figure 4.4, PL_1 correspond à $marque(v) \neq Ferrari$ et PL_2 correspond à $âge(p) \geq 33$.

Le deuxième but est basé sur la méthode proposée dans [31]. Il s'agit de découper une contrainte globale composée de variables quantifiées universellement sur une conjonction de prédicats au moyen de la définition de *prédicats intersites*. Ceci permet d'obtenir une sous-contrainte pour chaque base de données, qui décrit la responsabilité de cette base de données par rapport à la satisfaction de la contrainte globale, plus une contrainte particulière qui met en relation tous les prédicats intersites avec la contrainte à vérifier. Cette dernière conjonction est placée sur un seul site, qui sera le site sur lequel la vérification finale de la contrainte aura lieu. Il existe autant de décompositions d'une contrainte globale universelle qu'il existe de sites impliqués par cette contrainte. Pour la contrainte $C_{ferrari}$ il y aura donc deux découpages possibles:

1. Si on vérifie $C_{ferrari}$ sur le site S_1, par rapport aux mises à jour faites sur le site S_2, celle-ci est réécrit de la façon suivante:

   Sur le site S_1:

   $\forall(v \in Voiture, p \in Personne\_moins\_33), marque(v) = Ferrari \Rightarrow propriétaire(v) \neq p$

   Sur le site S_2: $\forall(p \in Personne), âge(p) < 33 \Leftrightarrow p \in Personne\_moins\_33$

   $Personne\_moins\_33$ est l'ensemble associé au prédicat intersite. Il contient les identificateurs des objets de la classe $Personne$ touchés par la transaction qui ne satisfont pas le prédicat local PL_2.

2. Si on vérifie $C_{ferrari}$ sur le site S_2, par rapport aux mises à jour faites sur le site S_1, celle-ci est reécrit de la façon suivante:

   Sur le site S_1:

   $$\forall(v \in Voiture, p \in Personne\_a\_ferrari),$$
   $$(marque(v) = Ferrari \wedge propriétaire(v) = p) \Leftrightarrow p \in Personne\_a\_ferrari$$

   Sur le site S_2: $\forall(p \in Personne), p \in Personne\_a\_ferrari \Rightarrow âge(p) \geq 33$

   Dans ce découpage, $Personne\_a\_ferrari$ est l'ensemble associé au prédicat intersite. Il contient les identificateurs des objets de la classe $Personne$ touchés par la transaction qui ne satisfont pas le prédicat local PL_1, c'est-à-dire les identificateurs des objets de la classe $Personne$ qui sont *propriétaires* d'une $Ferrari$.

Stratégie de vérification : Comme stratégie de vérification de cette classe de contraintes, nous proposons de garder toutes les combinaisons de découpage pour utiliser au moment d'effectuer la vérification, celle qui minimise l'échange de données. Ce choix s'effectue dynamiquement en se basant sur les sites sur lesquels s'exécutent les transactions qui touchent la contrainte.

La vérification d'une contrainte globale universelle conjonctive C est elle-même une transaction imbriquée. Le $SCA(C)$ déclenche sur son site le processus $Check(C)$ chargé de la vérification de C. Ce processus sera la racine d'une transaction imbriquée répartie qui aura une fille pour chaque site S_i où il existe des sous-transactions feuilles qui touchent C. Chaque fille, appelée $Check(CMJ_i)$, est en charge de la vérification de C par rapport aux mises à jour dans le site S_i. Les processus $Check(CMJ_i)$ auront le comportement suivant:

1. D'abord vérifier PL_i s'il existe. Si PL_i est satisfait, il ne sera pas nécessaire de réaliser d'autres actions, puisque C est satisfaite par rapport aux mises à jour sur

le site S_i. C'est le cas d'une transaction feuille Tl qui change l'âge d'une $Personne$ p dans la multibase de la figure 4.4. Pour vérifier la contrainte globale $C_{ferrari}$ on vérifie PL_2. Si le nouvel *âge(p)* est au moins de 33, il n'y a plus rien à faire que p soit propriétaire ou non d'une *Ferrari*.

2. Si PL_i n'est pas satisfait ou n'existe pas, on utilise les prédicats intersites pour identifier les objets sur S_i nécessaires pour faire la vérification sur les autres sites impliqués par la contrainte globale. Dans ce cas, on enverra au site sur lequel on va vérifier C les mises à jour effectuées dans S_i. Ensuite, on commencera le processus de vérification proprement dit.

Il faut bien voir que cette méthode pour la vérification déclenche une transaction imbriquée $Check(C)$ répartie sur plusieurs sites : les sites contenant des transactions feuilles qui touchent C et ceux où doit se vérifier C. Pour illustrer ceci, reprenons la contrainte $C_{ferrari}$ définie sur les données des sites S_1 et S_2 de la figure 4.4. Pour la vérification de $C_{ferrari}$ on va générer une des trois transactions suivantes, selon les sites où furent exécutées les transactions feuilles qui touchent $C_{ferrari}$.

Premier Cas: Les transactions feuilles qui touchent $C_{ferrari}$ s'exécutent toutes sur le site S_1. La structure de la transaction $Check(C_{ferrari})$ corresponde au sous-arbre à gauche dans la figure 4.5. Dans ce cas, $Check(C_{ferrari})$ a une seule fille $Check(CMJ_1)$ qui sera en charge de la vérification de $C_{ferrari}$ par rapport aux mises à jour dans le site S_1. Pour cela, $Check(CMJ_1)$ aura deux sous-transactions qui s'exécuteront en séquence (indiqué par le symbole ;). La première sous-transaction, $Check(PL_1)$, vérifie le prédicat local PL_1 suffisant pour garantir la satisfaction de C. Si PL_1 n'est pas vérifié, c'est à dire si les transactions feuilles donnent comme nouvelle valeur *Ferrari* à la $marque$ d'une $Voiture$, ou si elles affectent un nouveau *propriétaire* à une $Ferrari$, alors il faut construire l'ensemble $Intersite_1$ contenant les identificateurs p_j des propriétaires de ces voitures. Cet ensemble va être envoyé au site S_2 où $Check(CL_2)$ va tester si tout propriétaire p_j d'$Intersite_1$ a un âge supérieur ou égal à 33.

Deuxième Cas: Les transactions feuilles qui touchent $C_{ferrari}$ s'exécutent toutes sur le site S_2. Ce cas est le symétrique du précédent. Ici les mises à jour concernent l'attribut *âge(p)* des objets de la classe $Personne$. S'il y a des mises à jour qui donnent un *âge* inférieur à 33 à une $Personne$, il faut construire l'ensemble $Intersite_2$ contenant les identificateurs p_j de ces personnes. Cet ensemble va être envoyé au site S_1 où l'on va tester si toute personne de cet ensemble n'est pas propriétaire d'une

$Ferrari$. La structure de la transaction $Check(C_{ferrari})$ corresponde au sous-arbre à droite dans dans la figure 4.5.

Troisième Cas: Les transactions feuilles qui touchent $C_{ferrari}$ s'exécutent sur les sites S_1 et S_2. Une sous-transaction feuille s'exécutant complètement sur un seul site, il existe des transactions feuilles qui touchent la classe *Voiture* et des transactions feuilles qui touchent la classe *Personne*. La figure 4.5 montre la structure de la transaction $Check(C_{ferrari})$ qui aura deux sous-transactions, $Check(CMJ_1)$ et $Check(CMJ_2)$, qui s'exécuteront en parallèle (indiqué par le symbole $\|$). $Check(CMJ_i)$ vérifie la contrainte C par rapport à toutes les feuilles qui s'exécutent sur le site S_i. Les processus $Check(CMJ_1)$ et $Check(CMJ_2)$ utilisent des ensembles de données différents $Intersite_2$ et $Intersite_1$ respectivement, ce qui permet la parallélisation. Cette parallélisation peut amener à vérifier en double la contrainte sur les mêmes données. Considérons que dans l'exemple de la figure 4.4 une transaction imbriquée mette à jour la marque d'une voiture et l'âge de son propriétaire. Dans ce cas, les ensembles $Intersite_i$ auront des éléments en commun et par conséquent la contrainte $C_{ferrari}$ sera verifiée deux fois par rapport aux mêmes données. Une alternative peut être d'effectuer les deux processus séquentiellement pour supprimer les doublons dans le deuxième ensemble traité. Ceci sera moins efficace que la vérification en parallèle si $Intersite_1$ et $Intersite_2$ sont disjoints ou ont peu d'éléments communs.

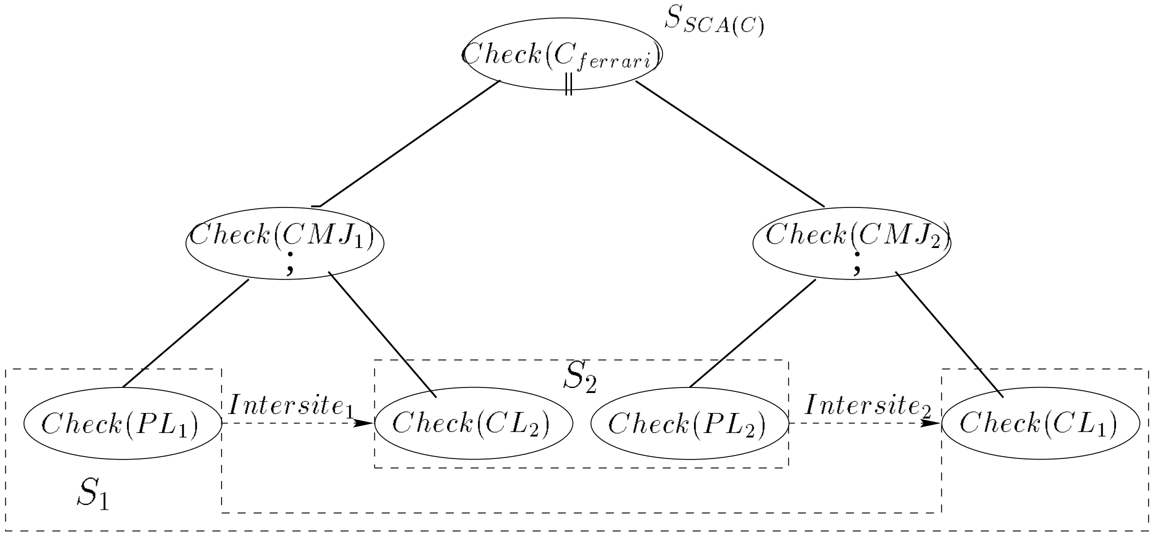

FIG. 4.5 – *Contrainte globale touchée par des transactions dans deux sites*

Notre recherche sur la vérification de contraintes d'intégrité pour des transactions imbriquées dans les systèmes multibase, a donné lieu à la rédaction de trois articles. Le premier [25] a été presenté à la conférence BDA'2000, et le second [19] à la conférence PANEL'2000. Une version revue a été pris à COOPIS'2001 [18].

Chapitre 5

Implémentation et application

Dans le but de valider la méthode que nous avons proposée au chapitre précédent, nous avons mise en œuvre un prototype d'un système supportant les transactions imbriquées.Notre implémentation (5.1) comprendra la méthode proposée dans le chapitre 4 de cette thèse, pour la vérification des contraintes d'intégrité globales dans un envirement centralisé. Les résultats de l'analyse syntaxique, et donc l'ensemble des contraintes que chaque sous-transactions feuille risque de violer, sont fournis dans un premier temps par simulation.

Nous présentons aussi un exemple d'application où notre approche pour la vérification de contraintes d'intégrité peut s'appliquer dans le domaine du commerce électronique (5.2).

5.1 Un prototype d'un gestionnaire de transactions imbriquées

Dans cette section nous présentons un prototype d'un gestionnaire, pour le modèle de transactions imbriquées (MT), où a été intégré au contrôle d'exécution des transactions, l'algorithme pour la vérification des contraintes d'intégrité, dans un environnement centralisé proposé dans le chapitre 4. Le développement a été réalisé sur un ordinateur PC Pentium III 800 Mhz, de 256 MB de Ram, sous Windows NT. Pour le développement a été utilisé le langage Java, JDK 1.2.2, étant données les facilités qu'offre la génération de *threads*[1] et leur portabilité sur d'autres plateformes. Notre effort a été centré sur le développement du contrôle d'exécution des transactions imbriquées. L'objectif qui est poursuivi avec ce développement, est de réaliser l'évaluation de l'algorithme proposé pour la vérification des contraintes d'intégrité.

1. Sous-processus légers partageant un méme espace d'exécution

La couche sous-jacente qui correspond au traitement de données a été simulée. Il n'y a pas de base de données définie et les transactions feuilles simulent l'accès aux données.

5.1.1 Contrôle d'exécution d'une transaction imbriquée

Le MT a été développé comme un serveur itératif, c'est-à-dire, qu'il supporte l'exécution d'un arbre à la fois. Pour l'exécution d'une transaction imbriquée, l'arbre sera parcouru par niveaux, en se basant dans la création de *threads* d'exécution de la manière suivante :

1. Si la racine est un opérateur parallèle, un *thread* est généré pour chaque fille. Ces *threads* se charge de l'exécution de l'arbre dont la racine est la fille correspondante et les différentes branches sont exécutées en même temps.

2. Si la racine est un opérateur séquentiel, alors des *threads* seront crées dans l'ordre, un pour chaque transaction fille, les prenant de gauche à droite en attendant la réponse de celui qui sera en train d'exécuter avant de commencer la suivante.

3. Si la racine est un opérateur alternatif, alors des *threads* seront crées dans l'ordre, un pour chaque transaction fille, les prenant de gauche à droite et en attendant la réponse de celui qui sera en train d'exécuter avant de commencer la suivante. Lorsque la première réponse de validation sera obtenue, les filles restantes ne seront pas initiées.

4. Si la racine de l'arbre est une transaction feuille, un message de sommation d'exécution est envoyé à la couche sous-jacente, qui sera appelée gestionnaire de données, et le *thread* demeure en attente jusqu'à recevoir la décision prise par la feuille : a validé ou a abandonné.

5. Le *thread* associé à une feuille envoie la décision prise, à son père tout comme aux transactions chargées de vérifier les contraintes que elle touche.

6. Lorsque la transaction $SCA(C_i)$ responsable de la contrainte C_i reçoit la réponse de toutes les feuilles qui touchent C_i, celle-ci commence un *thread* d'exécution qui correspond à la vérification de C_i. La génération automatique du code associé au processus de vérification de contraintes $Check(C_i)$ n'a pas été prise en considération dans cette thèse, en conséquence ces processus sont simulés. Le temps d'exécution est une variable aléatoire uniformément distribuée parmis les valeurs 1 à 5 secondes. Le résultat du contrôle d'une contrainte, c'est-à-dire satisfait ou

non, est simulé par une distribution de probabilité qui, selon les valeurs initiales, donnera différents scénarios d'expérimentation, comme il sera expliqué plus avant. Les valeurs de cette fonction de probabilité sont assignées comme des paramètres au moment d'initier le prototype du gestionnaire.

7. Dans le cas où la contrainte C_i n'est pas satisfaite, le *thread* correspondant à $SCA(C_i)$ envoie des messages vers le bas, jusqu'à atteindre le *thread* correspondant au plus petit ancêtre actif de chaque transaction feuille qui ait touché C_i. Ces transactions sont terminées abruptement (abandonnées) en réalisant une nouvelle vérification des contraintes auxquelles donnerait lieu ces abandons.

8. Chaque *thread* correspondant à $SCA(C_i)$ avant de prendre sa propre décision d'abandonner ou valider, vérifie la table de contraintes pour constater si la vérification de C_i doit être faite.

9. La décision de chaque transaction mère (de valider ou d'abandonner) est conséquence de la structure de l'arbre et elle est prise par le mécanisme de contrôle d'exécution. Celui-ci prend en considération le mode d'exécution des filles, la décision de chacune des filles, le type de transaction et le résultat des processus de vérification des contraintes.

10. Chaque *thread* généré applique récursivement les points 1 à 9 pour tous les nœuds de son sous-arbre.

Le processus décrit, montre la fonctionnalité du prototype MT développé. Cependant, pour mener à bien l'étude du comportement de l'algorithme pour la vérification de contraintes, il est nécessaire de disposer d'autres deux fonctionnalités : La création d'arbres qui devrait constituer une interface avec les utilisateurs, et le traitement de données et des opérations, qui doit correspondre à un SMBD. Ces deux fonctions ont été fournies dans ce prototype par simulation, en deux modules que nous appelons Générateur d'arbres et Gestionnaire de données.

5.1.2 Générateur d'arbres

Dans ce module les arbres de transactions qui serviront comme données de preuve pour l'étude que nous réalisons sont générés. Les arbres ont été générés en supposant qu'une transaction ait été développée avec quelque objectif spécifique, c'est à dire, pour réaliser un travail. Les paramètres considérés dans le modèle de simulation pour la génération d'arbres sont les suivants :

- La transaction racine est toujours obligatoire. Pendant la construction d'un arbre, on garantit pour le moins que l'une de ses branches soit constituée par des sous-transactions obligatoires.
- La probabilité qu'une sous-transaction soit obligatoire est fixée à 0,80, celle qu'elle soit optionnelle est donc fixée à 0,20.
- La profondeur des arbres est une variable aléatoire distribuée uniformément entre les valeurs 1 et 4. En effet, les applications distribuées qui sont modélisées par de transactions hiérarchiques, ne sont pas très profondes, car elles sont structurées en base aux nœuds qu'elles visitent [26, 9, 59, 58].
- La quantité de nœuds de l'arbre est associée à sa profondeur, plus l'arbre est profond, plus grande sera la quantité de nœuds.
- La quantité de filles pour chaque transaction mère est uniformément distribuée entre 2 et 5 filles par nœud.
- Le mode d'exécution (en parallèle, séquentiel ou alternatif) des filles d'une transaction est uniformément distribuée.
- On considère comme un ensemble limité de contraintes globales dans le système dont la cardinalité est définie en se fondant sur une variable aléatoire uniformément distribuée entre 10 et 20 contraintes.
- L'ensemble de contraintes qui touche chaque feuille dénotée comme $C(T_j)$, est modélisé à travers une distribution uniforme pour chaque élément, sans reprise. La cardinalité du sous-ensemble $C(T_j)$ est déterminée à travers une fonction de probabilité qui donne plus de poids aux cardinalités mineures.

5.1.2.1 Le gestionnaire de données

Le gestionnaire de données reçoit du MT la demande d'exécution d'une transaction feuille et retourne la décision (abandon ou validation) prise par cete feuille. Sans avoir défini une base de données, on simule l'exécution des feuilles. Les paramètres de la simulation sont les suivants :

- Le gestionnaire de données supporte l'exécution de multiples feuilles à la fois. Pour chaque feuille est généré un *thread*.

– La durée ou temps d'exécution de chaque feuille est une variable aléatoire uniformément distribuée entre 1 et 20 secondes. Avec cette ample marge de temps d'exécution possibles, nous tenons compte de la possibilité d'exécution des opérations simples (lecture et/ou écriture) ou complexes, étant données les caractéristiques de l'ordinateur sur lequel a été réalisé le développement.

– La décision prise pour chaque transaction feuille, de valider ou d'abandonner, est modélisée par l'intermédiaire d'une fonction de probabilité où la décision de valider est supposée avoir une probabilité de 0,90 alors qu'une décision d'abandon possède seulement 0,10.

5.1.3 Conception des expérimentations

Considérons deux scénarios différents pour le système sur lequel nous travaillons : l'un pessimiste et l'autre optimiste.

1. Le premier scénario correspond à une situation où les transactions ne sont pas bien écrite sémantiquement ou n'ont pas été prouvées exhaustivement, ce qui conduirait à violer fréquemment les contraintes d'intégrité définies sur les données. C'est pour cela que nous l'appelons pessimiste, car nous considérons un haut indice de violation des contraintes pendant l'exécution d'une transaction. Pour ce scénario pessimiste, est assignée une probabilité de 0,5 à chaque résultat possible du processus $Check(C_i)$. Ceci veut dire qu'on suppose qu'il est également possible qu'une contrainte soit satisfaite ou non.

2. Le second scénario, que nous appellerons optimiste, c'est celui où il est peu probable qu'une transaction viole les contraintes d'intégrité. Dans ce cas nous serions en train de modéliser un système stable et dépuré où les transactions ont été prouvées quasi exhaustivement et une violation peut surgir seulement lorsque se présentent certains états non prévus de la base de données. Pour ce scénario optimiste nous considérons qu'avec une probabilité de 0,90 le processus $Check(C_i)$ atteindra comme résultat que C_i se satisfait lui même et avec une probabilité de 0,20, le résultat sera que C_i est violée.

Pour étudier le comportement de l'algorithme dans chacun de ces deux scénarios, des expériences nous permettent d'obtenir de l'information statistique à propos des résultats de l'exécution d'un ensemble représentatif de transactions imbriquées dans le prototype de MT développé. L'information qu'il faut obtenir est :

- La proportion des transactions globales qui valident et par conséquent, la proportion des transactions qui abandonnent.
- la proportion des transactions qui arrivent à valider quoiqu'elles aient violé pendant leur exécution au moins une contrainte d'intégrité. Cette expérience nous dit le pourcentage de transactions sauvées par notre méthode, en comparaison à l'application d'une méthode classique qui vérifie les contraintes à la fin de la transaction globale.
- la proportion des transactions globales où s'est réalisée au moins une re-vérification de contraintes. Ceci nous permet de faire inférences pour savoir avec qu'elle fréquence il est nécessaire de réaliser des re-vérifications d'une contrainte, ce qui bien que soit nécessaire, constitue un aspect coûteux dans l'algorithme proposé.
- Une classification des transactions globales qui ont abandonné conformément à la proportion de la transaction qui avait été déjà exécutée au moment de l'abandon. Cette expérience nous permet de montrer le travail superflu évité par notre méthode, où est réalisée la vérification le plus tôt possible pendant l'exécution d'une transaction imbriquée. Le progrès atteint par une transaction au moment de son abandon (PTA) a été modélisé en base à la quantité de sous-transactions complètement exécutées (TE) sur la quantité totale de sous-transactions de l'arbre (TT), en accord à la formule suivante :

  $$PTA = \frac{TE}{TT}$$

  La valeur de PTA classifie la transaction par rapport à la progression atteinte au moment de l'abandon et nous permet d'obtenir un histogramme de la fréquence de chaque possible PTA.

5.1.4 Résultats obtenus dans l'évaluation

Pour évaluer le comportement de l'algorithme de vérification de contraintes d'intégrité en présence de transactions imbriquées, ont été générés deux échantillons conformés par 10.000 arbres chacune. Cette grandeur de l'échantillon nous permet garantir nos résultats avec une probabilité de 0,95 et un ϵ de 0.01 [2]. Les arbres d'un échantillon ont été exécutés dans le scénario *pessimiste* et ceux de l'autre dans *l'optimiste*.

2. $P(|\hat{p} - p|) > \epsilon) < 0,05$.

Pesimista

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0	0	0	0	0	0	2	0	0	6	0	0	0	5	9	0	27	0	0	41
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
23	2	0	3	0	0	15	0	82	0	0	41	0	38	0	155	92	0	95	370
41	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
30	0	218	11	0	32	0	0	13	1106	20	0	0	43	0	131	309	0	172	418
61	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**
135	0	42	198	0	113	0	105	0	0	108	0	4	0	0	5	0	5	75	158
81	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**	**100**
0	0	35	0	68	27	0	115	0	74	0	0	8	0	14	0	0	11	0	251

FIG. 5.1 – *Table de Fréquence du PTA dans un scénario pessimiste*

– Scénario pessimiste. ($Check(C_i)$ renvoie vrai dans 50 % des cas). Avec cette hypothèse, nous avons obtenu les résultats suivants : 4940 transactions arrivent à valider et 5060 abandonnent. Ceci donne comme résultat une proportion de 49,4 % de transactions validées et 50,6 % de transactions abandonnées. Dans les 4940 transactions validées, 1691 arrivent à valider (34 %) malgré avoir violé au moins une des contraintes qu'elles touchent pendant leur exécution. Ceci nous permet d'affirmer que dans le scénario pessimiste 34 % des transactions qui valident le peuvent grâce à notre algorithme, puisque dans d'autres circonstances celles-ci auraient été obligées d'abandonner pour violation de contraintes. En ce qui concerne le total de 10.000 transactions, notre méthode récupère 16,91 % des transactions qui arrivent à valider, malgré qu'elles aient violé quelque contrainte pendant leur exécution et par conséquent ont abandonné quelques unes de leurs sous-transactions.

La figure 5.1 montre la fréquence de chaque possible valeur du PTA dans les 5060 transactions qui ont abandonné le scénario pessimiste. Les numéros en noir représentent les valeurs du PTA et la fréquence correspondante apparaît juste audessous. On observe que le mode du PTA est de 50 % de progrès dans les transactions au moment de l'abandon, la moyenne étant de 57,03. Ceci peut s'interpréter de cette faccon : sous ce scénario pessimiste, l'application de notre algorithme est arrivé à éviter qu'il se réalise à peu prés 50 % du travail inutile. Avec n'importe quel autre mécanisme de contrôle de contraintes, inévitablement la transaction globale sera également abandonnée par violation de l'intégrité. Or, notre algorithme provoque son abandon tôt et n'attend pas la fin de la transaction. Ceci est dû à que notre algorithme garantit la vérification des contraintes le plus tôt possible, c'est à dire à la fin des dernières feuilles qui touchent la contrainte.

- Scénario optimiste. ($Check(C_i)$ renvoie vrai dans 90 % des cas). Avec cette hypothèse, nous avons obtenu les résultats suivants : 7342 transactions valident et 2658 abandonnent. Dit d'une autre faccon, une proportion de 73,42 % des transactions validées et 26,58 % des transactions abandonnées. Dans les 7342 transactions validées, 807 (11 %) le font, malgré avoir violé au moins l'une des contraintes qui touche pendant leur exécution. Ce qui signifie que le 11 % des transactions validées le doivent à algorithme de vérification de contraintes utilisé, puisque, avec l'application d'un autre mécanisme, elles auraient été abandonnées. En ce qui concerne la totalité de l'échantillon, sous le scénario optimiste notre méthode récupère le 8,00 % des transactions et arrive à que celles-ci valident, malgré avoir abandonné quelques-unes unes de leurs sous-transactions.

  La figure 5.2 montre la fréquence de chaque possible valeur du PTA dans les 2658 transactions qui ont abandonné. De ces résultats on peut déduire que 56,66 % des transactions de l'échantillon ont atteint entre 50 % et 85 % de leur progrès au moment de l'abandon. La moyenne du PTA est 70,90 %. Ceci signifie que l'application de notre algorithme a pu éviter que soit réalisé en moyenne le 30 % du travail inutile, en vérifiant tôt les contraintes et en abandonnant la transaction globale lorsque la violation de l'intégrité ne peut être évitée.

Optimista

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	13	0	0	2
21	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**
0	0	0	0	0	0	0	0	3	0	0	9	0	1	0	16	7	0	45	48
41	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**
61	0	68	5	0	30	0	0	52	244	0	0	0	0	7	0	93	0	31	145
61	**62**	**63**	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**
76	0	4	201	0	24	0	144	0	0	205	0	2	0	0	16	0	65	81	160
81	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**	**96**	**97**	**98**	**99**	**100**
0	0	1	0	7	162	0	35	0	87	0	0	5	0	21	0	0	69	0	412

FIG. 5.2 – *Table de Fréquence du PTA dans le scénario optimiste*

Pour vérifier les résultats quant à la re-vérification de transactions, on répète l'expérience 30 fois avec 10.000 transactions chaque fois, dans chaque scénario. La quantité de transactions qui réalisent au moins une re-vérification pour le scénario optimiste oscille entre un minimum de 394 et un maximum de 468, avec une moyenne de 428 dans les 30 preuves. Pour le scénario pessimiste les résultats ont varié entre 420 et 531, la moyenne étant de 491 transactions. Ces résultats toujours proches au 4 % de l'échantillon, semblent indiquer que la fréquence de re-vérification des contraintes est une variable indépendante de deux scénarios de preuve ici considérés. De plus, on peut conclure que, *relativement aux conditions d'expérimentation*, le surcoût induit par la re-vérification semble peu important (4% des cas) par rapport au gain obtenu en vérifiant les contraintes le plus tôt possible.

5.2 Transactions imbriquées et contraintes d'intégrité dans le commerce électronique

Cette section a comme but celui de montrer l'utilité de la méthode proposée, en prenant comme cas d'étude une application dans le champs du commerce électronique (CE). En premier lieu, on montre comment certaines activités propres au commerce électronique peuvent être structurées grâce au modèle de transactions imbriquées proposé dans ce travail. Pour ceci on prend comme exemple un magasin virtuel dont les ventes sont fondées sur l'inventaire de divers vendeurs associés, lesquels forment une multibase de données. Après, on donne des exemples de contraintes d'intégrité globales qui peuvent être définies sur cette multibase de données.

5.2.1 Transactions imbriquées dans des applications du commerce électronique

Besancenot et autres [6] considèrent que les transactions imbriquées sont adéquates pour modéliser des transactions en Internet, particulièrement dans le champ de CE o est utilisé pour des raisons politiques et/ou économiques le concept *d'agence intermédiaire*. Une agence c'est un composant intermédiaire qui permet d'établir une coopération sûre et crédule entre plusieurs participants pour exécuter une tâche commune [48]. Dans le contexte du commerce électronique, le rôle d'une agence est de modéliser les processus imbriqués nécessaires pour réaliser le travail souhaité par l'acheteur, le mener à bien et informer l'acheteur à fin qu'il puisse prendre les décisions qui sont de sa compétence. Ce modèle économique est à trois niveaux : vendeur - agence - acheteur. Il conduit d'une manière naturelle vers une architecture à trois niveaux de systèmes de base de données pour le commerce électronique. A chaque niveau nous pouvons trouver une ou plusieurs bases de données, plus ou moins intégrées. La base de données d'un vendeur est un groupe d'entités (biens et services) qui sont prédestinées à être intégrées au niveau le plus haut. Cette intégration est réalisée par un acteur économique indépendant : l'agence intermédiaire. L'élection d'un vendeur ou d'un autre, pour un produit donné dépend de l'optimisation de l'ensemble des processus avec lesquels travaille l'agence et les contraintes d'intégrité par elle définies, en considérant les caractéristiques observables des produits dans la base de données du vendeur. C'est pour cela que les bases de données des vendeurs doivent être conçues pour coopérer avec d'autres bases de données et des systèmes, méconnus au moment de leur conception.

Il est vrai que le modèle de transactions imbriquées ne permet pas de structurer le processus de commerce électronique dans sa totalité, puisqu'une méthode si stricte comme la méthode de contrôle de la concurrence oblige à maintenir des information en indisponibilité pour plusieurs jours, ce qui est en aucun cas acceptable et en conséquence impossible de réaliser. Cependant, l'utilisation effective du modèle de transactions imbriquées proposé dans cette thèse est possible au moins dans la phase du *traitement de commandes* du CE, comme nous le montrons dans cette section.

Dans le cadre du commerce électronique actuel, dans un espace de marché B2C (type amazon.com, cdnow.com ou tout autre magasin ou distributeur) la phase de traitement de commandes est réalisée de manière asynchrone dans le sens où l'acheteur émet un commande d'achat et celle-ci est créée par le magasin par rapport aux prévisions de l'état de l'inventaire qu'il maintient dans son catalogue. Ensuite, la commande est traitée par les vendeurs et/ou tierce personne pour matérialiser l'achat. De cette façon, existe toujours la possibilité qu'un produit ne soit plus disponible et donc la commande doit être annulée, ce qui se traduit dans tous les cas, par une insatisfaction de la part de l'acheteur envers le service offert par le magasin. En plus, le catalogue du magasin, contient les produits offerts par les vendeurs et une description de leurs utilisations. Mais la description des conditions relatives à la distribution du produit, les lois, les paiements et la garantie sont gardés dans la base de données des vendeurs. Dans le champ du commerce électronique idéal, un magasin qui fonctionne comme une agence doit proposer à ses clients seulement les produits sensés les intéresser, ainsi que les conditions annexes qui sont déterminantes dans l'élection définitive d'un produit. Pour arriver à ceci, il faut que le magasin ne copie pas l'information des vendeurs dans sa propre base de données, mais au contraire qu'elle coopère avec les bases de données des vendeurs afin que tout changement dans l'offre d'un vendeur soit immédiatement reflétée au niveau du magasin. Il faut noter que le traitement d'une commande est réparti entre les divers vendeurs associés au magasin et la vérification de l'existence est localisée à l'endroit de chaque vendeur. Le succès dépend de la transparence et de l'effectivité que représente pour l'usager le traitement de sa commande sur la multibase de données des vendeurs. Pour les biens et les services critiques ou indispensables (du point de vue de l'acheteur, il peut s'agir de n'importe quel article), le succès d'un magasin virtuel se trouve dans la minimisation de la possibilité qu'un ordre d'achat déjà accepté soit ultérieurement annulé. C'est de là que surgit la nécessité d'un magasin à réponse immédiate, c'est à dire, où un ordre d'achat génère des réquisitions vers des tiers, mais on ajoute un facteur de *confirmation immédiate* dans le sens où l'usager veut savoir à l'instant où il l'émet, si sa commande est totalement ou partiellement satisfaite pour prendre les décisions. Dans cette section, on prend comme exemple le cas d'un magasin virtuel où on assume que le commerce électronique est mené à bien sous les conditions suivantes :

- Le magasin opère comme une agence intermédiaire, qui travaille directement sur la base de données de multiples vendeurs associés.
- Le magasin offre des biens et/ou services qui sont indispensables pour l'acheteur.
- L'acheteur a besoin d'une réponse sûre et immédiate sur la commande réalisée, ne pouvant pas attendre une réponse à posteriori qui pour certains magasin peut prendre quelques jours.
- Du point de vue de l'acheteur les produits demandés peuvent être fournis par n'importe lequel des vendeurs, sans distinction.

Pour modéliser dans cet exemple la phase du *traitement de commandes*, il faut que chaque vendeur possède sa BD en ligne pour constituir une multibase de données. La phase du traitement de la commande, c'est à dire l'actualisation des inventaires et toute autre donnée en relation avec l'achat, peut être vue comme une transaction imbriquée.

De façon naturelle, le *traitement d'une commande* se divise dans une sous-transaction pour chaque produit. Ces sous-transactions sont optionnelles puisque ce qui est important c'est de disposer de la plus grande quantité des produits requis dans la commande. Chacune d'entre elles génère à son tour une sous-transaction pour chaque vendeur qui pourrait fournir le produit. Si nous considérons un ordre d'achat avec n articles, la transaction imbriquée résultante aura la structure qui est présentée dans la figure 5.3,montre la distribution du traitement d'une commande parmis plusieurs vendeurs associés à un magasin virtuel. De cette manière le modèle TI proposé dans ce travail permet de structurer la phase de traitement des commandes dans un magasin virtuel qui opère comme une agence intermédiaire.

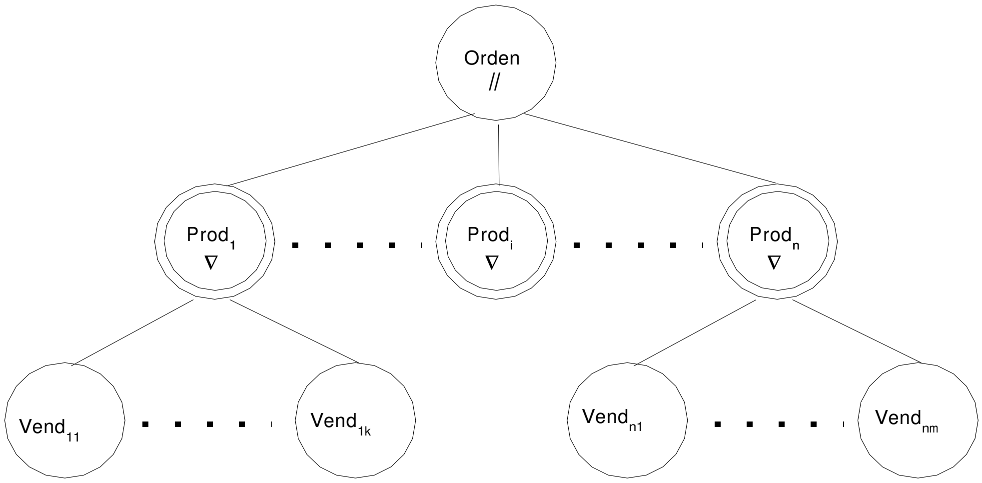

FIG. 5.3 – *Transaction pour traiter une commande en CE*

5.2.2 Contraintes d'intégrité dans une application du commerce électronique

Pour établir des contraintes d'intégrité (CI) il est nécessaire de détailler l'application qui est menée, car une contrainte refléte des propriétés du monde réel, c'est-à-dire le contexte de l'application avec celle qui est en train d'être traitée. Dans notre cas, l'application est un magasin virtuel, de style amazon.com, qui vend des produits divers : livres, CD, vidéos, articles ménagers, ordinateurs, etc.

Dans la figure 5.4 on voit le diagramme en notation UML [2] de classes de la BD associée au magasin. Seules les classes utiles pour l'exemple sont représentées. La figure 5.5 montre la répartition des données. La BD locale du magasin contient les classes *Catalogue, Acheteur, Commande, Pays et Dist-2-pays* qui permettent d'offrir divers produits à leurs clients à travers du site Web du magasin; et une BD local à chaque vendeur associé au magasin, où et maintenu le stock réel de chaque produit, et les diverses conditions de production, envoi, paiement, etc. Ceci signifie que les classes *Vendeur et Produit* sont fragmentées sur les sites de chaque vendeur associé au magasin.

Dans cette application de commerce électronique peuvent être définies les contraintes globales qu'impose le magasin à ses vendeurs associés pour réaliser ses opérations conjointement. Ci-dessous, nous présentons deux exemples.

1. Contraintes locales : En considérant la distribution des données montrée dans la figure 5.5, la contrainte suivante est locale au site du magasin:

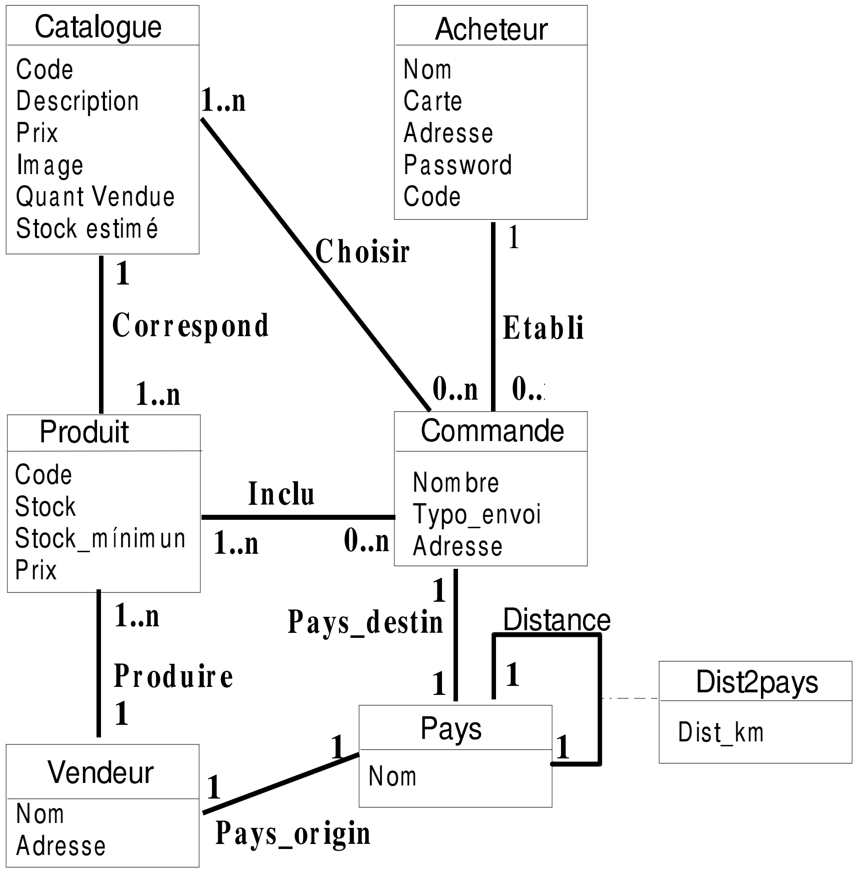

FIG. 5.4 – *Diagramme de classes de la BD qui supporte le magasin virtuel*

$$C_1 : \forall(i \in Catalogue, o \in Commande, (prix(i) \geq 500 \wedge choisir(i) = o)$$
$$\Rightarrow type\_envoi(o) = Special$$

C_1 établit que tout produit dont le prix dans le catalogue est supérieur à 500 \$ doit être transporté à travers un service de courrier spécial.

2. Contraintes globales non-évaluables localement : celles-ci correspondent aux politiques du magasin ou des lois nationales et/ou internationales qui régissent le commerce et doivent être satisfaites dans un processus d'achat/vente. En particulier nous pourrions avoir :

$$C_2 : \forall(p \in Produit, o \in Commande, v \in Vendeur, q, r \in Pays, d \in Dist2pays),$$

$$produire(p) = v \wedge inclu(p) = o \wedge pays\_origin(v) = q \wedge$$

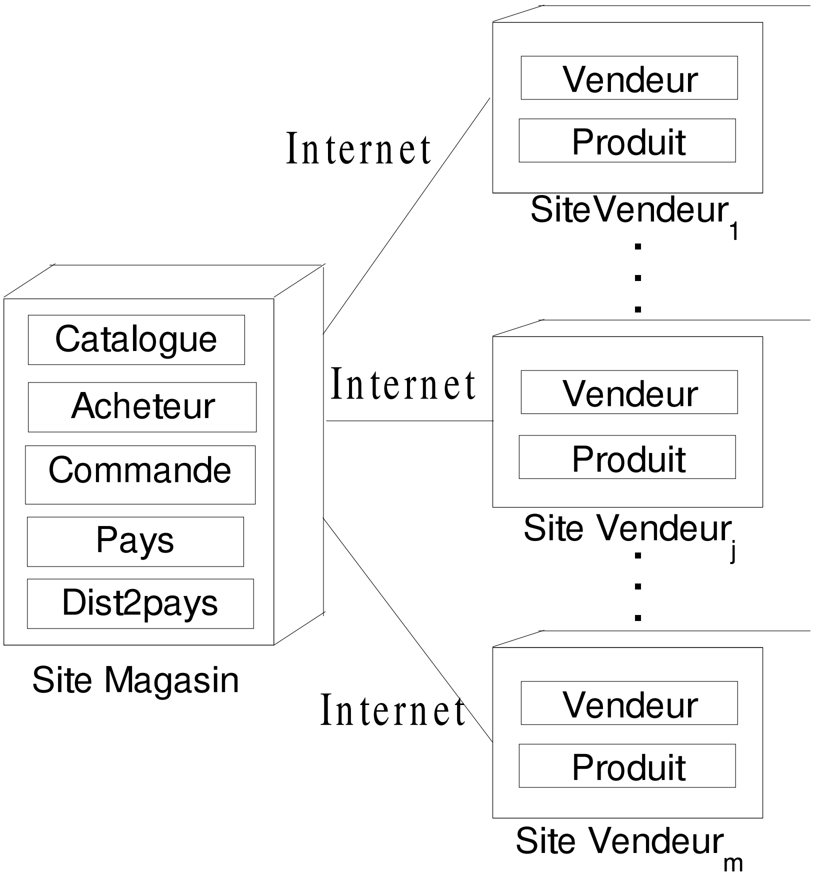

FIG. 5.5 – *Diagramme de distribution de classes*

$$pays\_destin(o) = r \wedge distance(q, r) = d \Rightarrow dist\_km(d) \leq 3000$$

C_2 établit que le magasin seulement négociera des produits lorsque le transport requis depuis le pays du vendeur à celui de l'achet soit au maximum de 3000 Km. Cette contrainte peut être considérée séparément pour chaque produit $Prod_i$ distribué par le magasin. De cette façon nous avons C_{2i}, la contrainte C_2 particularisée au $Prod_i$.

Si nous détaillons la BD de la figure 5.4 nous pourrions définir d'autres contraintes. Cependant, C_1 et C_2 sont suffisantes pour illustrer la vérification de contraintes d'intégrité dans cet exemple, comme nous le verrons à la suite.

5.2.3 Vérification de contraintes d'intégrité

La figure 5.6 montre la distribution de la transaction imbriquée pour le traitement d'une commande qui contient n produits, considérant la distribution de classes posée dans le diagramme de la figure 5.5. Les contraintes touchées par chaque transaction feuille sont présentées sous la feuille correspondante et le $SCA(C_i)$ est associé au processus de $Check(C_i)$.

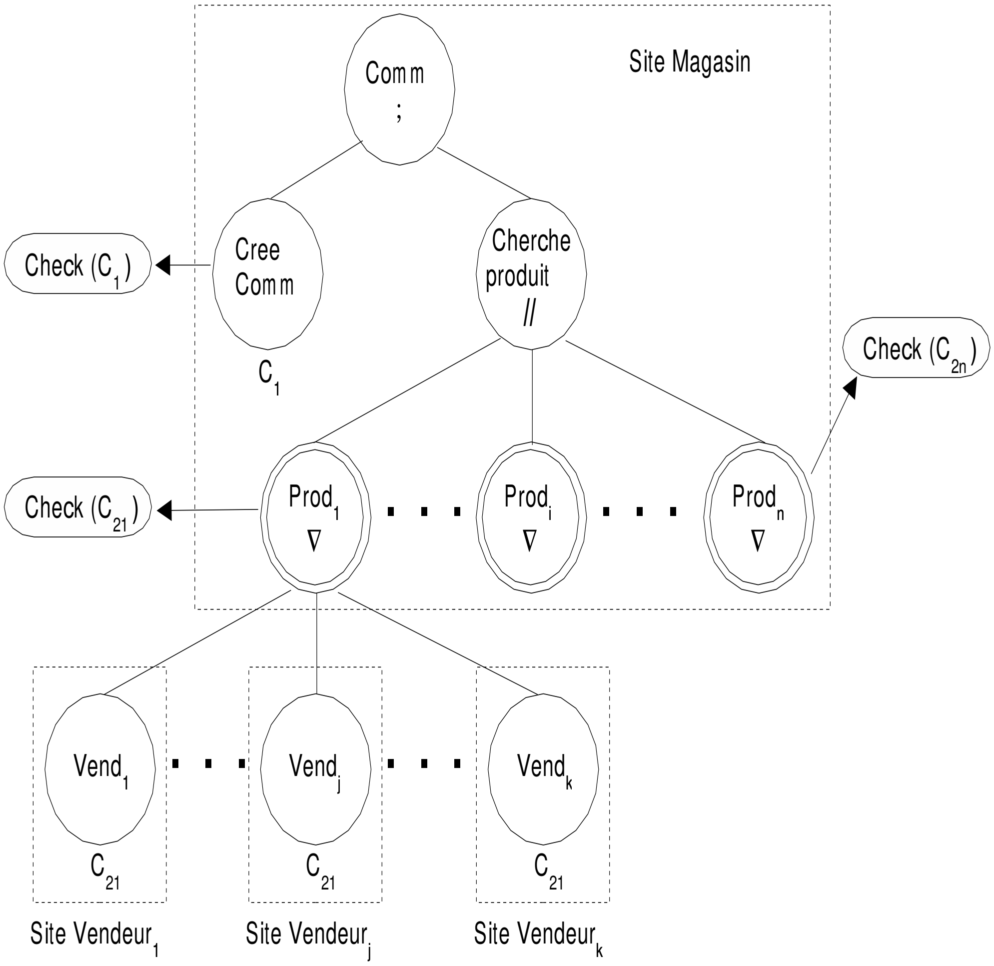

FIG. 5.6 – *Contraintes touchées par la transaction de traitement de commandes*

Ci-dessous, nous décrivons le processus de vérification des CI dans cette application particulière suivant les stratégies proposées dans le chapitre 4.

Pour la contrainte C_i qui est local au lieu du magasin, à la fin de la sous-transaction

Crée-Ordre commence sa vérification. Si la contrainte est satisfaite la TI continuéra son exécution, mais si CI est violée, la transaction racine abandonnera.

Dans le premier cas, commencera la sous-transaction *Cherche-produits*, la quelle commence en parallèle une sous-transaction $Prod_i$ pour chaque produit commandé dans l'ordre. Chacune de ces sous-transactions constitue le $SCA(C_{2i})$ de la partition de la contrainte C_2 correspondante au produit i. Le type d'exécution des filles de chaque sous-transaction $Prod_i$ est alternatif (indiqué par l'opérateur $\bigtriangledown$) ce qui signifie que seulement l'une d'elles est validé. Supposons que ce soit la fille $Vend_j$, qui s'est exécutée dans le nœud correspondant au Vendeur j. Lorsque $Prod_i$ reçoit le message de validation de $Vend_j$, commence le processus $Check(C_{2i})$ sur le nœud du magasin. Comme il est montré sur la figure 5.7, $Check(C_{2i})$ est la racine d'une transaction imbriquée qui a deux filles qui s'exécutent séquentiellement. La sous-transaction $Check(PL_j)$ s'exécute dans le nœud correspondant au Vendeur j, et étant donné que dans ce cas il n'existe pas de prédicat local dont la satisfaction soit suffisante pour garantir la contrainte globale, cette transaction a comme fonction de générer l'ensemble $Intersite_j$ qui doit être envoyé au nœud du magasin pour mener à bien la vérification de C_{2i}. Dans ce cas sera l'ensemble *Pays_produit* qui contient le pays d'origine du vendeur. La sous-transaction $Check(CL_{Magasin})$ utilise l'ensemble $Intersite_j$ et les objets locaux au magasin pour mener à bien localement la vérification de la contrainte globale. Ceci donne, vérifie que :

$$C_2 : \forall(r \in Pays\_produit, o \in Commande, q \in Pays, d \in Dist2pays),$$
$$Pays\_destin(o) = q \wedge Distance(r, q) = d \Rightarrow Dist\_km(d) \leq 3000$$

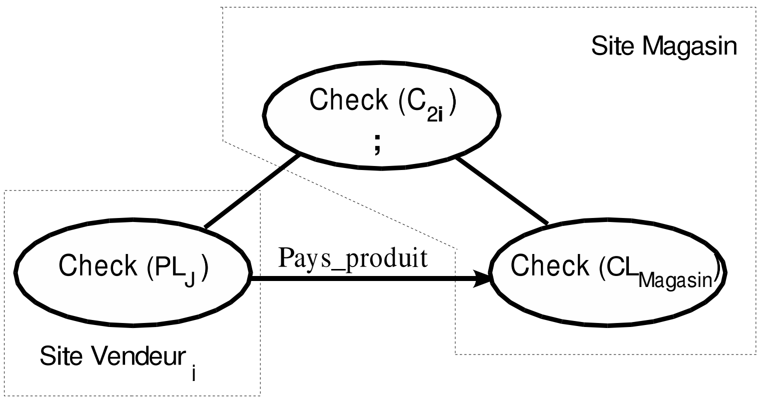

FIG. 5.7 – *Transaction imbriquée pour vérifier C_2 avec le produit i*

A travers un simple exemple conduit dans cette section on a montré l'applicabilité de la méthode proposée dans le chapitre 4 pour la vérification de contraintes d'intégrité globales dans les systèmes multibase qui supportent les transactions imbriquées, dans le contexte de commerce électronique. Egalement, nous montrons qu'il est possible de modéliser quelques phases du commerce électronique en utilisant le modèle de transactions imbriquées. De cette façon, nous montrons que les concepts proposés dans cette thèse peuvent être utilisés dans la pratique.

Conclusion

Pendant le développement de nos recherches sur les transactions imbriquées, nous avons obtenus des résultat publiés que nous présentons dans cette conclusion.

L'évaluation du système SIMA comme un cas d'étude des systèmes qui supportent transactions imbriquées réparties nous a fait remarque qu'avoir des applications basées sur le modèle des transactions imbriquées est une caractéristique puissante et très flexible. En effet, le modèle des transactions imbriquées utilisé par SIMA garantit une concurrence fiable et un contrôle d'exécution approprié. Le coût ajouté par l'outil pour l'exécution d'une application est negligeable comparé au temps que l'utilisateur gagne dans la construction de ses applications dans le domaine de traitement des images médicales.

Nous avons proposé dans dans cette Thèse une méthode novatrice pour la maintenance de contraintes d'intégrité définies de manière déclarative dans un environnement de base de données centralicée supportant des transactions imbriquées. L'idée principale de cette solution est d'associer le contrôle d'une contrainte à la transaction dans l'arbre qui correspond au plus petit ancêtre commun de toutes les feuilles qui touchent la contrainte. Cette solution a été mise en œuvre à l'Universidad Central de Venezuela dans un prototype, avec l'objectif de valider et d'étudier le comportement de l'algorithme proposé. Les résultats obtenus dans les preuves réalisées nous permettent d'établir certaines conclusions respect au comportement de l'algorithme pour vérifier des contraintes d'intégrité proposé dans cette thèse.

En premier lieu, nous pouvons affirmer que notre algorithme constitue une solution efficace pour la vérification des contraintes d'intégrité en présence de transactions imbriquées. Dans un scénario optimiste, où la probabilité que les contraintes d'intégrité soient violées par les transactions est suppose basse (0,1), la contribution est notable, puisque algorithme arrive à sauver le 8 % du total des transactions et le 11 % du total des transactions validées. Dans un scénario pessimiste où la probabilité que les contraintes d'intégrité soient violées par les transactions est supposée haute (0,5), l'efficacité est agrandie, puisque notre algorithme permet de sauver 16,91 % du total des transactions et 34 % des transactions qui valident, le font grâce à lui.

D'un autre côté, avec notre méthode les contraintes se vérifient le plus tôt possible pendant l'exécution d'une transaction imbriquée et par conséquent est abandonnée la transaction entière lorsque la violation de l'intégrité ne peut pas être recuperée. Ceci constitu un gain par rapport aux autres méthodes où la vérification de contraintes est réalisée à la fin de la transaction racine. Dans le scénario pessimiste, on évite que soient exécutées en moyenne 50 % des sous-transactions d'une transaction imbriquée qui inévitablement seraient abandonnées. Dans le scénario optimiste, le pourcentage de travail inutile évité est de 30 % en moyen.

Nous avons adapté cette approche à un environnement multibase: nous montrons comment minimiser le coût de vérification des contraintes en minimisant les transferts de données entre sites. Cette approche permet de garantir la cohérence de la multibase même en présence de contraintes globales, c'est-à-dire impliquant des données de plusieurs sites, tout en abandonnant un minimum de sous-transactions en cas d'abandon partiel de la transaction imbriquée. A l'aide d'une typologie, nous avons établi des stratégies différentes de vérification en fonction de la nature des contraintes et de la structure des transactions.

A notre connaissance, notre approche est la seule à traiter de la vérification de contraintes globales en présence de transactions imbriquées réparties. D'autres approches pour maintenir des contraintes d'intégrité globales [55, 30, 31] ne s'intéressent qu'à des modèles de transactions réparties à un seul niveau d'imbrication.

Finalement, nous réalisons une étude pour montrer l'utilité de notre méthode pour le maintien de contraintes d'intégrité dans des transactions imbriquées, dans le commerce électronique.

Nos travaux dans cette thèse ont été assujettis à certaines limites, mais ils laissent entrevoir de nombreuses perspectives tant au niveau de la validation et de l'amélioration de notre méthode, qu'au niveau de l'extension vers d'autres modèles de transactions.

- Quant au mécanisme proposé proprement dit, nous pensons qu'il peut être optimisé, principalement en ce qui concerne à abandonner seulement les feuilles qui ont provoqué la violation d'une contrainte donnée et celles qui certainement aient observé leurs résultats. Ceci permettra d'abandonner moins de transactions et par conséquent une plus grande probabilité d'atteindre la validation de la transaction globale. Cependant, il est nécessaire de bloquer l'exécution de la transaction imbriquée pendant la vérification des contraintes et d'intervenir avec le contrôle d'exécution et le contrôle de concurrence afin de faire des recherches à propos des transactions qui effectivement ont violé la contrainte.

- Un travail à développer c'est l'utilisation de notre algorithme dans une application et un environnement réel. Ce travail nécessite l'intégration au sein du même système, d'un module de gestion de contrainte optimisé selon l'approche classique, d'un gestionnaire de transactions imbriquées et de notre prototype.

- Bien que les transactions imbriquées soient les plus connues et les plus utilisées des transactions avancées, elles ne sont pas bien adaptées à toute sorte d'applications complexes. En particulier la méthode de contrôle de concurrence [45, 42] est très stricte pour des applications comme le commerce électronique, le travail coopératif, et d'autres. Maintenir le verrouillage des objets, jusqu'à la fin de la transaction est ici mal adapté [21, 28, 6, 37]. Ainsi plusieurs autres modèles ont été proposés, où est plus flexible la propriété d'isolation comme on l'a vu au chapitre 1. Nous sommes intéressés à étendre notre approche pour l'adapter au comportement des autres modèles de transactions avancées. Pour cela, il faut étudier la sémantique des modèles de transactions ouvertes qui permettraient de valider une partie de la transaction alors qu'elle n'est pas globalement terminée. De tels modèles on besoin de la mise en oeuvre de mécanismes de compensation dans le cas une partie validée soit remise en cause et doit être finalement annulée, ce qui pose de nouveaux problèmes quant à la satisfaction des contraintes.

Bibliographie

[1] Transarc Encina Product Information. http://www.transarc.com/Solutions. http://www.transarc.com/Product/Txseries/Encina/.

[2] OMG UML—Unified Modeling Language Specification. Version 1.3. URL: http://www.omg.org/uml, June 1999.

[3] S. De Amo and N. Bidoit. Contraintes dynamiques d'inclusion et schémas transactionnels. In *Proc. Neuvièmes Journées Bases de Données Avancées, BDA'93*, pages 401–424, Toulouse, France, 1993.

[4] V. Benzaken and A. Doucet. Thémis: A Database Programming Language Handling Integrity Constraints. *The VLDB Journal*, 4(3):493–517, July 1995.

[5] V. Benzaken and X. Schaefer. Static management of integrity in object-oriented databases: Design and implementation. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Proc. of the 6th Int. Conf. on Extending Database Technology, the Advances in Database Technology - EDBT'98*, volume 1377, pages 311–325. LNCS, March 1998.

[6] J. Besancenot, M. Cart, J. Ferrié, R. Gerraoui, P. Pucheral, and B. Traverson. *Les systèmes transactionnnels*. Hermes, Paris, 1997.

[7] B. T. Blaustein. *Enforcing Database Assertions*. PhD thesis, Harvard University, Cambridge, MA, 1981.

[8] E. Boertjes, P. W. P. J. Grefen, J. Vonk, and P. M. G. Apers. An Architecture for Nested Transactions Support on Standard Database Systems. In G. Quirchmayr, E. Schweighofer, and T. J. M. Bench-Capon, editors, *Proc. 9th Int. Conf. Database and Expert Systems Applications, DEXA'98*, volume 1460 of *LNCS*, pages 448–459, Vienna (Austria), August 1998. Springer-Verlag.

[9] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, 1(2):181–240, October 1992.

[10] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Automatic Generation of Production Rules for Integrity Maintenance. *ACM Transactions on Database Systems*, 19(3):367–422, September 1994.

[11] S. Ceri and J. Widom. Deriving Production Rules for Constraint Management. In D. McLeod, R. Sacks-Davis, and H.-J. Schek, editors, *Proc. of the 16th Int. Conf. on Very Large Data Bases, VLDB'90*, pages 566–577, Brisbane (Australia), August 1990. Morgan Kaufmann Publishers.

[12] P. K. Chrysanthis and K. Ramamritham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. In H. Garcia-Molina and H. V. Jagadish, editors, *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data*, volume 19 of *ACM SIGMOD Record*, pages 194–203, Atlantic City, NJ (USA), June 1990. ACM Press.

[13] J. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems – Concepts and Design.* Addison-Wesley, second edition, 1994.

[14] C.T. Davies. Data processing spheres of control. *IBM Systems Journal*, pages 179–198, 1978.

[15] U. Dayal, M. Hsu, and R. Ladin. A Transaction Model for Long-Running Activities. In G. M. Lohmann, A. Sernadas, and R. Camps, editors, *Proc. of the 17th Int. Conf. on Very Large Data Bases, VLDB'91*, pages 113–122, Barcelona (Spain), September 1991. Morgan Kaufmann Publishers.

[16] B. Defude and H. Martin. Integrity checking for Nested Transactions. In R. Wagner and H. Thoma, editors, *Proc. 7th Int. Conf. Database and Expert Systems Applications, DEXA'96*, pages 147–152, Zurich (Switzerland), September 1996. IEEE-CS Press.

[17] A. Doucet, S. Gançarski, G. Jomier, and S. Monties. Integrity Constraints in Multiversion Databases. In R. Morrison and J. B. Keane, editors, *Advances in Databases: 14th British National Conf. on Databases, BNCOD 14*, volume 1094 of *LNCS*, pages 56–73, Edinburgh (UK), July 1996. Springer-Verlag.

[18] A. Doucet, S. Gançarski, C. León, and M. Rukoz. Checking integrity constraints in multidatabase systems with nested transactions. In *6th Int. Conference on Cooperative Information Systems,COOPIS'2001, Trento, Italy, September 5-7, 2001.*

[19] A. Doucet, S. Gançarski, C. León, and M. Rukoz. Estrategias para verificar restricciones de integridad globales en multi base de datos con transacciones anidadas.

In *XXVI Conferencia Latinoamericana de Informática, CLEI'2000*, Ciudad de Mexico, September 2000. Centro de Estudios Latinoamericanos de Informática.

[20] A. Doucet, S. Gançarski, C. León, and M. Rukoz. Nested Transactions with Integrity Constraints. In G. Saake, K. Schwarz, and C. Türker, editors, *Transactions and Database Dynamics, 8th Int. Workshop on Foundations of Models and Languages for Data and Objects,TDD'99, Dagstuhl Castle, Germany, September 27-30, 1999, Selected Papers*, volume 1773 of *LNCS*, pages 130–149, Berlin, 2000. Springer-Verlag.

[21] A. K. Elmagarmid, editor. *Database Transaction Models For Advanced Applications*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[22] D. Enser and I. Stevenson. *Oracle8 Design Tips*. O'Reilly & Associates, Cambridge, MA, 1997.

[23] J. L. Eppinger, L. B. Mummert, and A.Z. Spector, editors. *Camelon and Avalon — A Distributed Transaction Facility*. Morgan Kaufmann Publishers, San Francisco, CA, 1991.

[24] C. Fahrner, T. Marx, and S. Philippi. DICE: Declarative Integrity Constraint Embedding into the Object Database Standard ODMG-93. *Data & Knowledge Engineering*, 23(2):119–145, August 1997.

[25] S. Gançarski, C. León, and M. Rukoz. Vérification de contraintes dans les transactions imbriquées réparties. In *Proc. 16èmes Journées Bases de Données Avancées, BDA'2000*, pages 61–80, Blois,France, October 2000.

[26] H. Garcia-Molina and K. Salem. Sagas. In U. Dayal and I. L. Traiger, editors, *Proc. of the 1987 ACM SIGMOD Int. Conf. on Management of Data*, volume 16 of *ACM SIGMOD Record*, pages 249–259, San Franscisco, CA (USA), May 1987. ACM Press.

[27] J. Gray. The Transaction Concept: Virtues and Limitations. In C. Zaniolo and C. Delobel, editors, *Proc. of the 7th Int. Conf. on Very Large Data Bases, VLDB'81*, pages 144–154, Cannes (France), September 1981. IEEE CS Press.

[28] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[29] P. W. P. J. Grefen and P. M. G. Apers. Integrity Control in Relational Database Systems — An Overview. *Data & Knowledge Engineering*, 10:187–223, 1993.

[30] P. W. P. J. Grefen and J. Widom. Protocols for Integrity Constraint Checking in Federated Databases. *Distributed and Parallel Databases*, 5(4):327–355, October 1997.

[31] S. Grufman, F. Samson, S.M. Embury, P.M.D. Gray, and T. Risch. Distributing Semantic Constraints Between Heterogeneous Databases. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, ICDE 1997, April 7-11*, pages 33–42, Birmingham U.K., April 1997. IEEE Computer Society.

[32] A. Gupta and J. Widom. Local Verification of Global Integrity Constraints in Distributed Databases. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, volume 22 of *ACM SIGMOD Record*, pages 49–58, Washington, D.C.(USA), May 1993. ACM Press.

[33] T. Härder and K. Rothermel. Concurrency Control Issues in Nested Transactions. *The VLDB Journal*, 2(1):39–74, January 1993.

[34] A. Hsu and T. Imielinski. Integrity Checking for Multiple Updates. In S. Navathe, editor, *Proc. of the 1985 ACM SIGMOD Int. Conf. on Management of Data*, volume 14 of *ACM SIGMOD Record*, pages 152–168, Austin, Texas (USA), May 1985. ACM Press.

[35] IBM Corporation. *IBM Database Referencial Integrity Usage Guide*, 1989.

[36] H. V. Jagadish and X. Qian. Integrity Maintenance in an OODB. In L.-Y. Yuan, editor, *Proc. of the 18th Int. Conf. on Very Large Data Bases, VLDB'92*, pages 469–480, Vancouver (Canada), August 1992. Morgan Kaufmann Publishers.

[37] S. Jajodia and L. Kerschberg, editors. *Advanced Transaction Models and Architectures*. Kluwer, 1997.

[38] H. F. Korth and G. D. Speegle. Formal Aspects of Concurrency Control in Long-Duration Transactions Systems Using the NT/PV Model. *ACM Transactions on Database Systems*, 19(3):492–535, September 1994.

[39] C. León and M. Rukoz. Control de concurrencia en un servidor de imágenes distribuido. In *Proc. XX Conferencia Latinoamericana de Informática, PANEL'94*, pages 1363–1372, Ciudad de Mexico, September 1994. Centro de Estudios Latinoamericanos de Informática.

[40] C. León, M. Rukoz, and M. Rívas. Una Herramienta Java para Aplicacionnes Distribuidas de Tratamiento de Imágenes Bioméedicas. In *Proc. XXIV Conferencia Latinoamericana de Informática, PANEL'98*, pages 855–866, Quito (Ecuador), October 1998. Centro de Estudios Latinoamericanos de Informática.

[41] B. Liskov. Distributed Computing in Argus. *Communications of the ACM*, 31(3):300–312, March 1988.

[42] N. A. Lynch. Concurrency Control for Resilient Nested Transactions. *Advances in Computing Research*, 3:335–373, 1986.

[43] S. K. Madria. A study of Concurrency Control and Recovery Algorithms in Nested Transaction Environment. *The Computer Journal*, 40(10):630–639, 1997.

[44] S. Monties. *Cohérence des Bases d'Objets Multiversion*. PhD thesis, Université de Paris I, Paris, France, 1997.

[45] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, Cambridge, MA, 1985.

[46] J. E. B. Moss. Log-Based Recovery for Nested Transactions. In P. M. Stocker, W. Kent, and P. Hammersley, editors, *Proc. of the 13th Int. Conf. on Very Large Data Bases, VLDB'87*, pages 427–432, Brighton (England), September 1987. Morgan Kaufmann Publishers.

[47] J.-M. Nicolas. Logic for Improving Integrity Checking in Relational Data Bases. *Acta Informatica*, 18(3):227–253, 1982.

[48] OCDE. *Perspectives des technologies de l'information de l'OCDE 2000–TIC, commerce électronique et économie de l'information*. Organization de Coopération et de Developpement Économiques, París, France, 2000.

[49] P.-Y. Policella. *Cohérence dans les Bases de Données Orientées Objet*. PhD thesis, Université de Paris XI, Paris, France, 1996.

[50] M. Rukoz. Hierarchical Deadlock Detection for Nested Transactions. *Distributed Computing*, 4:123–129, 1991.

[51] M. Rukoz, C. León, and M. Rívas. SIMA: A Java Tool for Constructing Image Processing Applications on a Heterogeneous Network. *to appear in Parallel and Distributed Computing Practices. Special Issue on Distributed Object Systems*, 2001. Acepted 1999.

[52] K. Schwarz, C. Türker, and G. Saake. Analyzing and Formalizing Dependencies in Generalized Transaction Structures. In M. Tamer Özsu, A. Dogac, and Ö. Ulusoy, editors, *Proc. of the Third Int. Conf. on Integrated Design and Process Technology, IDPT - Volume 2, Int. Workshop on Issues and Applications of Database Technology, IADT'98*, pages 55–62, Berlin (Germany), July 1998. Society for Design and Process Science.

[53] T. Sheard and D. Stemple. Automatic Verification on Database Transaction Safety. *ACM Transactions on Database Systems*, 14(3):322–368, September 1989.

[54] A.P. Sheth and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[55] A.P. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 555–581. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[56] Sybase Inc. Press, Emeryville,CA (USA). *Sybase SQL Server*, 1989.

[57] H. Wächter and A. Reuter. The ConTract Model. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 219–263. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[58] G. Weikum, A. Deacon, W. Schaad, and H.-J. Schek. Open Nested Transaction in Federated Database Systems. *IEEE Data Engineering Bulletin*, 16(2):4–7, June 1993.

[59] G. Weikum and H.-J. Schek. Concepts and Applications of Multilevel Transactions and Open Nested Transactions. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 515–553. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[60] J. Widom and S. Ceri, editors. *Active Database Systems — Triggers and Rules for Advanced Database Processing.* Morgan Kaufmann Publishers, San Francisco, CA, 1996.