Un modèle objet pour les SGBD temps réel

Nizar Idoudi, Claude Duvallet, Bruno Sadeg
LITIS, UFR des Sciences et Techniques,
25 rue Philippe Lebon, BP 540,
F-76058 Le Havre Cedex,
{*Prenom.Nom*}*@univ-lehavre.fr*

Faiez Gargouri, Rafik Bouaziz
MIRACL-ISIMS,
BP 1030, Sfax 3018,
TUNISIE
{*Faiez.Gargouri,Raf.Bouaziz*}*@fsegs.rnu.tn*

Abstract

L'essor des travaux sur les SGBD temps réel nous amène à envisager le problème de la conception des applications reposant sur ces travaux. Pour les applications classiques, l'utilisation du langage UML est très répandu. Pour la prise en considération des contraintes temporelles dans les applications temps réel, des extensions ont été apportées à UML afin d'aboutir à la proposition d'UML temps réel. Maintenant, il semble nécessaire de proposer des méthodes de conception et de spécification des applications reposant sur l'utilisation des SGBD temps réel. Dans cet article, nous proposons donc un modèle objet pour les SGBD temps réel. Ce modèle a pour vocation de fournir des outils pour permettre la conception d'applications reposant sur l'utilisation de SGBD temps réel.

1 Introduction

Les applications deviennent de plus en plus sophistiquées en termes de traitements effectués et de services rendus à l'utilisateur. La complexité des applications rend nécessaires l'utilisation de méthodes de conceptions extrêmement rigoureuses et efficaces qui permettent d'augmenter l'efficacité et la rapidité de conception, et de faciliter la maintenance des applications.

Depuis plusieurs années, les méthodes de conception ont été unifiées au moyen du langage UML 2.0 [1]. La spécification de la plupart des applications est possible grâce à l'utilisation des différents diagrammes et modèles fournis par UML.

Certaines applications nécessitent d'effectuer des traitements avant des dates fixes (échéances) au delà desquelles le traitement peut soit devenir caduque soit, avoir des conséquences graves. Ces applications sont dites temps réel et elles sont caractérisées par l'utilisation de contraintes temporelles, fixant les échéances de certains de leurs traitements.

Des extensions temps réel ont été apportées pour UML afin de prendre en considération la spécification des applications temps réel. Plus particulièrement, l'introduction de notations (ou de concepts) spécifiques pour exprimer les contraintes temporelles.

Depuis la fin des années 80, des travaux concernant les SGBD temps réel (SGBDTR) ont débuté afin de prendre en considération la gestion de quantités volumineuses de données au sein des applications temps réel. En effet, il s'agit dans ces applications d'exploiter les mécanismes présents au sein des SGBD pour la gestion efficace des données.

Malheureusement, ces travaux souffrent de l'absence de moyens de spécification des applications utilisant des SGBDTR. Une voie possible est celle de l'extension du langage UML qui permet déjà de tenir compte des contraintes temporelles des applications (tâches) mais qui ne considère pas la gestion des bases de données temps réel.

Dans cet article, nous proposons un modèle objet pour la conception d'applications reposant sur l'utilisation de SGBDTR.

Dans la section 2, nous décrivons les caractéristiques des systèmes temps réel avant de présenter les extensions UML qui ont été introduites pour leur conception (cf. section 3). Dans la section 4, nous décrivons les principales caractéristiques du modèle de SGBDTR sur lequel nous basons nos travaux. Dans la section 5, nous dressons un bilan sur les différences entre les systèmes temps réel et les SGBDTR afin d'extraire les caractéristiques essentielles qui doivent être reprises pour la réalisation de notre proposition concernant un modèle objet pour les SGBDTR (cf. section 6). Nous concluons sur l'apport de notre proposition et nous donnons quelques perspectives à notre travail.

2 Les systèmes temps réel

Généralement, un système de contrôle temps réel est un système connecté à un procédé physique externe dont il doit contrôler et commander le comportement. Comme l'illustre la figure 1, un système temps réel (STR) se décompose alors en sous systèmes qui sont : le système de contrôle et le procédé contrôlé.

– Le système de contrôle est principalement composé d'un calculateur temps réel et d'interfaces d'entrées/sorties servant à la communication avec le procédé à contrôler.
– Le procédé à contrôler est quant à lui composé du procédé physique externe ainsi que de capteurs et d'actionneurs.

Les capteurs vont scruter les données du procédé physique et les transmettre au système de contrôle sous forme de mesures ou d'événements. Le système de contrôle va alors effectuer les calculs, en respectant les propriétés temporelles, sur ces données et transmettre des commandes aux actionneurs du procédé à contrôler. Enfin, les actionneurs vont agir sur le procédé physique pour réaliser ces commandes. Le procédé contrôlé est souvent appelé l'environnement du système de calcul temps réel. Toutes les définitions de systèmes temps réel mettent en avant la notion de temps de réponse, c'est-à-dire le temps que met le système pour fournir les résultats à partir des données d'entrée. *"Un calculateur temps réel est un système de calcul pour lequel la validité de son comportement dépend non seulement des résultats des calculs, mais également de l'instant physique de production de ces résultats."* [4].

3 UML2 et le temps réel

Bien que UML2 possède en standard des concepts de base pouvant être utilisé pour modéliser certaines caractéristiques particulières des applications temps réel, il ne satisfait pas complètement les besoins de modélisation de ces catégories de systèmes. Pour combler cette lacune, un profil UML spécifique au temps réel a été spécifié par l'OMG, il s'agit du profil "SPT - *Scheduling, Performance and time*" [7]. Ce profil défini un ensemble minimal d'extensions UML (i.e. stéréotypes, valeurs marquées et contraintes) nécessaires à l'analyse, la modélisation et l'implémentation d'applications temps réel. En complément au SPT, l'OMG a démarré une réflexion plus détaillée sur le concept de qualité de service (QdS) au travers d'un nouveau profil UML, le profil pour la *"Quality of Service and Fault Tolerance"* [6].

3.1 Les mécanismes d'extension d'UML

Les mécanismes permettant de spécialiser et/ou d'étendre UML pour des besoins spécifiques reposent sur trois concepts spécifiques :

1. Les stéréotypes qui permettent en fait de spécialiser un concept existant du méta-modèle de UML (méta-classe). Un stéréotype est un moyen pour changer la sémantique d'un concept existant. De plus, un stéréotype peut également posséder des valeurs marquées et des contraintes (élément décrit ci-après).
2. Les valeurs marquées (*"tagged value"*) qui sont un moyen d'attacher des propriétés supplémentaires (i.e. n'existant pas au niveau du méta-modèle de UML) à un concept de UML.
3. Les contraintes qui permettent de spécifier une nouvelle sémantique à un élément du méta-modèle de UML via une expression s'appuyant sur un langage de contrainte donné. Ce langage peut être un langage conçu spécifiquement pour l'expression de contraintes comme OCL (*Object Constraint Language*) [11], un langage de programmation, une notation mathématique ou bien encore le langage naturel comme pour une partie de la description de la sémantique de UML.

Ces trois mécanismes d'extension permettent donc d'adapter UML à un domaine d'application particulier, soit en y ajoutant de nouveaux concepts, soit en spécialisant des concepts existants.

3.2 Le profil "Scheduling, Performance and Time"

Afin de couvrir les besoins et les spécificités de l'analyse temps réel des modèles UML, l'OMG propose le profil UML SPT [7]. L'intérêt principal de cette norme est qu'elle fournit les moyens d'annoter un modèle avec des caractéristiques de QdS [9] permettant d'effectuer des analyses quantitatives. Il est alors possible de vérifier et de valider des propriétés extra fonctionnelles comme les temps de réponse ou les tailles des files d'attente en se basant sur des données comme les échéances, les pires temps d'exécution ou les politiques d'ordonnancement. Le profil SPT se compose de trois paquetages principaux (cf. figure 2).

1. Le paquetage GRMF (*General Resource Modeling Framework*) définit un cadre générique pour supporter n'importe quel type d'analyse de modèle. Il est lui-même constitué de trois sous-paquetages dédiés à la modélisation des ressources (GRM pour *General Resource Modeling*), de la concurrence (GCM pour *General Concurrency Modeling*) et du temps (GTM pour *General Time Modeling*). Ces paquetages spécifient respectivement les moyens de modéliser la QdS des ressources en considérant les propriétés physiques du logiciel et du matériel support d'exécution d'une application temps réel, de définir le concept d'unité concurrente et de représenter les concepts liés au temps.

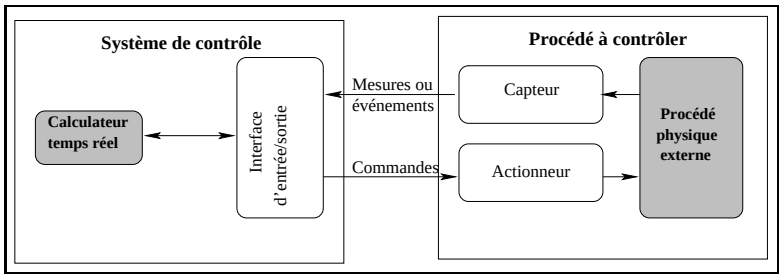

FIG. 1 – Architecture d'un système temps réel.

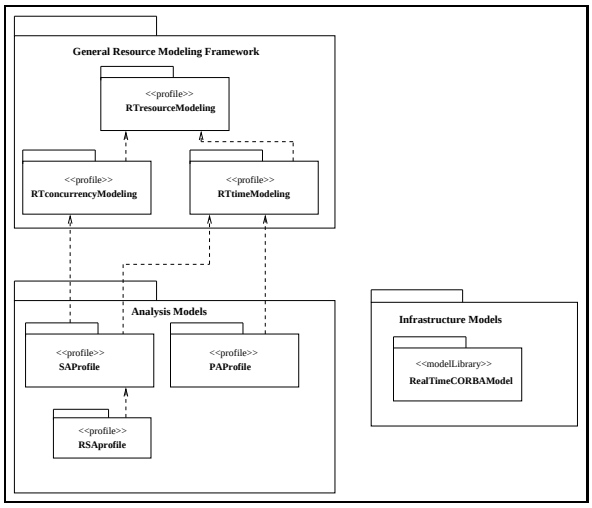

FIG. 2 – Scheduling, Performance and Time.

2. Le second grand paquetage du SPT (*Analysis Models*) spécialise le paquetage précédant en utilisant les concepts génériques de façon à définir les moyens nécessaires à une analyse d'ordonnançabilité, d'une part, et d'autre part, à une analyse de performance.
3. Le dernier paquetage (*Infrastructure Models*) est dédié à la description de la plate-forme d'exécution "CORBA temps réel".

Dans la suite de cette section, nous décrivons en détails des constituants du GRMF :

1. Le modèle de ressource générale (*CoreResourceModel*) : il est basé sur le paradigme client-serveur [9], c'est-à-dire qu'un client demande un service de la part d'un serveur, cette demande est appelée (*ResourceUsage*), en spécifiant un ensemble d'exigences non-fonctionnelles (QdS requises) et le serveur fournit des services caractérisés par des exigences non-fonctionnelles offertes (QdS offertes). La relation entre les serveurs et les clients est définie par un contrat de QdS (cf. figure 3).
2. Le modèle de causalité (*CausalityModel*) : il est la base de toutes les descriptions dynamiques associées au profil. Il traduit les chaînes de causes à effets du comportement des instances d'une application. Il est basé sur le concept d'occurrence d'événement.
3. Le modèle d'utilisation de ressource (*ResourceUsageModel*) : il décrit comment un ensemble de clients utilise un ensemble de ressources et leurs services associés. On distingue deux types d'usages :
   (a) L'usage statique : il suffit de spécifier quels sont les ressources et services sous-jacents utilisés sans entrer dans les détails de l'usage. Ainsi, un client spécifie uniquement les relations structurelles qu'il entretient avec les ressources. S'il s'agit d'un type (classe, composant,...), il peut spécifier par exemple une association ou une dépendance de type usage. S'il s'agit d'une instance (objet, instance de composant...), il peut spécifier un lien vers les instances ressources ou les services de ressources.
   (b) L'usage dynamique : il est utilisé pour décrire finement la relation du client vis-à-vis de ses ressources. Un usage dynamique est alors considéré comme un scénario, et plus particulièrement, comme une suite ordonnée d'exécutions d'actions. Cette suite se conforme au modèle prédécesseurs-successeurs, avec la possibilité d'avoir des prédécesseurs et des successeurs multiples et concurrents.
4. Le modèle des types de ressource (*ResourceTypes*) : Le GRM propose trois taxonomies possibles des ressources :
   (a) En fonction de leur but, on considère les ressources de types processeur (physique ou virtuel) qui désignent les entités capables de stocker et d'exécuter le code d'un programme, les ressources de communication qui permettent aux autres ressources d'échanger de l'information et les périphériques qui regroupent les ressources qui ne sont ni des processeurs, ni des médias de communication.

(b) En fonction de leur nature, active ou passive : une ressource active est capable de générer des stimulus de son propre chef sans être sollicitée par un stimuli explicite extérieur (exemple : une tâche d'un système d'exploitation, un composant matériel...). Une ressource qui n'est pas active est alors dite passive.

(c) En fonction de leur protection associée : les instances de ressources protégées proposent au moins une instance de service d'exclusivité. Cette dernière restreint l'accès concurrent à la ressource en spécifiant une politique de contrôle d'accès. Une ressource non protégée est une ressource qui ne propose aucun mécanisme de protection d'accès.

5. Le modèle de gestion des ressources (*ResourceManagement*) : il définit le cadre de modélisation des différents types de services de gestion des ressources communément trouvés dans les systèmes d'exploitation. Ce modèle identifie deux concepts en particulier :

   (a) Le courtier : il est en charge d'allouer et de libérer un ensemble d'instances de ressources utilisées par un client en fonction d'une politique d'accès aux ressources.

   (b) Le gestionnaire : il est en charge de la création des ressources et en général de leur cycle de vie.

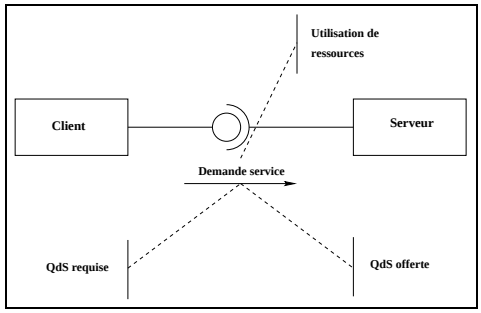

FIG. 3 – Le paradigme Client/Serveur.

3.3 Le profil QoS&FT

Le profil qualité de service et tolérance aux fautes (QoS&FT - *Quality of Services & Fault Tolerance*) a été adopté en septembre 2004 par l'OMG [6].

Ce profil permet de prendre en compte les différentes caractéristiques de qualité de service et de tolérance aux fautes d'un système. Ces deux notions sont traitées de façon disjointe dans le profil.

Les exemples de caractéristiques de qualité sont la périodicité, l'irrégularité, les limites, la gigue, les échéances ou encore le type d'une exigence (dure, moyenne, souple). Les niveaux de qualité expriment des degrés de satisfaction d'une propriété non fonctionnelle. Ils sont exprimés au travers de contrats qui sont eux-mêmes associés aux caractéristiques de QdS.

Dans ce profil, un méta-modèle est spécifié. Il permet de définir un concept abstrait d'un langage de modélisation supportant des concepts de modélisation de QdS. On ne définit pas de syntaxe concrète à ce niveau.

Le méta-modèle dédié à la QdS est composé de trois paquetages. Le premier paquetage spécifie les caractéristiques de QdS (*QoSCharacteristics*). Il donne les bases permettant de définir les contraintes de QdS dans un second paquetage (*QoSConstraints*). Ce second paquetage peut ensuite être utilisé dans les contrats qui servent finalement à définir les niveaux de QdS, concept introduit dans le troisième et dernier paquetage (*QoSLevels*).

Pour situer ce profil par rapport à SPT (*Schedulability, Performance and Time*), les définitions des caractéristiques et des valeurs de QdS dans le profil QoS&FT sont des spécialisations des définitions de ces notions introduites dans le profil SPT. Les contrats sont des relations exprimées entre les QdS offertes et requises. Les relations entre les ressources et la QdS sont plus spécialement adressées dans le profil SPT.

L'intégration de ces notions au sein de UML se fait par la définition de profils UML associés aux différents paquetages décrits précédemment.

On définit également dans ce profil comment traiter et décrire la gestion des risques et la tolérance aux fautes. Pour cette dernière, la solution proposée s'appuie principalement sur des systèmes complexes et généralement distribués avec des exigences de robustesse. La façon de traiter la tolérance aux fautes dans ce profil est uniquement liée à la robustesse d'un système. Les bases de ce profil sur la tolérance aux fautes sont la réplication des objets et des composants et la détection des fautes.

4 L'approche "qualité de service" dans les SGBD temps réel

Dans la littérature, on considère le plus souvent que les SGBDTR se trouvent en mémoire principale. Cela permet de s'abstraire de la problématique des entrées/sorties et des écritures différées sur disque. Un SGBDTR repose sur l'utilisation d'une base de données temps réel qui peut à la fois contenir des données classiques (dites non temps réel) et des données temps réel. Comme pour les SGBD classiques les unités de traitement sont les transactions

mais elles peuvent posséder des contraintes temporelles qui en font alors des transactions temps réel.

4.1 Les données temps réel

Les données temps réel représentent la capture de l'état du monde réel. Par exemple, dans une centrale nucléaire, on peut trouver des capteurs de température qui sont utilisés pour contrôler le système et détecter les éventuelles anomalies (surchauffe du système ou autre). Dans les systèmes boursiers, des transactions mettent à jour les prix des actions à un instant donné. Ces données doivent être mises à jour régulièrement afin de refléter au plus près le monde réel. Elles possèdent donc une durée de validité qui représente la période pendant laquelle elles peuvent être utilisées. L'état de l'environnement tel qu'il est perçu par le système contrôleur doit être reflété le plus fidèlement possible. Cette exigence a une conséquence sur la conception du SGBD temps réel, qui doit non seulement respecter les contraintes d'intégrité (cohérence logique), mais aussi respecter les contraintes temporelles des données. La cohérence temporelle pour une donnée regroupe en réalité deux notions :

- la cohérence absolue qui est liée directement au fait que l'image de l'environnement dans le système doit refléter l'état réel de cet environnement,
- la cohérence relative qui est liée aux données utilisées pour dériver d'autres données dans la base qui doivent être cohérentes entre elles.

Pour illustrer ces deux notions, reprenons la définition d'une donnée temps réel d proposée dans [8] : d = (d_{valeur}, $d_{estampille}$, d_{dva}), où d_{valeur} est la valeur actuelle de la donnée d, $d_{estampille}$ représente l'instant où cette valeur a été mesurée ou calculée et d_{dva} est la durée de validité absolue de la valeur de d. Un ensemble de cohérence relative R est l'ensemble des données utilisées pour dériver une nouvelle donnée. À tout ensemble R est associée une durée de validité relative R_{dvr}. Soit d∈R, on dit que d est dans un état correct, si et seulement si :

1. d_{valeur} est logiquement cohérente (elle satisfait aux contraintes d'intégrité),
2. d est temporellement cohérente :
   - cohérence absolue : (instant_courant - $d_{estampille}$) $\leq d_{dva}$.
   - cohérence relative : pour tout $d' \in R$, $d_{estampille}$ - $d'_{estampille} \leq R_{dvr}$.

4.2 Les transactions temps réel

Une transaction temps réel effectue généralement une suite d'opérations de lecture ou d'écriture des données temps réel ou non temps réel en respectant une échéance (contrainte temporelle). Les transactions temps réel considérées dans les SGBDTR pour la QdS possèdent des échéances de type firm (échéances strictes non critiques) [2]. Par conséquent, 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 de base. 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.

5 STR versus les SGBDTR

Les systèmes temps réel sont bien adaptés pour la gestion des contraintes temporelles des applications. L'unité de traitement dans les STR est la tâche temps réel qui permet de prendre en considération les contraintes temporelles des applications. Ces systèmes peuvent utiliser des données temps réel.

En l'absence de bases de données, il est nécessaire pour chaque accès en lecture à la valeur d'un capteur de faire d'abord son acquisition. En effet, ces valeurs ne sont pas stockées par le système lorsqu'elles sont acquises et par conséquent, lorsque le nombre de capteurs (i.e. données temps réel) augmente de façon considérable ou alors que le nombre d'accès en lecture aux données temps réel augmente le nombre d'acquisitions de ces données devient très coûteux en temps.

Pour la gestion des données en grande quantité et la prise en compte des contraintes temporelles, les SGBDTR sont plus adaptés car ils possèdent les mécanismes adéquats pour utiliser des bases de données temps réel. Dans les SGBDTR, l'unité de traitement est la transaction temps réel.

Dans un SGBDTR, il est nécessaire d'avoir recours à des transactions de mises à jour des données temps réel (issues de capteurs) afin de disposer d'une base de données temps réel qui soit toujours temporellement cohérente (i.e. fraîche). Contrairement aux STR, dans les SGBDTR, on utilise donc une base de données qui va servir à stocker les données et à optimiser les accès à ces données.

Lorsqu'on utilise une base de données, on effectue l'acquisition des données temps réel de façon périodique au moyen de transactions de mise à jour et on stocke ces données. Ensuite, les accès se font directement au niveau de la base de données et on peut disposer de tous les moyens d'optimisation des accès

présents dans les SGBD. Il n'est plus nécessaire pour accéder à une donnée temps réel d'effectuer son acquisition directement auprès du capteur correspondant.

La différence essentielle entre les STR et les SGBDTR réside donc dans leur capacité à traiter des volumes très importants de données temps réel et à optimiser la gestion de ces données ainsi que l'accès à ces données.

Nous avons vu dans la section 3 qu'il existait des profils pour la conception des applications temps réel. Toutes ces approches sont particulièrement bien adaptées pour la spécification des contraintes temporelles. Par contre, ces approches de conception ne permettent pas de prendre en considération la conception de bases de données temps réel car elles ne gèrent pas le maintien de la fraîcheur des données (i.e. les contraintes temporelles des données stockées dans la base de données temps réel).

6 Un modèle objet pour les SGBD temps réel

Les applications temps réel sont conçues pour contrôler des systèmes qui sont par nature parallèles. Ce parallélisme est modélisé via le concept d'Objet Temps Réel (OTR) [10], [3]. L'OTR permet la modélisation d'entités concurrentes encapsulant les contrôles de concurrence et d'état. Il s'agit d'une extension de celui d'objet actif de UML où il est possible d'attacher des contraintes temps réel aux traitements des messages effectués par les objets actifs de l'application. Ainsi, nous présentons l'OTR selon quatre composants (cf. figure 4) :

1. un ensemble d'attributs pour encapsuler les données (classiques, temps réel) ;
2. un ensemble d'opérations (comportement) ;
3. une file d'attente ;
4. et un contrôleur local d'état et de concurrence.

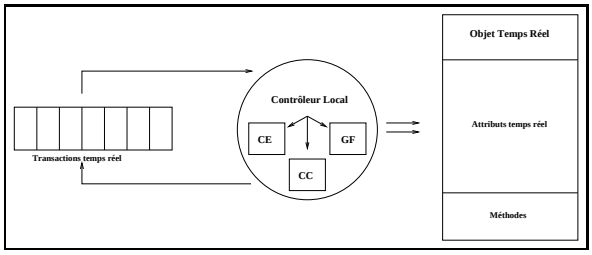

FIG. 4 – Un objet temps réel.

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 opérations en cours d'exécution dans l'OTR. Il peut ainsi être vu comme un serveur de tâches. Au niveau exécution, chaque demande d'exécution d'une de ses opérations (service) correspond alors à l'activation d'une *transaction temps réel* dans l'application. Un contrôleur global à l'application ordonnance les demandes de service de tous les OTR d'une application avec une politique d'ordonnancement globale.

6.1 La communication

Un OTR est capable de recevoir et d'émettre des messages qui peuvent être de deux natures : un signal ou un appel.

1. la communication par signal : La communication par signal est issue de l'action *BroadcastSignalAction*. Ce type de communication est asynchrone et se fait en mode diffusion. L'OTR récepteur d'un signal ignore l'identité du son émetteur, et l'émetteur ignore l'identité du ou des récepteurs (il n'est donc pas nécessaire d'avoir un lien structurel entre l'émetteur et le(s) récepteur(s) d'un signal). L'ensemble des objets cibles pour un signal donné est constitué de tous les objets ayant déclaré le signal en réception. Les signaux sont des classifier, au sens UML, et ils peuvent ainsi posséder des attributs. Ceux-ci définissent les paramètres d'un signal.
2. la communication par appel d'opération : Un appel d'opération correspond à l'exécution d'une action de type *CallOperationAction*. Ce type d'action possède un attribut *IsSynchronous* spécifiant le mode d'appel qui peut être synchrone ou asynchrone.
   (a) les appels synchrones sont obligatoires dans le cas d'appel d'une opération avec valeur de retour. Dans ce type d'appel, l'appelant attend que l'objet appelé exécute l'opération et lui retourne le résultat de l'exécution.
   (b) les appels asynchrones correspondent à la situation où l'appelant n'attend rien de l'appelé et continue son exécution.

   Qu'ils soient synchrones ou asynchrones, les appels d'opérations peuvent posséder des contraintes temps réel.

Dans ce papier, les messages (appel d'opération, envoi de signal) sont considérés comme étant des transactions (utilisateur/mise à jour).

6.2 La file d'attente

La file d'attente sert à stocker les transactions en provenance des autres objets temps réel ou des

transactions de mise à jour en provenance des capteurs. Chacune de ces transactions reçues possède une échéance qu'elle doit respecter sinon elle sera rejetée (*firm*). Les objectifs de la file d'attente sont :

1. redonner la main à l'objet appelant le plus rapidement possible (si le mode d'appel le permet), car celui-ci peut avoir d'autres traitements à effectuer.
2. allouer un fil d'exécution pour traiter les transactions qui viennent d'arriver et réaliser toutes les opérations de contrôle associées.
3. détecter les éventuelles fautes temporelles (dépassement d'échéance par exemple).

Une politique de gestion des messages en attente est associée à la file d'attente, il s'agit bien souvent de EDF (*Earliest Deadline First*) [5] : les messages sont traités par ordre d'échéance temporelle. Les messages possédant l'échéance la plus courte sont traités en premier.

6.3 Le contrôleur local

L'objectif du contrôleur local est, d'une part, de vérifier si un objet est en état de traiter une transaction reçu et, d'autres 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 propriétés de l'objet et de ne pas autoriser l'accès à des données (attributs) non fraîches. Ces objectifs sont respectivement remplis par le contrôleur d'état, le gestionnaire de fraîcheur et le contrôleur de concurrence :

1. Le contrôleur d'état : L'aspect contrôle du modèle de comportement des objets temps réel est spécifié par une machine d'états-transitions de UML. A un instant donné, l'objet est caractérisé par un état donné, nommé *état courant* de l'objet. La problématique du contrôle d'état est de vérifier si un message donné peut déclencher le tirage d'une transition ou non.
   Le contrôle d'état est le premier contrôle effectué sur la réception d'une transaction, 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'objet 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 la fraîcheur.
2. Le gestionnaire de fraîcheur : Il vérifie la fraîcheur des données (attributs)qui vont être accédé par des transactions. Si les données 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 données seront mises à jour (certainement si elle n'a pas ratée son échéance).
3. Le contrôleur de concurrence : 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, les valeurs de ses attributs, des rôles et les 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. Concernant la concurrence, le langage UML propose trois possibilités via le méta-attribut *concurrency* de la métaclasse *BehavioralFeature*. Ce méta-attribut est de type *CallConcurrencyKind* qui représente une énumération pouvant prendre une des trois valeurs suivantes :

   (a) *sequential :* signifie qu'aucun mécanisme de contrôle de concurrence n'est associé à l'opération. C'est au concepteur d'éviter les conflits en s'assurant de la coordination des différents appelants potentiels.

   (b) *guarded :* signifie que différents fils d'exécution peuvent appeler simultanément des opérations gardées d'un objet. Cependant, une seule est autorisée à s'exécuter et les autres sont bloquées en attendant la fin de l'exécution en cours. C'est au concepteur de gérer les cas d'inter blocages.

   (c) *concurrent :* signifie que différents fils d'exécution peuvent appeler simultanément les opérations d'une instance et s'exécuter de manière concurrente.

   Ces concepts, proposés par le standard UML, ne sont pas adaptés à notre modèle d'objet temps réel. En effet, les objets actifs de UML ne nécessitent pas le traitement de la concurrence interne à cause de l'hypothèse RTC « Run-To-Completion » du modèle d'exécution des machines à états-transitions de UML qui résulte d'une sérialisation des traitements internes d'un objet actif.

Notre objet temps réel s'appuie sur un modèle d'exécution interne multi-tâches qui nécessite de mettre en place un mécanisme de gestion de la concurrence d'exécution des opérations d'un objet [3]. Pour cela, un stéréotype *Concurrency* a été appliqué sur les opérations et ayant la propriété mode typé par l'énuméré *concurrencyMode* et pouvant prendre une des valeurs suivantes :

(a) read : signifie que l'opération peut utiliser les attributs ou rôles de l'objet mais uniquement en lecture. Cette opération peut donc être exécutée avec d'autres opérations utilisant ces mêmes rôles et/ou attributs en lecture seule.

(b) write : signifie que l'opération peut utiliser les attributs ou rôles de l'objet en écriture. Cette opération ne peut donc pas être exécutée en parallèle avec d'autres opérations utilisant ces mêmes rôles et/ou attributs (en lecture ou en écriture).

(c) parallel : permet à un utilisateur de marquer une opération de façon à ce qu'elle puisse court-circuiter le contrôle de concurrence. Une opération dont l'exécution est considérée comme atomique par le développeur peut être spécifiée de type parallèle. Cette option permet d'optimiser les traitements internes d'un message.

6.4 Le contrôleur global

Une fois le contrôle terminé, un message est généré pour signifier qu'il n'existe ni problème d'état, ni problème de concurrence avec les messages en cours d'exécution de l'objet. Ce message est alors transmis au contrôleur global de l'application. Ce contrôleur gère la concurrence inter-objets temps réel et ordonnance l'exécution des messages de l'application dans un plan d'ordonnancement global à l'application.

7 Conclusion et perspectives

Dans cet article, nous avons effectué une étude comparative des SGBD temps réel et des systèmes temps réel afin de présenter un modèle objet pour les SGBD temps réel. Ce modèle objet permet de gérer des transactions temps réel (par le biais de messages inter-objets) et de prendre en considération les données temps réel (par le biais de la notion d'attributs temps réel pour les OTR). Pour ces dernières, un second aspect a été pris en compte : il s'agit de la mise à jour des données par le biais d'opérations (de mise à jour).

Dans la suite de nos travaux, il sera nécessaire de prendre en considération la modélisation des intervalles de validité des transactions et des données temps réel dans notre modèle. Il s'agira aussi de définir la manière de spécifier les contraintes d'intégrité (cohérence absolue) et les contraintes temporelles (cohérence temporelle) pour les bases de données temps réel.

En nous appuyant, sur le profil UML QoS&FT, nous envisageons de définir un modèle objet permettant de spécifier la gestion de la qualité des données, de la qualité des transactions et de la qualité de services dans les SGBD temps réel. Il s'agira essentiellement de définir le mode de présentation des contraintes temps réel au moyen du langage OCL (Object contraint language).

Références

[1] G. Booch, I. Jacobson, and J. Rumbaugh. *UML 2.0.* CampusPress, 2004.

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

[3] S. Gérard. The ACCORD/UML methodology. DTSI/SLA/02-326, 2003.

[4] H. Kopetz. *Real-Time Systems : Design Principles For Distributed Embedded Applications.* Kluwer Academic Publishers, 1997.

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

[6] OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms. ptc/04-09-01, 2004.

[7] OMG. UML Profile for Schedulability, Performance and Time. formal/05-01-02, 2005.

[8] K. Ramamritham. Real-time databases. In *Proceedings of the* 14^{th} *International VLDB Conference*, 1993.

[9] B. Selic. *UML for Real : Design of Embedded Real-Time Systems*, chapter Modeling Quality of Service with UML. Kluwer Academic Publishers, 2003.

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

[11] J. Warmer and A. Kleppe. *The Object Constraint Language : Precise Modeling with UML.* Addison Wesley, 1999.