Un modèle de données temps réel orienté-objet

Nizar Idoudi, Claude Duvallet, Bruno Sadeg
LITIS, Université du Havre
{prenom.nom}@univ-lehavre.fr

Rafik Bouaziz, Faiez Gargouri
MIRACL-ISIMS
{prenom.nom}@fsegs.rnu.tn

Résumé

Dans les bases de données temps réel (BDTR), les données et les traitements sur ces données sont assujettis à des contraintes temporelles. La modélisation des bases de données temps réel doit donc prendre en considération à la fois des contraintes temps réel du point de vue quantitatif (échéances, périodes, priorités, etc.) et des contraintes temps réel du point de vue qualitatif (concurrence, parallélisme, calculs imprécis, etc.). Dans cet article, nous proposons un modèle de données temps réel orienté-objet pour les bases de données temps réel. L'apport principal de notre proposition consiste en une adaptation du modèle objet aux exigences temporelles des BDTR, i.e. attributs temps réel, méthodes temps réel, support de contrôle de concurrence, etc. Nous illustrons cet apport par des exemples tirés d'une application concrète.

Mots clés : attributs temps réel, objets temps réel, méthodes temps réel, qualité de données.

1. Introduction

Ces dernières années, les besoins en termes de données et de services temps réel se sont beaucoup accrus dans un grand nombre d'applications. Traditionnellement, ces applications sont gérées par des STR[1], bien adaptés pour la prise en compte des contraintes temporelles. Cependant, ils ne sont pas satisfaisants pour la gestion efficace de grands volumes de données. Les systèmes de type SGBDTR[2] ont été conçus pour gérer de grandes quantités de données tout en respectant les contraintes temps réel des applications [14] [15]. Les SGBDTR sont nés du rapprochement entre les SGBD classiques (gestion efficace des données) et les STR classiques (gestion efficace des contraintes temps réel) [16]. Ils se caractérisent essentiellement par la prise en considération des contraintes temps réel, tant au niveau des traitements qu'au niveau des données. Ainsi, dans une BDTR[3], les données et les traitements sur ces données sont assujettis à des contraintes temporelles [9].

Le modèle de données le plus utilisé pour modéliser une BDTR est le modèle relationnel. Toutefois, en raison de la nature de nombreuses applications temps réel qui doivent traiter des données complexes dans des délais très courts, de nombreux chercheurs pensent que le modèle orienté-objet est plus naturel et plus puissant que le modèle relationnel [10]. Plusieurs projets de recherche sur les BDTR ont adopté le modèle orienté-objet pour le développement de leurs prototypes [4] [11]. Ainsi, une BDTR est composée d'un ensemble d'objets appartenant à des classes, qui interagissent les uns avec les autres ou avec l'environnement dynamique de la base, en réponse à des événements [3].

Dans cet article, nous présentons un *modèle de données temps réel orienté-objet*, qui intègre les données temps réel et les transactions temps réel des BDTR et supporte la gestion des données complexes grâce au modèle orienté-objet. Le reste de cet article est organisé comme suit. La section 2 présente le modèle de données et de transactions temps réel que nous utilisons. La section 3 est réservée à la description de notre modèle d'objet temps réel permettant la prise en charge des contraintes temporelles des BDTR. Il comprend entre autres des *attributs temps réel*, des *attributs dérivés temps réel*, des *méthodes temps réel*, une *file d'attente et* un *protocole de contrôle de concurrence.* Nous illustrons notre modèle d'objet sur le cas d'un *Système de surveillance du trafic aérien* proposé dans [13]. En conclusion, nous présentons un bilan de notre contribution et donnons quelques perspectives à nos travaux

2. Modèles de BDTR utilisés

2.1. Modèle de données

Les applications temps réel basées sur les BDTR gèrent essentiellement deux types de données temps réel [15] :

- Les données de base (sensorielles) : il s'agit généralement de données issues de capteurs qui permettent de rendre compte de l'état de l'environnement.
- Les données dérivées : elles sont calculées à partir d'une ou plusieurs données de base et permettent de déduire des informations sur l'environnement.

[1] Système temps réel.
[2] Système de gestion de bases de données temps réel.
[3] Base de données temps réel.

2.2. Modèle de transactions

Les transactions temps réel considérées dans nos travaux possèdent des échéances de type firm (échéances strictes non critiques [5]). C'est à dire que lorsqu'elles dépassent leur échéance, elles deviennent inutiles pour le système et sont abandonnées. Nous considérons deux types de transactions temps réel :

- Les transactions de mise à jour : elles ont pour tâche de mettre à jour régulièrement les données sensorielles. Ces transactions sont exécutées périodiquement pour rafraîchir la base de données.
- Les transactions utilisateur : elles effectuent des opérations de lecture/écriture de données non temps réel et/ou des lectures de données temps réel.

3. Un modèle d'objet temps réel

Les objets temps réel (OTR) représentent les entités de la BDTR [4] [11]. Ils modélisent l'environnement dynamique du monde réel contrôlé par l'application temps réel. Notre modèle d'objet temps réel (OTR) est une extension du concept d'objet actif temps réel [17] permettant d'attacher des contraintes temporelles aux données (*cohérence absolue*, *cohérence relative*, *qualité des données*, ...) et/ou aux méthodes (*échéance*, *périodicité*, *calcul imprécis*, ...) [8] [9]. Un OTR permet la modélisation d'entités concurrentes et encapsule leurs *données* et *traitements* temps réel.

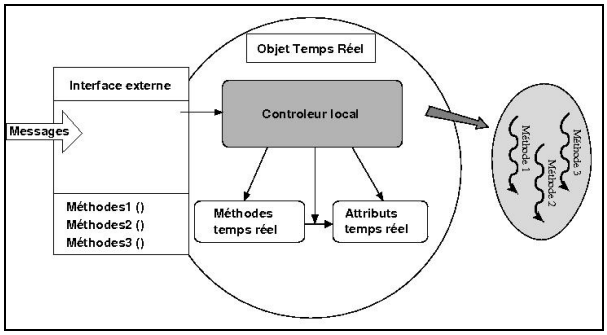

Figure 1 : Un objet temps réel

Comme l'illustre la figure 1, nous présentons l'OTR selon quatre composants : (i) un ensemble d'attributs pour encapsuler les données (classiques, temps réel), (ii) un ensemble de méthodes temps réel, (iii) une file d'attente et (iv) un contrôleur local.

Au niveau exécution, l'OTR reçoit des messages dans sa file d'attente, ce qui stimule son contrôleur local. Ce dernier vérifie les contraintes temporelles attachées aux messages et sélectionne le message ayant l'échéance la plus proche. Le contrôleur local vérifie alors les contraintes de concurrence et d'état en fonction des autres méthodes en cours d'exécution dans l'OTR. Un contrôleur global à l'application ordonnance les demandes de service de tous les OTR d'une application avec une politique d'ordonnancement globale. Chaque OTR est considéré comme une ressource de calcul autonome, capable de manipuler plusieurs messages en même temps. En effet, elle peut se doter d'un fil d'exécution pour chacun des messages qu'elle doit traiter, sous réserve que les contraintes de concurrence et d'état l'autorisent. Dans ce cas, chaque message est associé, durant son traitement, à un fil d'exécution qui lui est propre.

Nous présentons dans ce qui suit, les caractéristiques de chacun des composants de l'OTR.

3.1. Les attributs temps réel

Dans nos travaux, le modèle de données est basé sur le modèle de Ramamaritham proposé dans [14]. Nous associons à ce modèle la notion de *QdD*[4] introduite dans [2]. Ainsi, chaque donnée temps réel d est caractérisée par le quadruplet (d_{valeur}, $d_{estampille}$, d_{dva}, d_{mde}) [9], où d_{valeur} est la valeur actuelle de la donnée d, $d_{estampille}$, l'instant où cette valeur a été mesurée ou calculée, d_{dva} est la durée de validité absolue de la donnée d et d_{mde} est l'erreur maximale tolérée entre la valeur réelle et la valeur de l'attribut stockée dans la base. Nous caractérisons ainsi un OTR par deux types d'attributs permettant la gestion de ces données :

- Attribut sensoriel : On appelle attribut sensoriel, un attribut destiné à la mémorisation d'une donnée sensorielle qui doit être mise à jour régulièrement afin de refléter le plus fidèlement possible l'état du monde réel qu'elle représente [9]. Chaque valeur d'un attribut sensoriel possède une durée de validité pendant laquelle elle peut être utilisée de manière efficace. En dehors de cet intervalle, cette valeur devient obsolète (non fraîche). Pour le cas du « *Système de surveillance du trafic aérien* », nous proposons de caractériser l'OTR *Avion* par trois attributs sensoriels qui sont : *Altitude*, *Vitesse* et *Position*. Ces attributs sont rafraîchis périodiquement par des *méthodes de rafraîchissement* afin de refléter les propriétés temps réel de l'OTR *Avion*.
- Attribut dérivé : On appelle attribut dérivé un attribut destiné à la mémorisation d'une donnée dérivée [9]. Cet attribut dérivé doit être *calculé* à partir d'autres attributs sensoriels. Chaque valeur d'un attribut dérivé possède une durée de validité pendant laquelle elle peut être utilisée de manière efficace. En dehors de cet intervalle, cette valeur devient obsolète (non fraîche). L'intervalle de validité d'un attribut dérivé correspond à l'intersection des intervalles de validité des attributs sensoriels utilisés pour son calcul. Pour le cas du « *Système de surveillance du trafic aérien* », nous proposons de caractériser l'OTR *Avion* par l'attribut

[4] Qualité de données.

dérivé *Couloir*. Sa valeur est calculée à partir des valeurs des attributs de base *Position* et *Altitude*.

Chacun de ces attributs temps réel (sensoriel, dérivé) est caractérisé par le quintuplet suivant [9] : <N, CV, TS, VD, MDE>, où :

- N (***N**ame*) est le nom de l'attribut.
- CV (***C**urrent **V**alue*) contient la dernière valeur de l'attribut, correspondant à la dernière mise à jour effectuée par l'opération correspondante.
- TS (***T**ime**S**tamp*) mémorise l'estampille de la dernière mise à jour de l'attribut. Le calcul de la fraîcheur d'une donnée exige que le système enregistre l'instant de la détermination de la valeur (cela peut être l'instant de la génération de la donnée par un capteur, ou l'instant où la donnée a été écrite).
- VD (***V**alidity **D**uration*) mémorise la durée de validité de la valeur de l'attribut. Elle représente le temps durant lequel la donnée est considérée comme valide. Ce champ permet de déterminer, avec *TS*, l'intervalle de validité de l'attribut [2].
- MDE (***M**aximum **D**ata **E**rror*) mémorise l'erreur maximale tolérée entre la valeur réelle et la valeur de l'attribut stockée dans la base. Parfois, les contraintes temporelles rendent le calcul précis impossible [2]. Par conséquent, dans beaucoup d'applications, une certaine imprécision peut être tolérée. Par exemple, l'imprécision sur la valeur de l'attribut *"Vitesse"* peut être de *5 km/h*. Ce champ est de même type que le champ *CV* (**C**urrent **V**alue).

3.2. Les méthodes temps réel

Dans le modèle d'OTR, que nous proposons, nous supposons que le traitement de chaque message correspond à une *transaction temps réel* [9]. Autrement dit, chaque demande d'exécution de l'une des méthodes de l'OTR correspond à l'activation d'une transaction dans l'application. Un OTR peut donc être considéré comme un *Serveur de transactions*, car ses méthodes peuvent être appelées simultanément par différents OTR Clients. Un OTR est caractérisé par trois types de méthodes :

- Méthodes *utilisateur* : une méthode *utilisateur* déclarée au sein d'un OTR est une méthode dont les traitements sont assujettis à une contrainte temporelle, prenant souvent la forme *d'échéance*. Elle comporte des accès en mode *lecture/écriture* aux attributs classiques (non temps réel) et/ou en mode exclusif *lecture* aux attributs sensoriels de cet OTR.

  En général, une méthode *utilisateur* a une visibilité de type *public* et fait partie de l'interface de l'OTR qui l'encapsule. L'exécution d'une méthode *utilisateur* est considérée, dans nos travaux, comme une *transaction utilisateur* [9]. En outre, cette méthode se déclenche suite à une demande provenant de l'OTR lui-même ou d'un autre OTR. L'OTR appelé doit tout faire pour respecter l'échéance fixée par le client, afin que les résultats de la méthode (transaction) ne soient pas rejetés, du fait qu'ils sont considérés inutiles pour le système.

  Nous caractérisons l'OTR *Avion* par un ensemble de méthodes *utilisateur*, telle que *LireVitesse()* qui effectue une opération de *lecture* de la valeur de l'attribut sensoriel *Vitesse*, en respectant la contrainte temporelle d'échéance.
- Méthodes de rafraîchissement : une méthode de rafraîchissement déclarée au sein d'un OTR est une méthode qui accède en mode exclusif *écriture* aux attributs sensoriels et/ou attributs dérivés de cet OTR. Les traitements qu'elle effectue sont assujettis à deux contraintes temporelles qui sont *l'échéance* et *la périodicité*.

Une méthode de rafraîchissement a pour rôle la mise à jour des attributs sensoriels et des attributs dérivés. Son mode d'accès est toujours en *écriture*. L'exécution d'une méthode de rafraîchissement est considérée, dans nos travaux, comme une *transaction de mise à jour* [9]. Elle doit se terminer avant l'échéance fixée, sinon la valeur à écrire est considérée obsolète (non fraîche).

Nous caractérisons l'OTR *Avion* par un ensemble de méthodes de rafraîchissement, telle que *MaJAltitude ()* qui effectue une opération d'*écriture* de la valeur de l'attribut sensoriel *Altitude*.

3.3. La file d'attente

La file d'attente sert à stocker les messages en provenance des autres OTR. Chacun de ces messages reçus possède une échéance qu'il doit respecter sinon il sera rejeté (*firm*). Les objectifs de la file d'attente sont : (i) de redonner la main à l'OTR appelant le plus rapidement possible (si le mode d'appel le permet), car celui-ci peut avoir d'autres traitements à effectuer, (ii) d'allouer un fil d'exécution pour traiter les messages qui viennent d'arriver et réaliser toutes les opérations de contrôle associées, et (iii) de détecter les éventuelles fautes temporelles (dépassement d'échéance, notamment).

Nous associons une politique de gestion des messages aux files d'attente. Il s'agit bien souvent d'EDF (***E**arliest **D**eadline **F**irst*) [12] : les messages sont traités par ordre d'échéance temporelle. Le message possédant l'échéance la plus courte est traité en premier.

3.4. Le contrôleur local

L'objectif du contrôleur local est, d'une part, de vérifier si un OTR est en état de traiter une transaction reçue et, d'autre part, de s'assurer que toutes les transactions en cours de traitement s'exécutent sur des données cohérentes et fraîches. Il s'agit de ne pas autoriser des écritures et des lectures simultanées sur des attributs de l'OTR et de ne pas autoriser l'accès à des valeurs d'attributs non fraîches. Ces objectifs sont remplis respectivement par le *contrôleur d'état*, le *contrôleur de fraîcheur* et le *contrôleur de concurrence :*

- Le contrôleur d'état : le contrôle d'état est le premier contrôle effectué à la réception d'un message, car il est inutile de traiter les autres problèmes liés au traitement de cette transaction (la concurrence par exemple) si l'état courant de l'OTR ne permet pas l'exécution de celui-ci. Si l'état courant de l'objet ne permet pas le traitement de la transaction reçue, elle est alors remise en attente (i.e. dans la file d'attente). Si au terme de son échéance, une transaction n'est pas traitée à temps, elle sera abandonnée. Si au contraire la transaction est acceptée par le contrôleur d'état, elle passe au contrôleur de fraîcheur.
- Le contrôleur de fraîcheur : il vérifie la fraîcheur des attributs qui vont être accédés par des transactions. Si les valeurs des attributs sont obsolètes (non fraîches) alors la transaction est de nouveau mise en attente (i.e. dans la file d'attente). Cette transaction sera réveillée lorsque les valeurs des attributs seront mises à jour (si elle n'a pas ratée son échéance).
- Le contrôleur de concurrence : le concept d'OTR nous permet de spécifier deux types de concurrence [9] :
  1. *La concurrence inter-objet* : elle est réalisée en associant au moins un fil d'exécution à chaque OTR. Aucune contrainte n'est émise sur la concurrence inter-objet. En effet, les méthodes d'OTR différents peuvent s'exécuter en parallèle sans risque d'incohérence puisque les données sont locales aux OTR. La cohérence globale des données est alors assurée en interdisant le partage de données entre les OTR. Le partage de données n'est possible qu'au sein d'un même OTR.
  2. *La concurrence intra-objet* : elle permet d'exprimer le parallélisme des méthodes au niveau d'un même OTR. Ainsi, le problème de la gestion de la concurrence est réduit à un problème interne à l'OTR. Il n'apparaît que si un même OTR peut supporter simultanément plusieurs fils d'exécution.

Le contrôleur de concurrence a pour objectif de paralléliser au maximum le traitement des messages en interne d'un objet, tout en assurant la cohérence des données de l'objet. Les données de l'objet correspondent à son état, aux valeurs de ses attributs, aux rôles et aux valeurs des instances référencées par ses rôles. Afin d'être en mesure d'effectuer ces vérifications, le contrôleur de concurrence a besoin d'informations structurelles telles que l'ensemble des attributs ou rôles accédés en lecture et/ou en écriture par chacune des transactions.

Dans notre modèle, la gestion de la concurrence intra-objet temps réel est basée sur les protocoles de contrôle de concurrence et d'ordonnancement des transactions temps réel. Comme nous l'avons déjà évoqué, l'exécution d'une méthode est considérée comme étant une transaction. Au moment des accès concurrents et incompatibles des méthodes aux attributs, nous ne verrouillons pas la totalité de l'objet comme le font les objets actifs temps réel [17] [7]. Notre gestion de la concurrence s'effectue à un niveau de granularité plus fin qui est celui des attributs (données).

Nous adoptons dans nos travaux le protocole de contrôle de concurrence : 2PL-HP (***T**wo **P**hase **L**ocking **H**igh **P**riority)* [1]. Ce protocole est basé sur la résolution des conflits en tenant compte des priorités des transactions, définies à partir de leurs échéances. Le protocole 2PL-HP permet de garantir qu'une méthode avec une priorité élevée n'est pas gênée par une autre méthode de plus basse priorité. Autrement dit, les conflits sont résolus en faveur des méthodes les plus proches de leur échéance. Le principe du protocole est le suivant :

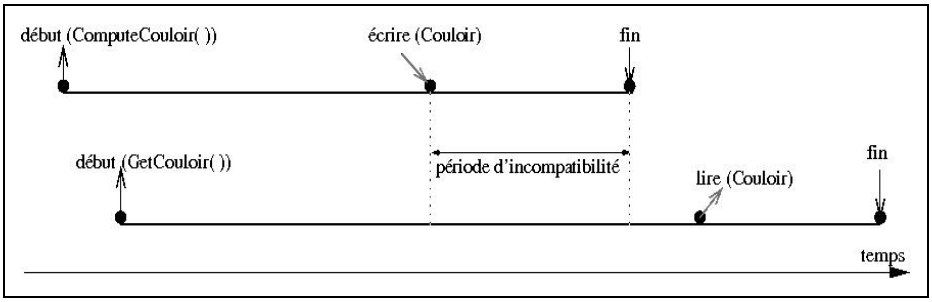

Figure 2 : Période d'incompatibilité

- Si une méthode M requiert un verrou sur une donnée déjà verrouillée (dans un mode incompatible) par d'autres méthodes de plus faible priorité, alors ces dernières sont abandonnées et M obtient le verrou et continue de s'exécuter.
- Si la priorité de M est inférieure aux priorités des autres méthodes, alors M attend que la donnée soit libérée (algorithme 2PL classique [6]).

Illustration : Comme le montre la figure 2, les méthodes *ComputeCouloir()* et *GetCouloir()* accèdent au même attribut (*Couloir*) en mode incompatible (*L/E*). Dans le domaine des STR [17] [7], il ne sera pas possible d'exécuter *GetCouloir()* avant la terminaison de la méthode *ComputeCouloir()*. Ceci repousse donc le démarrage de l'exécution de la méthode *GetCouloir()* et augmente ainsi le risque de dépassement d'échéance. Lorsque l'on analyse cet exemple, on s'aperçoit que la lecture de l'attribut *Couloir* ne se produit qu'après la terminaison de la méthode *ComputeCouloir()*. Il n'y a donc pas réellement de conflit d'accès. On pouvait donc démarrer l'exécution concurrente des deux méthodes à partir du moment où l'accès par la seconde méthode ne se fait pas pendant la *``période d'incompatibilité''* qui correspond à la période allant du début de l'écriture du *Couloir* par la méthode *ComputeCouloir()* jusqu'à la terminaison de cette même méthode (donnée verrouillée en mode écriture). De plus, si la méthode *GetCouloir()* est plus prioritaire que la méthode *ComputeCouloir()*, elle est condamnée à attendre, ce qui augmente le risque qu'elle rate son échéance.

4. Conclusion et perspectives

Dans cet article, nous avons présenté notre modèle de données temps réel orienté-objet. Ce modèle est une adaptation du modèle objet aux exigences temporelles des BDTR. Ainsi, une BDTR est constituée par un ensemble d'objets, intitulé « **O**bjet **T**emps **R**éel » (OTR) qui modélisent les entités concurrentes du monde réel et encapsulent leurs données et traitements temps réel.

Un OTR est caractérisé par des attributs temps réel (sensoriels, dérivés) qui mémorisent les propriétés temporelles de l'OTR. Les méthodes qui opèrent sur ces attributs temps réel sont assujetties à des contraintes temporelles (échéance, périodicité) qu'elles doivent respecter afin d'assurer la qualité de service de la BDTR. De plus, chaque OTR possède un contrôleur local qui assure la fraîcheur et la cohérence de ses données temps réel.

À court terme, dans nos futurs travaux, nous étudierons l'implantation de ce modèle de données sous des SGBD objets-relationnels. Une autre voie que nous explorerons est l'adaptation de ces travaux à des systèmes multimédia tels que la vidéo à la demande.

5. Remerciements

Ce travail a été effectué dans le cadre d'un projet financé par le région de Haute-Normandie (Contrat de Plan Etat-Région : CPER).

6. Références

[1] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions with Disk Resident Data. In Proc. of the 15th Intl. Conf. on Very Large Data Bases (VLDB), pages 385–396, Amsterdam, The Netherlands, 1989. Morgan Kaufmann.

[2] M. Amirijoo, J. Hansson, and S. H. Son. Specification and management of QoS in real-time databases supporting imprecise computations. IEEE Transactions on Computers, 55(3):304–319, 2006.

[3] A. Bestavros, K.-J. Lin, and S.Son. Real-Time Database System: Issues and Applications, chapter Advances in Real-Time DataBase Systems Research, pages 1–14. Kluwer Academic Publishers, 1997.

[4] L. DiPippo and V. Wolfe. Object-based semantic real-time concurrency control. In IEEE Real-Time Systems Symposium, pages 87–96, 1993.

[5] C. Duvallet, Z. Mammeri, and B. Sadeg. Les SGBD temps réel. Technique et Sciences Informatiques, 18(5):479–517, 1999.

[6] K. Eswaran, J. Gray, R. Lorie, and I. Traiger. The notions of consistency and predicate locks in a database system. Communication of the ACM, 19(11):624–630, 1976.

[7] S. Gerard, C. Mraidha, F. Terrier, and B. Baudry. A UML-based concept for high concurrency: the real-time object. In Proceedings of 7th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2004), pages 64–67, 2004.

[8] N. Idoudi, C. Duvallet, B. Sadeg, R. Bouaziz, and F. Gargouri. Un modèle objet pour les SGBD temps réel. Actes du colloque international sur l'Informatique et ses Applications (IA'2006), pages 285–292, 31 Octobre au 2 Novembre 2006.

[9] N. Idoudi, C. Duvallet, B. Sadeg, R.Bouaziz, and F. Gargouri. Structural model for realtime databases : an illustration. In Proceedings of 11th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2008), à paraître (8 pages), Orlando, United State, May 5-8, 2008.

[10] W. Kim. Object-Oriented Database Systems: Promises, Reality, and Future. Modern Database Systems, pages 255–280. Addison Wesly, 1995.

[11] Y.-K. Kim and S. H. Son. Predictability and consistency in real-time database systems. In Advances in real-

time systems, pages 509–531, Upper Saddle River, NJ, USA, 1995. Prentice-Hall, Inc.

[12] C. Liu and J. Leyland. Scheduling algorithms for multiprogramming in hard real-time environment. Journal of the ACM, 1(20):46–61, 1973.

[13] D. Locke. Applications and system characteristics. In Real-Time Database Systems: Architecture and Techniques, pages 17–26. Kluwer Academic Publishers, 2001.

[14] K. Ramamritham. Real-Time Databases. Journal of Distributed and Parallel Databases, 1(2):199–226, 1993.

[15] K. Ramamritham, S. Son, and L. DiPippo. Real-Time Databases and Data Services. Real-Time Systems, 28:179–215, 2004.

[16] J. Stankovic, S. Son, and J. Hansson. Misconceptions about real-time databases. Computer, 32(6):29–36, 1999.

[17] F. Terrier, G. Fouquier, D. Bras, L. Rioux, P. Vanuxeem, and A. Lanusse. A real time object model. In TOOLS Europe96, Paris, France, 1996.