J-RADEX[1] : UN SIMULATEUR DE SGBDTR[2] CONVIVIAL[3]

Jérôme Haubert
Etudiant Doctorant en Informatique
Jerome.Haubert@univ-lehavre.fr, +33 (0)2 32 74 43 84

Bruno Sadeg, Laurent Amanton,
Maîtres de conférences en Informatique
Bruno.Sadeg@univ-lehavre.fr, +33 (0)2 32 74 44 05
Laurent.amanton@univ-lehavre.fr, +33 (0)2 32 74 43 19

Roland Coma
Etudiant Doctorant en Informatique
Roland.Coma@univ-lehavre.fr, +33 (0)2 32 74 43 84

Adresse Professionnelle
LIH, UFR des Sciences et Techniques ★ Université du Havre
★ 25 rue Philippe Lebon ★ BP 540 ★ F-76 058 Le Havre Cedex

Résumé : L'utilisation d'un simulateur est très importante pour évaluer les performances des nouveaux protocoles. RADEx est un simulateur de SGBDTR puissant prenant en compte bon nombre des paramètres relatifs aux SGBDTR. Dans cet article, nous proposons notre contribution à l'évolution de ce simulateur pour le rendre plus convivial et pour y intégrer de nouveaux modules et concepts apparus récemment. Le but de notre contribution est de permettre à l'utilisateur de gérer un environnement de simulation correspondant à ses attentes.

Abstract : A simulator is generally used to test the performances of the new protocols. RADEx is a simulator of RTDBMSs. It is powerful since it takes into account the main parameters relative to RTDBMSs. In this article, we want to give our contribution in the evolution of this simulator by making it easier to use and by integrating some new modules and concepts. The aim is to provide the user a simulation environment corresponding to its expectation.

Mots clés : Base de donnée temps réel, simulateur, modules, interface d'entrée et de sortie, contrôle de concurrence, ordonnancement.

Key words : Real-time database, simulator, modules, input and output interface, concurrency control, scheduling.

[1] *Java Real-time Active Database Experimental system*
[2] Système de Gestion de Base de Données Temps Réel
[3] Ce travail est subventionné par le Ministère Français de la Recherche dans le cadre d'une ACI-JC (no1055)

J-RADEx : un simulateur de SGBDTR convivial

L'utilisation d'un simulateur est très importante pour évaluer les performances des nouveaux protocoles. RADEx est un simulateur de SGBDTR puissant prenant en compte bon nombre des paramètres relatifs aux SGBDTR. Dans cet article, nous proposons notre contribution à l'évolution de ce simulateur pour le rendre plus convivial et pour y intégrer de nouveaux modules et concepts apparus récemment. Le but de notre contribution est de permettre à l'utilisateur de gérer un environnement de simulation correspondant à ses attentes.

1 - INTRODUCTION

La plupart des recherches sur les Systèmes de Gestion de Base de Données (SGBD), notamment temps réel, concerne la gestion des transactions. Les problèmes rencontrés sont alors le contrôle de concurrence, la validation des transactions distribuées ou encore l'ordonnancement des files d'attente de transactions. Pour résoudre chacun de ces problèmes, des protocoles ont été proposés. Pour les SGBD Temps Réel (SGBDTR), ces méthodes sont généralement testées et évaluées par le biais de simulations car peu de prototypes existent (Ramamritham, 1993). Chaque concepteur de protocoles évalue donc sa méthode dans un environnement de simulation particulier, ce qui peut aboutir à des interprétations différentes, voire à des résultats incompatibles (Huang, 1990).
Sivasankaran et al. (1994) ont proposé un simulateur de bases de données temps réel (*Real-time Active Database Experimental system* : RADEx) dont le modèle a ensuite été étendu par Hansson (1998). Ce simulateur est construit suivant un modèle permettant de tester des protocoles de contrôle de concurrence, des protocoles de validation, des méthodes de recouvrement, etc. En effet, chaque composant du modèle gère une partie de l'exécution des transactions sur la base de données.
Le simulateur RADEx est basé sur un modèle objet et a été écrit en langage C/C++. La première étape avant toute utilisation du simulateur est l'initialisation des paramètres de chaque gestionnaire (ou composant). Dans la version de base de RADEx, les valeurs sont lues dans un fichier et quelques paramètres et variables sont initialisés dans le fichier Makefile (qui construit le simulateur à partir des programmes sources sous Unix).
Comme on peut le constater dans (Sivasankaran, 1994), RADEx est un simulateur puissant. En effet, l'utilisation d'un simulateur permet aux simulations d'être uniformes quel que soit le protocole testé. De plus, le modèle utilisé permet de tenir compte des principaux paramètres apparaissant dans les SGBDTR. Nous avons donc choisi de contribuer à l'évolution de ce simulateur.
Notre première contribution à cette extension est la réécriture de tous les composants de RADEx en langage Java. On augmente ainsi la portabilité du système. Cette nouvelle version de RADEx est nommée J-RADEx.
La deuxième étape de notre travail est la réalisation d'une interface graphique d'entrée et de sortie pour faciliter l'initialisation des paramètres, visualiser les résultats et simplifier l'utilisation du simulateur.
La troisième étape consiste à intégrer dans le simulateur de nouveaux protocoles de base pour la comparaison des résultats avec les nouveaux protocoles à tester.
Enfin, une quatrième étape ajoute de nouvelles notions apparues ces dernières années et pouvant être utilisées dans l'étude des performances (cf. section 3.3).
Le but de ce papier est donc de montrer les avantages conséquents du simulateur RADEx et de présenter une adaptation de celui-ci (J-RADEx) plus conviviale et plus complète. La suite du document est organisée de la façon suivante. La section 2 présente le modèle de base du simulateur. La section 3 présente le modèle utilisé pour J-RADEx, en particulier, les ajouts par rapport à RADEx. Enfin, la section 4 conclut ce document en donnant des perspectives à ce travail.

2 - LE SIMULATEUR RADEX DE BASE

Dans cette section, nous présentons l'architecture qui a servi de base à la conception du simulateur RADEx par Sivasankaran et al. (1994) (figure 1).
Initialement, le modèle utilisait une collection d'objets en multi-niveaux mais pour des raisons de simplicité, seuls deux niveaux sont utilisés. Le premier niveau définit tous les composants du modèle (les différents gestionnaires de la figure 1) et le second niveau représente les communications entre ces composants (représentées par des flèches sur la figure 1).
Le simulateur RADEx prend en compte un très grand nombre de paramètres relatifs aux SGBDTR centralisés et distribués. En effet, certains paramètres comme le nombre de CPU ou le nombre de disques

permettent de représenter la répartition des données sur plusieurs sites. On peut donc dire que le simulateur RADEx s'intéresse à au moins quatre types de SGBD : centralisé (temps réel ou non) et distribué (temps réel ou non).

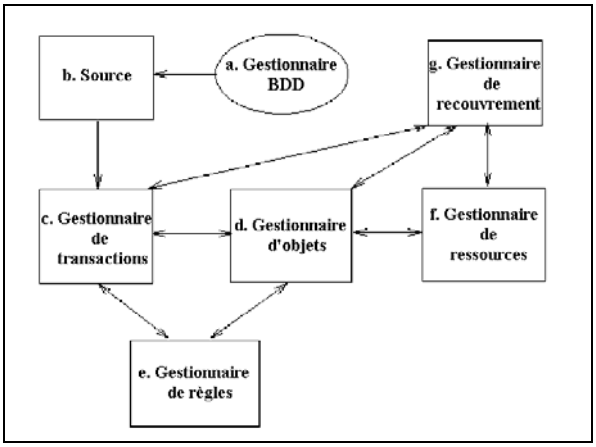

Figure 1 : Architecture du simulateur

Voyons maintenant en détail les différents composants de RADEx. Une liste exhaustive des paramètres inhérents à chaque composant est présentée dans Sivasankaran et al. (1994).

a - Gestionnaire de la base de données

Ce composant, appelé aussi *DBManager*, définit la base de données en elle-même (Sivasankaran, 1994). Par exemple, il détermine le nombre de classes d'objets dans le système, le nombre d'instances de chaque classe, etc. Chaque classe d'objets possède un certain nombre de méthodes ayant une durée d'exécution estimée et une probabilité d'exécution. La base de données est également définie par le nombre de pages disques dont elle dispose. Pour des raisons de simplicité, il a été décidé qu'à une classe d'objets correspond une page disque (Sivasankaran, 1994). Dans les SGBD distribués, le gestionnaire de la base doit notamment savoir si on autorise la duplication des données sur plusieurs sites.
Les paramètres du gestionnaire de base de données permettent donc de définir le contexte de la simulation ou, autrement dit, le contexte dans lequel vont s'exécuter les transactions (temps réel ou non).

b - Gestionnaire de la source

Le gestionnaire de la source peut être considéré comme un générateur de transactions puisqu'il simule la soumission des transactions au système (Sivasankaran, 1994). Le générateur crée régulièrement de nouvelles transactions qu'il transmet au gestionnaire de transactions. Il est donc responsable de l'initialisation des caractéristiques des transactions. On peut par exemple simuler l'exécution de transactions apériodiques en utilisant la loi de Poisson pour le taux d'arrivée des transactions. Le gestionnaire de la source s'occupe également d'autres caractéristiques plus globales des transactions. Parmi celles-ci, on peut citer le nombre de classes de transactions, le type d'échéance,...
Nous avons donc d'un côté les informations concernant les transactions et de l'autre les informations concernant les accès aux données de ces transactions. Comme exemple de caractéristiques d'accès aux données, on peut citer la probabilité qu'une transaction modifie une donnée (*update transaction*) ou le nombre d'objets accédés par une classe de transaction donnée. Une fois ces caractéristiques déterminées, le gestionnaire de la source crée de nouvelles transactions qu'il soumet ensuite au gestionnaire de transactions.

c - Gestionnaire de transactions

L'exécution des transactions est simulée par le gestionnaire de transactions ou *Transaction Manager* (TM) (Sivasankaran, 1994). En particulier, le gestionnaire de transactions gère les évènements relatifs aux transactions (*begin*, *commit* ou *abort*) qui sont transmis au gestionnaire de règles. Une autre tâche gérée par le gestionnaire de transactions est l'assignation des priorités aux transactions. C'est grâce à ce gestionnaire que l'on peut tester différentes politiques d'assignations de priorités. Une fois les priorités des transactions calculées, le gestionnaire de transactions exécute réellement la transaction en demandant au gestionnaire d'objets d'exécuter les accès aux objets en évoquant les méthodes d'accès à ces données.
Le schéma maître-esclaves (ou coordinateur-participants) est très utilisé pour représenter les SGBD distribués. Le coordinateur s'occupe de recevoir les transactions, de distribuer les sous-transactions sur les différents sites participants et de récupérer les résultats. Les participants reçoivent des parties de transactions du coordinateur qu'ils exécutent sur leur base de données locale. Les informations concernant les différents sites (coordinateur et participants) sont définies dans le gestionnaire de transactions. Mais, le travail principal du gestionnaire reste l'assignation des priorités.

d - Gestionnaire d'objets

Le gestionnaire d'objets (*Object Manager* (OM)) lance les méthodes relatives aux objets dont l'accès a été requis par le TM (Sivasankaran, 1994). Il est donc responsable du contrôle de concurrence entre les transactions. Par exemple, la méthode est exécutée (l'objet est accessible) ou un message de blocage ou d'abandon est retourné au TM (l'objet n'est pas accessible).

Le gestionnaire d'objets est relié à la fois au gestionnaire de ressources où les données sont effectivement manipulées et au gestionnaire de transactions qui le sollicite non seulement pour les accès aux données mais aussi pour la validation des transactions. En fonction de l'exécution de la transaction, le gestionnaire OM doit décider de valider (commit) ou d'abandonner (abort) la transaction. En terme de simulation, cela implique que le gestionnaire d'objets permet de tester à la fois des protocoles de contrôle de concurrence et des protocoles de validation.

e - Gestionnaire de règles

Ce gestionnaire pourrait être surnommé le contrôleur d'intégrité puisqu'il permet de vérifier que les transactions qui s'exécutent respectent les contraintes d'intégrité du système. Les transactions doivent respecter certaines règles en fonction des données auxquelles elles accèdent pour que la base reste cohérente (Delobel, 1982). Par exemple, le nombre de réservation dans un avion ne peut excéder le nombre de places. Ce module est en fait complexe et possède de nombreux paramètres.
Régulièrement, ce gestionnaire reçoit les évènements des transactions en provenance du gestionnaire de transactions et du gestionnaire d'objets. Il vérifie ensuite que ces évènements respectent les règles d'intégrité de la base. Si aucune règle n'est violée, un message autorise les gestionnaires TM et OM à continuer l'exécution de la transaction. Sinon, le gestionnaire de règles tente de trouver les conditions nécessaires pour que les règles soient respectées et, si de telles conditions existent, les opérations correspondantes sont transmises au TM comme s'il s'agissait d'une nouvelle transaction (Sivasankaran, 1994). Une autre tâche du gestionnaire de règles est le chargement des transactions périodiques.

f - Gestionnaire de ressources

Ce gestionnaire modélise les ressources physiques du SGBDTR (Sivasankaran, 1994). Il s'occupe notamment des pages disques, de la mémoire centrale et du (ou des) CPU. Lorsque, le gestionnaire d'objets effectue l'accès à une donnée, le gestionnaire de ressources doit vérifier que la donnée est localisée en mémoire centrale. Si ce n'est pas le cas, il doit charger la page correspondante depuis le disque en utilisant, si nécessaire, une méthode de remplacement des pages mémoires. Le gestionnaire de ressources permet donc de tester les algorithmes d'ordonnancement pour les files d'attente du processeur et des disques mais aussi des algorithmes de remplacement de pages mémoires.
Les principaux paramètres de ce gestionnaire donnent donc des informations sur le nombre de processeurs, de disques, le temps d'accès disque, etc. Dans le cas où une panne surviendrait pendant l'exécution, des échanges de messages avec le gestionnaire de recouvrement permettent de retrouver une base de données stable et cohérente.

g - Gestionnaire de recouvrement

Le gestionnaire de recouvrement n'a pas été étudié en détails dans la première version de RADEx (Sivasankaran, 1994) mais a été ajouté dans la version RADEx++ de Hansson (1998). Son rôle est la gestion des fichiers journaux qui sont des supports stables utilisés en cas de panne. Les évènements des transactions sont régulièrement soumis au gestionnaire de recouvrement qui note les opérations correspondantes dans les fichiers journaux. Les politiques de recouvrement utilisées après une panne examinent ces fichiers pour rendre la base à nouveau cohérente et stable. Nous étudierons plus en détail ce gestionnaire dans des travaux futurs.

3 - LE SIMULATEUR J-RADEX

À partir du simulateur RADEx proposé par Sivasankaran et al. (1994), puis Hansson (1998), nous proposons quelques améliorations pour rendre le simulateur plus attractif et plus complet. Nous avons réécrit le simulateur en langage Java (J-RADEx), ce qui le rend plus portable. Nous avons également intégré une interface graphique pour permettre une utilisation plus aisée et plus conviviale du simulateur. On permet également à l'utilisateur de saisir les paramètres des différents composants en diminuant les risques d'erreurs. Dans un second temps, nous nous sommes intéressés à l'extension du modèle en étudiant notamment l'ajout de nouveaux protocoles et concepts récents permettant de comparer les résultats des protocoles à tester face à ces protocoles. Enfin, dans une dernière étape, nous avons ajouté de nouvelles notions introduites dans les SGBDTR ces dernières années.

3.1 - Présentation de l'interface utilisateur

Dans la version initiale du simulateur, les concepteurs utilisaient un système de fichiers pour les entrées/sorties. Le fichier d'entrée intègre l'ensemble des paramètres à initialiser et le fichier de sortie présente les résultats de la simulation. Nous allons remplacer ces fichiers par des interfaces pour une utilisation plus intuitive du simulateur.

3.1.1 - Interface d'entrée

L'interface d'entrée permet de fournir les valeurs de l'ensemble des paramètres de manière plus attractive. Deux modes de saisie des paramètres sont ainsi proposés à l'utilisateur : à partir d'un fichier (comme pour le simulateur RADEx) ou de manière interactive.

La première méthode doit d'abord vérifier le contenu du fichier avant d'initialiser les paramètres. La seconde méthode oblige l'utilisateur à entrer un par un les paramètres de chaque composant. La figure 2 illustre, par exemple, le gestionnaire de ressource et ses paramètres. Lorsque l'utilisateur a entré tous les paramètres d'un composant et qu'il valide (OK), le programme vérifie la cohérence des valeurs et informe l'utilisateur en cas d'erreur (figure 3).

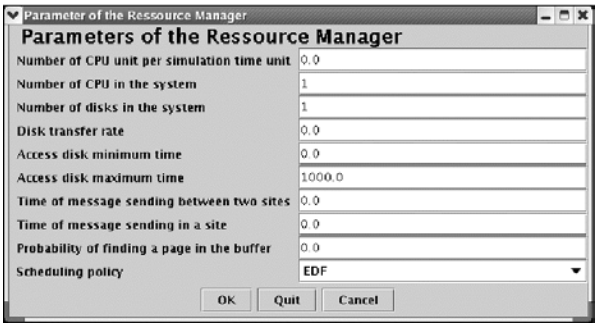

Figure 2 : Exemple de composant, le gestionnaire de ressource.

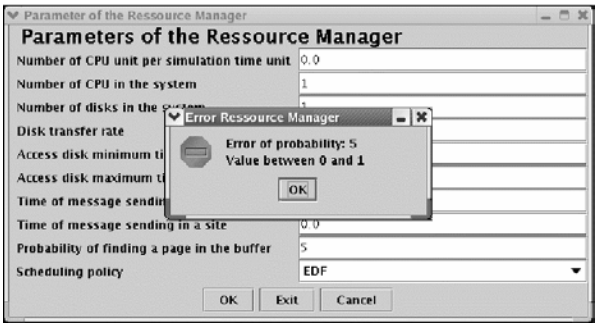

Figure 3 : Vérification des valeurs données par l'utilisateur.

L'utilisateur ne peut passer à l'étape suivante qu'une fois tous les composants initialisés. Un système de coloration des boutons permet de visualiser les composants initialisés on non (dans la figure 4, les gestionnaires de la base et de la source sont initialisés mais pas les autres).

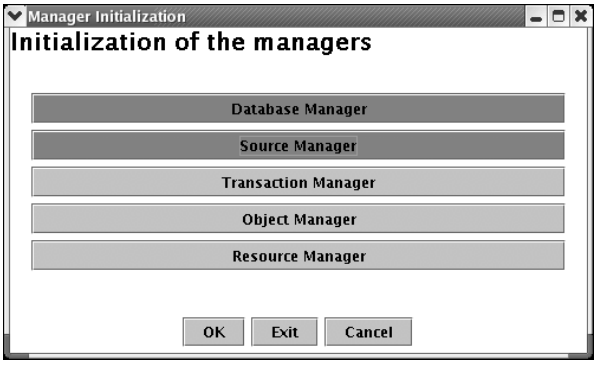

Figure 4 : Les gestionnaires non initialisés.

L'étape suivante est identique pour le mode graphique et le mode fichier. Pour des raisons de réutilisabilité, l'utilisateur peut sauvegarder les paramètres initialisés avant de lancer la simulation. L'avantage de ce type de méthode est que l'utilisateur peut lancer plusieurs simulations en modifiant simplement quelques paramètres et sans sortir du simulateur.
Ensuite, l'étape suivante consiste à choisir le type de protocoles qu'il désire tester (contrôle de concurrence ou validation ou autres...). La simulation peut maintenant être réalisée.

3.1.2 – Interface de sortie

Le fichier de sortie construit par RADEx est difficilement exploitable tel quel : la figure 5 présente une partie d'un fichier de sortie fournit par RADEx. Les renseignements sont fournis de manière brute et la comparaison des résultats n'est pas évidente.

```
* Run 1
Performance Averages:
=================
Miss Ratio of Class0 Transactions[13390/32014]: 0.41825
Number Of Aborts .......: 13390
Number Of Aborts on Entry: 293
Disk Work Wasted On Restarts: 0
CPU Work Wasted On Restarts: 0
Number Of Restarts ......: 16
Restarts On Validation...: 0
.....that complete: 0
[...]
Class Distributions:
1. Transaction Size (SIZ) in number of operations.
2. Number of Transactions of that size (Ns)
3. Number of Aborts for that size (As)
[...]
SIZ   NS     As    AEs Rs    Ts      Ws  [...]
  4   1782   684   192  0   808.39 0.45 [...]
  5   3933  1239    81  1   998.04 0.68 [...]
  6   6397  2147    19  3  1251.97 0.99 [...]
  7   7561  2984     1  3  1507.27 1.38 [...]
Normalized Miss Ratio....          : 46.251%
Normalized Restart Ratio           : 0.071%
[…]
```

Figure 5 : Partie du fichier de sortie fournie par RADEx

Pour pallier ces inconvénients, nous avons élaboré une interface construisant des graphiques à partir du fichier de résultats fourni par RADEx.
Pour cela, nous avons séparé le fichier résultat en deux parties. La première partie concerne les résultats globaux de la simulation, ce qui correspond à la partie *Performance Averages* dans le fichier résultat. La seconde partie détaille les résultats suivant les classes de transactions, ce qui correspond au tableau *Class Distribution* dans le fichier.

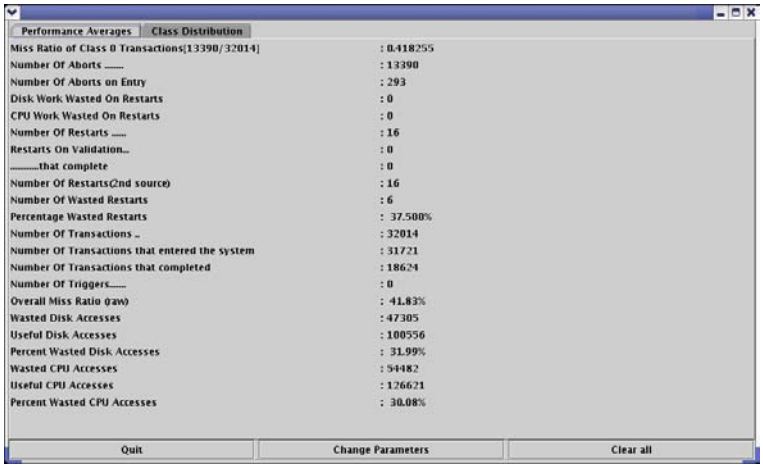

Figure 6 : Résultats globaux de la simulation

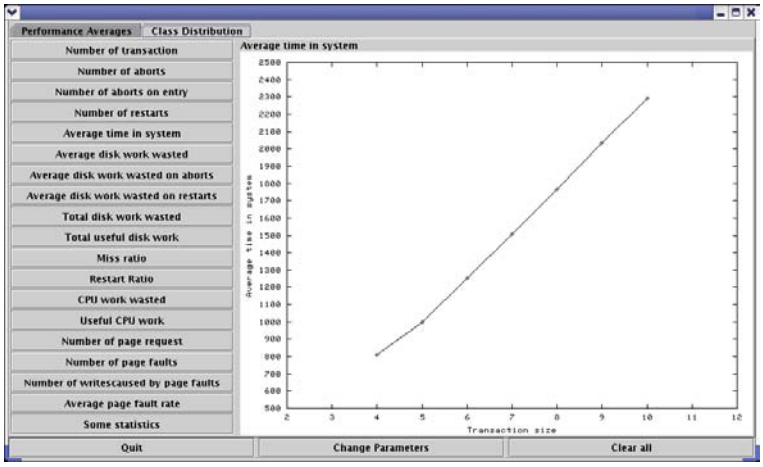

Figure 7 : Résultats en fonction des classes de transactions.

La première partie fournit simplement des résultats numériques indépendants alors que les colonnes du tableau de la deuxième partie permettent d'illustrer l'évolution des différents paramètres en fonction de la classe de transactions utilisée.
Les figures 6 et 7 illustrent respectivement les deux parties du fichier résultat. Ainsi, la figure 6 représente les résultats globaux de la simulation et la figure 7 permet de visualiser les courbes d'évolution de chaque paramètre correspondant aux différentes colonnes du tableau *Class Distribution* du fichier de sortie (les boutons de gauche permettent de passer d'un paramètre à un autre).

3.2 - Extension du modèle

La réalisation initiale de RADEx ne tient pas compte du gestionnaire de recouvrement à cause de sa complexité. En effet, ce gestionnaire implique une étude approfondie des différents types de pannes et des mécanismes de recouvrement permettant de rétablir la base. C'est pourquoi le gestionnaire de recouvrement a été intégré au modèle étendu par Hansson (1998) : RADEx++. Ce dernier ajoute des composants et des paramètres par rapport à RADEx comme présenté en figure 8.
Dans les SGBD distribués, une notion très importante doit être prise en considération dans l'exécution de simulations : les communications. En effet, suivant les capacités du réseau (en terme de surcharge par exemple), les résultats des simulations peuvent être faussés. En général, les messages sont ordonnés sur le réseau par la méthode FCFS (*First Come First Serve*) dont les performances dans un contexte temps réel sont discutables (Cottet, 2000). Il est donc intéressant d'étudier les politiques d'ordonnancement pour les appliquer aux communications et ainsi permettre à davantage de transactions de respecter leur échéance. C'est le but principal des SGBD temps réel (Ramamritham, 1993).
Les simulations comparent généralement les performances des nouveaux protocoles par rapport à ceux qui existent déjà et qui servent de base (comme

le 2PL[4] et ses adaptations pour le contrôle de concurrence). Dans le simulateur RADEx que nous utilisons (datant de 1999), pour le contrôle de concurrence, seuls les protocoles 2PL et OCC[5] ont été incorporés. Or, les performances de ces protocoles sont critiquables dans un contexte temps réel. Il est donc intéressant d'étudier quels sont les protocoles qui servent de base dans les SGBDTR, voire même d'intégrer régulièrement de nouveaux protocoles qui apparaissent dans la littérature.

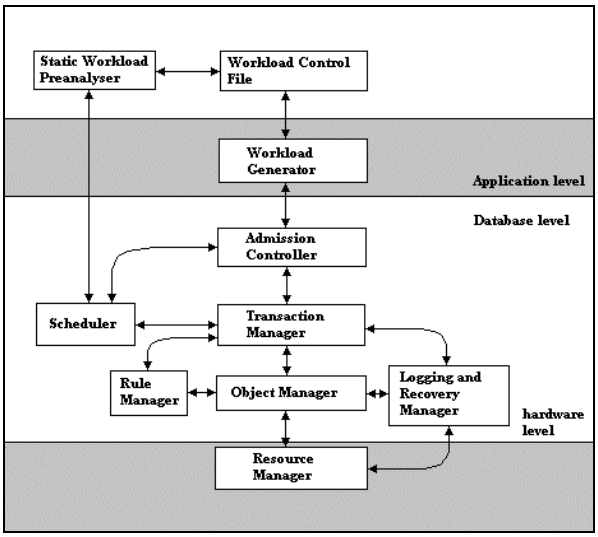

Figure 8 : Le modèle proposé par Hansson (1998)

3.3 - Introduction de nouveaux concepts

Depuis le travail d'implémentation de RADEx (1999), de nouveaux concepts sont apparus dans les SGBD distribués temps réel. Parmi ces concepts, certains sont intéressants pour évaluer les performances des nouveaux protocoles.

Pour les transactions à échéances non strictes (*soft* (Duvallet, 1999)), nous proposons d'étendre le simulateur en utilisant les notions d'ε-donnée et de Δ-échéance (Sadeg, 2000). Ces notions sont particulièrement adaptées aux applications multimédia et on les utilise de la façon suivante. Lorsqu'une transaction ne peut pas respecter son échéance principale, une échéance étendue lui est assignée (échéance+Δ). Si l'échéance étendue d'une transaction ne peut pas être respectée, la transaction retourne des résultats partiels avec une imprécision ne dépassant pas ε. Pour intégrer ces notions au simulateur, il est nécessaire de modifier le gestionnaire de transactions, le gestionnaire de règles et le gestionnaire d'objets.

Un deuxième concept que l'on peut intégrer au simulateur est le prêt des données non encore validées proposé dans le protocole de validation PROMPT (*Permits Reading Of Modified Prepared-data for Timeliness* (Haritsa, 2000)) pour les transactions distribuées. Lorsqu'un site attend la décision finale du coordinateur (*commit* ou *abort*), il peut prêter des données modifiées à d'autres transactions voulant accéder à ces données. Si la transaction "prêteuse" valide (resp. abandonne), les transactions ayant emprunté des données continuent de s'exécuter (resp. sont redémarrées). Certaines conditions ou paramètres permettent d'éviter notamment l'abandon en cascade (Haritsa, 2000). Pour intégrer cette notion au simulateur, il est donc nécessaire de modifier le gestionnaire d'objets.

Récemment, nous avons étudié des transactions distribuées particulières : les transactions pondérées (*weighted transactions* (Sadeg, 2003)). Dans ce modèle, les transactions se divisent en deux types de sous-transactions en fonction de leur poids : les sous-transactions obligatoires (ou vitales) et les sous-transactions optionnelles (ou non-vitales). Par exemple, une transaction peut se décomposer en une partie vitale qui contrôle un processus et une partie non-vitale qui affiche des résultats à l'écran. On retrouve cette notion dans les modèles de transactions étendus comme les transactions emboîtées. Des protocoles utilisant la pondération des transactions ont ainsi été proposés pour le contrôle de concurrence et/ou la validation des transactions distribuées (Sadeg, 2003). L'ajout de cette notion se fait en ajoutant de nouveaux paramètres au moment de la génération des transactions et en intégrant les protocoles de contrôle de concurrence et de validation adéquats dans les gestionnaires d'objets et de transactions.

Dans le simulateur J-RADEx, les concepts ci-dessus peuvent être considérés comme des options pour l'utilisateur lorsqu'il initialise les paramètres de la simulation. De cette façon, la simulation peut être aussi proche que possible du modèle désiré par l'utilisateur.

4 - CONCLUSION

La plupart des nouveaux protocoles sont testés par le biais de simulations. Malheureusement, ces simulations ne sont pas uniformes et peuvent avoir des résultats incompatibles dans certains cas. Il est donc intéressant de se pencher sur la réalisation d'un simulateur de SGBD (distribués et/ou temps réel). Un tel simulateur (RADEx) a été proposé par Sivasankaran et al. (1994).

Malgré ses capacités puissantes, le simulateur RADEx n'est pas très aisé à utiliser, notamment par la quantité de paramètres à initialiser pour lancer une simulation.

[4] *Two-Phase Locking*

[5] *Optimistic Concurrency Control*

Les paramètres sont initialisés à partir de fichiers qui ne sont pas évident à construire : le fichier de données contient de nombreuses valeurs et l'initialisation de certains paramètres se fait dans le fichier Makefile (qui construit le simulateur à partir des programmes sources écrit en C/C++ sous Unix).
Dans ce papier, nous avons proposé d'étendre le simulateur RADEx de base (Sivasankaran, 1994) de la façon suivante : (1) en construisant une interface graphique pour la saisie des paramètres, l'affichage des résultats et plus généralement pour une utilisation plus interactive du simulateur, (2) en ajoutant de nouveaux modules comme la gestion du réseau dans un SGBD distribué et (3) en insérant de nouveaux protocoles et concepts apparus depuis la réalisation du simulateur. Pour réaliser ces modifications, nous avons choisi de réécrire le simulateur en langage Java (J-RADEx). Dans un premier temps, le simulateur de base a donc été réécrit en y intégrant l'interface graphique. Ensuite, nous l'avons augmenté pour qu'il prenne en compte les nouveaux modules et concepts.
Nos travaux futurs consistent d'une part à maintenir à jour le simulateur en y ajoutant de nouveaux concepts et protocoles qui apparaissent dans la littérature et d'autre part d'offrir des outils à l'utilisateur pour lui permettre de créer et d'intégrer lui-même ces propres concepts et protocoles. Nous pensons que le simulateur J-RADEx pourra être considéré comme un bon simulateur de SGBD distribués temps réel et qu'il sera utilisé pour tester et évaluer uniformément les nouveaux protocoles pouvant apparaître dans les différents domaines de recherche sur les SGBD distribués temps réel.

BIBLIOGRAPHIE

Cottet, F., Delacroix, J., Kaiser, C., Mammeri, Z. (2000), "Ordonnancement Temps Réel". Hermès ed.

Delobel, C., Adiba, M. (1982), "Base de données et systèmes relationnels". Dunod Informatique.

Duvallet, C., Mammeri, Z., Sadeg, B. (1999), "Les SGBD Temps Réel". *Technique et Science Informatiques*, vol. 18, n° 5, pp 479–517 (1999).

Hansson, J. (1998), "RADEx++". Technical report, University of Skövde.

Haritsa, J., Ramamritham, K., Gupta, R. (2000), "The PROMPT Real-Time Commit Protocol". *IEEE Transactions on Parallel and Distributed Systems*, vol 11, n° 2.

Ramamritham, K. (1993), "Real-Time Databases". *Journal of Distributed and Parallel Databases*, vol 1, n° 2, pp 199-226.

Sadeg, B., Amanton, L., Haubert, J. (2003), "Tradding Precision for Timeliness in Distributed Real-Time Databases". *5th International Conference on Enterprise Information Systems (ICEIS)*.

Sadeg, B., Saad-Bouzefrance, S. (2000), "Gestion des transactions temps réel à échéance non stricte". *8th International Conférence on Real-Time and Embedded Systems (RTS)*, pp 232-246. Téknéa ed.

Sivasankaran, R., Purimetla, B., Stankovic, J., Ramamritham, K., Towsley, D. (1994), "Design of RADEx - Real-time Active Database Experimental System". Technical report, University of Massachusetts.