Vers une méthode de conception des bases de données temps réel

Nizar Idoudi, Claude Duvallet, Bruno Sadeg, Faiez Gargouri

LITIS, UFR des Sciences et Techniques, 25 rue Philippe Lebon, BP 540, F-76058 LE HAVRE CEDEX,
{Nizar.Idoudi,Claude.Duvallet,Bruno.Sadeg}@univ-lehavre.fr, Faiez.Gargouri@fsegs.rnu.tn

Résumé – *L'objet de cet article consiste d'une part de présenter rapidement les SGBDTR*[1] *et, d'autre part à décrire les capacités intrinsèques de UML à modéliser des applications temps réel (TR). Nous proposons ensuite quelques pistes pour modéliser les applications reposant sur des SGBDTR en exploitant les principes de modélisation orientée objet d'UML et en introduisant la notion de contraintes temps réel.*

Mots-clés : *SGBD temps réel, systèmes temps réel, UML.*

I. INTRODUCTION

Les applications temps réel sont généralement composées d'un système contrôleur (le système informatique) et d'un système contrôlé (l'environnement de l'application). Elles se distinguent des applications traditionnelles par les contraintes temporelles qu'elles doivent respecter et qui sont matérialisées sous forme d'échéances des actions et/ou de durées de validité des données.

Certaines de ces applications manipulent des quantités importantes de données (qui sont, en général, des paramètres représentant les caractéristiques de l'environnement) ; l'utilisation d'un SGBD (Système de Gestion de Bases de Données) peut s'avérer nécessaire pour gérer ces données de manière efficace. Cependant, les SGBD traditionnels ne permettent pas de répondre aux besoins de ces applications car ils n'intègrent pas de mécanismes qui permettent de prendre en compte les contraintes temporelles [3] [6], leur objectif en termes de performances étant de minimiser le temps de réponse moyen des transactions. Contrairement à la gestion d'une base de données traditionnelle, les données dans une base de données temps réel doivent être gérées de telle façon qu'elles soient non seulement cohérentes de manière logique (c'est-à-dire qu'elles doivent satisfaire aux règles d'intégrité), mais aussi du point de vue temporel [9].

Depuis la fin des années 80, des travaux sur une nouvelle génération de SGBD ont commencé à voir le jour [1] [8] : ce sont les SGBD temps réel (SGBDTR).

Dans cet article, nous commençons par présenter les SGBD temps réel : définitions et caractéristiques. Dans la section III., nous présenterons brièvement les constructions UML permettant de modéliser les aspects temps réel d'une application. Dans la section IV., nous définirons la problématique de la conception d'applications reposant sur des bases de données temps réel et nous présenterons un début d'approche pour résoudre celle-ci. Enfin, nous conclurons cet article sur le travail effectué et les objectifs de nos travaux futurs.

[1] SGBD temps réel

II. SGBDTR : DÉFINITION ET CONTEXTE

II.1. Les STR : définition et limites

Dans de nombreuses applications temps réel, l'objectif est la gestion efficace des données en respectant les contraintes temporelles qui leur sont imposées. Les systèmes adaptés pour la gestion d'applications de ce type sont les systèmes temps réel (STR), car ils disposent d'algorithmes efficaces pour l'ordonnancement des tâches temps réel de manière à respecter leurs échéances, de mécanismes de communication via des mémoires communes, etc.

Un STR est un système dont le comportement dépend non seulement de l'exactitude des traitements effectués, mais également du temps où les résultats de ces traitements sont fournis [10]. Un STR doit être conçu tel que ses performances seraient définies dans le pire cas, savoir à l'avance si un système va respecter ses contraintes temporelles. Mais ils ne sont pas satisfaisants pour la gestion efficace des données lorsque celles-ci sont volumineuses et seul un système de type SGBD pourrait répondre à ce besoin.

II.2. Un SGBD : définition et limites

Un SGBD est un système qui permet d'interagir avec la base de données. Il permet à un utilisateur de définir les données, de consulter la base et de la mettre à jour. Un des aspects essentiels de ces systèmes est qu'ils s'avèrent utiles pour gérer de manière efficace d'importants volumes de données. L'objectif de performance des SGBD est de maximiser le débit général des transactions, sans se soucier du temps d'exécution d'une requête individuelle. Leur objectif de qualité est d'assurer l'intégrité des données.

Dans les SGBD classiques, il n'est pas tenu compte des contraintes temporelles des transactions. Il peut donc arriver que le temps de réponse de certaines transactions soit très important, ce qui n'est pas acceptable dans une application temps réel où chaque transaction doit s'exécuter durant un intervalle de temps précis.

II.3. Les applications temps réel et leurs spécifications

Le fonctionnement général des applications temps réel peut se résumer ainsi :

- acquisition des données depuis l'environnement à l'aide de capteurs,
- traitement des données et élaboration des résultats au bout d'un délai limité,
- envoi d'ordres de commande de l'environnement à l'aide d'actionneurs.

À ces fonctions de base, et selon le domaine d'application considéré, viennent s'ajouter d'autres fonctions telles que la maintenance de l'installation industrielle et des équipements informatiques, la gestion des personnels, le service après-vente, etc. Cependant, toutes les applications temps réel ont en commun la prépondérance du facteur temps. En effet, les applications temps réel doivent réagir en tenant compte de l'écoulement du temps, c'est la caractéristique fondamentale qui distingue globalement les applications temps réel des autres types d'applications informatiques.

II.4. Pourquoi un SGBD temps réel ?

Les STR possèdent des mécanismes pour gérer les contraintes temporelles des opérations (tâches) mais le volume des données qu'ils manipulent reste limité. Les SGBD temps réel sont des systèmes qui permettent de gérer la double contrainte, cohérence logique et cohérence temporelle des données [3]. Les contraintes temps réel dans un SGBDTR peuvent prendre des formes diverses (périodes, échéances absolues ou relatives, etc.) et s'appliquer à divers composants d'une application, c'est-à-dire à des actions (par exemple, une transaction doit se terminer avant une échéance fixée), à des données (par exemple, la validité temporelle des données est limitée) ou à des événements (par exemple, tel événement doit apparaître x unités de temps après tel autre événement) [5] [7].

II.4..1 Taxonomie des données dans un SGBD temps réel

Un SGBDTR permet de gérer différents types de données :

- les données temps réel qui ont un intervalle de validité en dehors duquel leur valeur est obsolète,
- les données dérivées dont la valeur est obtenue à partir des données temps réel,
- les données non temps réel qui correspondent aux données classiques que l'on trouve dans les bases de données classiques.

Les opérations qui s'exécutent dans un tel système doivent tenir compte des différents types de données.

II.4..2 Taxonomie des transactions temps réel

Dans les SGBD temps réel, les transactions possèdent des contraintes temporelles, souvent sous forme d'échéances qui peuvent provenir de deux sources [7] :

- des contraintes temporelles liées à l'environnement (par exemple, l'action qui consiste à corriger la trajectoire d'un robot téléguidé doit s'exécuter avant que le robot ne rentre en collision avec un obstacle),
- des contraintes temporelles liées à des choix de conception et d'implémentation d'une application (par exemple, durée d'exécution d'une opération par le processeur, délai de communication, etc).

Selon les conséquences engendrées sur l'application ou sur l'environnement lorsqu'une transaction manque son échéance, les transactions qui s'exécutent dans un SGBD temps réel sont classées en trois catégories [3] [6] :

1. Transactions à échéances strictes critiques (*hard deadline transactions*) : une transaction qui manque son échéance, peut avoir des conséquences graves sur le système ou l'environnement contrôlé. Par exemple, dans un système d'interception de missiles, les transactions qui permettent de contrer des missiles adverses sont à échéances strictes critiques [5].
2. Transaction à échéances strictes non critiques (*firm deadline transactions*) : si une transaction manque son échéance, elle devient inutile pour le système. Elle est donc abandonnée dès qu'elle rate son échéance. L'exemple ci-dessus du robot en est une illustration : le robot doit changer de direction avant la collision avec un objet détecté.
3. Transactions à échéances non strictes (*soft deadline transactions*) : si une transaction manque son échéance, le système peut ne pas l'abandonner immédiatement. En effet, elle peut avoir une certaine utilité pendant un certain temps encore, mais la qualité de service qu'elle offre est moindre. Les applications multimédia sont également un très bon exemple de système manipulant des transactions de type *soft*.

III. UML POUR LES SYSTÈMES TEMPS RÉEL

UML représente l'état de l'art des langages de modélisation objet [2]. Il fournit les fondements pour spécifier,

construire, visualiser et décrire les artefacts d'un système logiciel. Pour cela, UML se base sur une sémantique précise et sur une notation graphique expressive. Il définit des concepts de base et offre également des mécanismes d'extension de ces concepts.

III.1. Modéliser le comportement

Du point de vue comportemental, UML propose principalement trois constructions : les machines d'états, les diagrammes d'activité et un langage d'action (Action).

III.1..1 Les machines d'états

Les machines d'états offrent de nombreux concepts, tels que la notion d'état hiérarchique, composite, historique et organisé par nœuds de branches qui, combinés, couvrent la plupart des formalismes sur la notion d'état. La description de la sémantique des machines d'états est de type opérationnel et repose sur une machine d'exécution hypothétique qui présente les trois caractéristiques suivantes (cf. Figure 1) :

- une file d'attente d'événement qui sert à stocker les instances d'événements entrantes en attendant de les consommer,
- une politique de choix des événements qui détermine l'ordre d'extraction des occurrences d'événement contenues dans la file d'attente,
- un processeur à événement qui exécute les traitements associés aux événements en respectant la sémantique des machines d'états-transitions de UML et, en particulier, l'hypothèse d'exécution (***Run-to-Completion***).

Les événements sont dépilés un par un et consommés par une machine d'états-transitions. L'ordre dans lequel ils sont dépilés n'est pas précisé dans UML, cela constitue un point de variation sémantique. La sémantique d'exécution des événements est basée sur l'hypothèse dite de (Run-to-Completion). Cela signifie qu'un événement ne peut être dépilé puis consommé que lorsque le traitement de l'événement précédent est achevé.

III.1..2 Les diagrammes d'activité

Les diagrammes d'activité servent à faciliter la modélisation de traitements complexes en termes de flots de contrôle et de flots d'objets entre les différents constituants de l'activité. La sémantique associée aux diagrammes d'activité repose sur une circulation de jetons, proche de celle rencontré dans les réseaux de Pétri. Un jeton modélise une donnée ou un objet. Sa circulation dans le réseau est conditionnée par les éléments de contrôle (arcs ou nœuds). Ces éléments permettent d'exprimer des notions de parallélisme et de synchronisation.

On distingue trois types de nœuds. Les nœuds d'action transforment les flux de données/contrôle d'entrée en flux de données/contrôle de sortie. Ces derniers sont alors les entrées d'autres actions. Les nœuds de contrôle définissent les règles de circulation des jetons à travers le graphe. Les nœuds objets servent à stocker temporairement des données ou des objets. Pour connecter ces nœuds, il existe deux types d'arcs : les arcs de flux de contrôle et les arcs de flux d'objet. Les premiers synchronisent le début d'une action (destination de l'arc) avec la fin d'une autre action (origine de l'arc). Les arcs de flux d'objet permettent de faire passer des valeurs entre deux nœuds.

III.1..3 Le langage d'action

UML définit le concept d'*Action* comme étant l'unité fondamentale de spécification comportementale permettant à des modèles UML d'être complètement exécutable. Le principe repose sur le fait que les actions échangent des flux de contrôle et de données via des fiches d'entrée et de sortie.

La spécification de la sémantique d'action de UML est constituée des parties suivantes :

- les actions de base (*Foundation action*) qui définissent les bases des notions d'action (*Action*) et de procédure (*Procedure*). Cette dernière consiste en actions qui sont composées de flots de données (objets) via des entrées (*InputPin*) et des sorties (*OutputPins*).
- les actions composites (*Composite actions*) qui définissent des structures récursives à partir desquelles il est possible de définir des actions plus complexes à partir des actions primitives.
- les actions de lecture et d'écriture (*Read and write actions*) qui définissent comment créer, détruire des objets, comment consulter, modifier leurs attributs, les variables, les liens, etc.
- les actions de calcul (*Computation actions*) qui définissent comment transformer des données. Elles sont conçues pour modéliser essentiellement des fonctions mathématiques.
- les actions de manipulation de collections (*Collection actions*) qui définissent des actions de haut niveau permettant d'appliquer une sous action sur un ensemble d'éléments.
- les actions de communication (*Messaging actions*) qui définissent comment les objets peuvent communiquer en échangeant des messages (synchrones, asynchrones, diffusion, etc.).
- les actions de saut (*Jump actions*) qui définissent comment quitter un flot de contrôle principal afin de poursuivre dans le contexte d'un autre flot de contrôle.

Cette sémantique d'actions a été élaboré afin de pouvoir supporter une très large variété de langages de programmation. Cela inclut, bien évidemment, les langages orientés objets courants comme C++ ou Java.

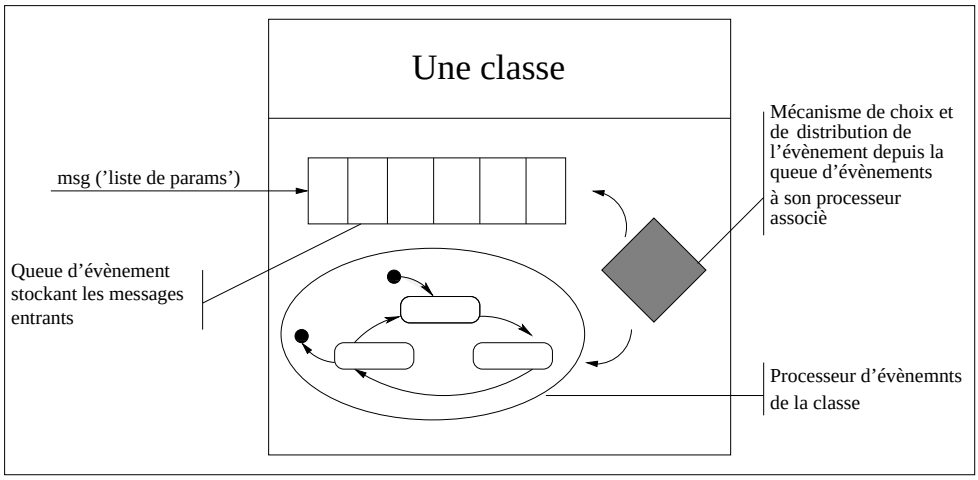

FIG. 1. *Sémantique opérationnelle du principe de la machine d'états UML.*

III.2. Modéliser les interactions

Une interaction est constituée d'un ensemble de messages que différents objets s'échangent afin de réaliser une activité ou une tâche donnée. Ce concept permet de décrire le comportement global du système [11].

III.2..1 Le diagramme de séquence

Le diagramme de séquence permet de décrire une interaction qui est elle-même un ensemble de messages entre des instances en vue de réaliser l'opération ou le résultat désiré. Le diagramme de séquence associe à chacun des objets impliqués dans une interaction une ligne de vie verticale représentant le temps (le temps s'écoule de haut en bas) et permettant d'identifier explicitement la séquence et l'ordre des messages émis et reçus par les objets. L'ordre est partiel par rapport à tout le système. Lorsqu'on désire indiquer qu'un message déclenche un traitement particulier dans l'objet, on représente ce traitement avec un petit rectangle vertical le long de sa ligne de vie.

III.2..2 Le diagramme de collaboration

Le diagramme de collaboration est une autre et équivalente manière de représenter les interactions entre les objets. Contrairement au diagramme de séquence qui permet de mettre l'accent sur l'aspect temporel (ordre des messages) d'une interaction, le diagramme de collaboration met l'accent sur l'aspect structurel d'une interaction.

III.3. Le temps dans UML

UML définit deux types de données particuliers relatifs au temps,

- le type ***Time*** : définit une valeur représentant un moment absolu ou relatif du temps,
- le type ***TimeExpression*** : permet de spécifier des expressions dont l'évaluation donne une valeur de type Time.

Elles peuvent être utilisées, en particulier, dans deux types de diagramme : les diagrammes d'états et les diagrammes de séquence.

III.3..1 Modélisation de temps dans les machines à états

UML définit un événement spécifique appelé ***TimeEvent***. Il sert à modéliser l'expiration d'une échéance qui peut être relative ou absolue :

- un événement dénotant le passage d'une quantité de temps suite à l'entrée dans l'état contenant la transition est noté avec le mot-clé ***after*** suivi d'une expression de type ***TimeExpression*** qui donne la valeur temporelle de l'événement.
- un événement dénotant l'occurrence d'une date absolue est noté via le mot-clé ***when*** suivi d'une date absolue de type ***Time***.

La Figure 2 décrit trois extraits de machine à états-transitions illustrant l'utilisation possible des événements temporels de UML. Dans les trois cas, lorsque le temporisateur armé arrive à échéance, il génère un événement qui est stocké comme tout autre événement dans la file d'attente associée à la machine d'états. Si celle-ci est dans l'état **S**

Modélisation des évènements temprels	Description
S — after(10 ms) /'procédure' → …	L'évènement est généré 10 ms après la date d'entrée dans l'état S.
S — after(10 ms after the exit of S2) /'procédure' → …	L'évènement est généré 10 ms après la date de sortie de l'état S.
S — When (1er Janvier 2000, 0h00) /'procédure' → …	L'évènement est généré le 1er Janvier 2000 à 0h00.

FIG. 2. *Exemple de spécification d'événements temporels.*

au moment où l'événement temporel est sélectionné, l'événement est consommé et la transition est tirée. Dans le cas contraire, l'événement temporel est perdu. Contrairement aux autres événements, les événements temporels ne peuvent pas être conservés (***defered***).

La date d'occurrence d'un événement temporel est la même que sa date de réception. Cependant, il faut noter qu'il peut exister un délai variable et difficilement mesurable entre la date de réception d'un événement temporel et le moment auquel il est pris en compte par l'objet récepteur. En effet, ce délai est la conséquence de la politique de stockage et de traitement mise en œuvre par chaque méthode pour gérer la file des événements reçus [4].

III.3..2 Modélisation du temps dans les diagrammes de séquence

Les diagrammes de séquence possèdent deux dimensions :

- l'axe horizontal qui contient les objets impliqués dans l'interaction décrite par le diagramme,
- et l'axe vertical qui représente le temps.

Le temps progresse dans le sens descendant de l'axe vertical mais sans qu'il soit attaché de métrique. En règle générale, dans les diagrammes de séquence de UML, on suppose que le délai de transmission des messages échangés entre les objets de l'application est négligeable devant le reste de l'activité de l'application, ce qui explique le fait que les flèches symbolisant l'échange de messages sont disposées horizontalement de l'émetteur vers le récepteur. Cependant, si pour une raison particulière, l'utilisateur ne peut pas se satisfaire de cette hypothèse, il peut incliner vers le bas la flèche du message pour modéliser le délai de transmission d'un message (cf. Figure 3).

UML introduit un certain nombre de fonctions temporelles dédiées à la manipulation du temps dans les échanges de messages réalisés dans les diagrammes de séquence. Les fonctions ***sendTime*** (date d'envoi d'un message) et ***receiveTime*** (date de réception d'un message) peuvent être appliquées à l'identification des messages pour renseigner sur

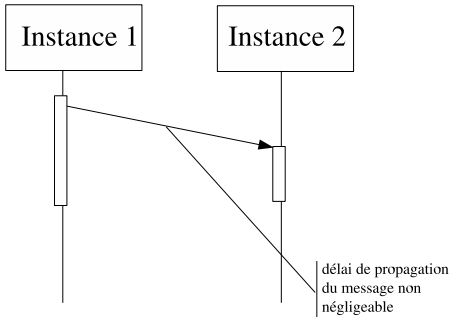

FIG. 3. *Spécification du délai de propagation.*

des caractéristiques temporelles particulières des messages.

Pour spécifier une contrainte de temps dans les diagrammes de séquence, le guide de notation de UML propose deux solutions : via une cotation temporelle ou via l'expression de contrainte (cf. Figure 4).

IV. COMMENT CONCEVOIR DES BASES DE DONNÉES TEMPS RÉEL ?

Les méthodes de conception d'un SGBD classique sont généralement basées sur la cohérence logique alors que les méthodes de conception des STR sont basées sur la cohérence temporelle. Puisque les SGBD temps réel combinent les deux notions, il est nécessaire de définir des méthodes de conception de ces systèmes en se basant sur les méthodes de conception des SGBD classiques et des STR.

L'objectif de ce travail est de pouvoir proposer des méthodes de conception pour les applications reposant/utilisant des bases de données temps réel. Ces méthodes doivent à la fois permettre la spécification de la co-

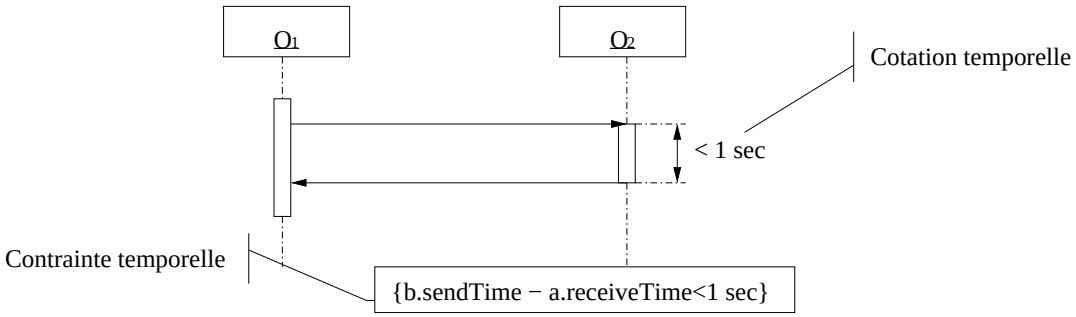

FIG. 4. *Spécification temporelle dans un diagramme de séquences.*

hérence logique (contraintes d'intégrité) et de la cohérence temporelle (contraintes temporelles) des données. En outre, elles doivent aussi permettre de spécifier les contraintes temporelles des transactions.

La prise en compte des contraintes temporelles des transactions pourra s'effectuer en faisant un parallèle entre tâches temps réel et transactions temps réel et par conséquent en se basant sur les travaux effectués pour la conception des STR, notamment avec UML temps réel. Il s'agit de spécifier les différents types de contraintes temporelles au niveau des diagrammes de séquence comme nous l'avons déjà évoqué précédemment.

Il faut aussi considérer la cohérence temporelle des données et ceci pourra être fait au travers des diagrammes d'états qui permettent de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composants ou avec des acteurs.

V. CONCLUSION ET PERSPECTIVES

Dans cet article, nous avons abordé la problématique de la conception des applications reposant sur les bases de données temps réel. Nous partons du constat que ces applications sont à l'intersection des systèmes temps réel et des applications utilisant des bases de données classiques. Notre approche consiste donc à nous appuyer sur les travaux concernant les méthodes de conception pour les systèmes temps réel et pour les bases de données classiques. Un état de l'art de ces différentes méthodes de conception a été réalisé afin d'en dégager les limites et les propriétés réutilisables pour notre problématique.

Une approche intéressante semble être celle qui consiste à s'appuyer sur UML temps réel. Cet article donne quelques pistes de recherche basées sur cette approche. Dans nos futurs travaux, nous étendrons ces pistes afin de proposer une méthode de conception applicable et appliquée à des cas d'études ou des applications concrêtes.

RÉFÉRENCES

[1] R. Abbott and H. Garcia-Molina. Scheduling Real-Time Transactions : A Performance Evaluation. In *Proceedings of the* 14^{th} *Very Large Data Bases Conference*, pages 1–12, 1988.

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

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

[4] F.Bause and P.Buchhloz. Protocol analysis using a timed version of sdl. In *Formal Description Techniques*, North Holland, 1991.

[5] Z. Mammeri. Expression et dérivation des contraintes temporelles dans les applicationtemps réel. *APII-JESA*, 32(5-6) :609–644, 1998.

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

[7] K. Ramamritham. Where do time constraints come from and where do they go ? *Journal of Database Management*, 7(2) :4–10, 1996.

[8] S. Son. *ACM SIGMOD Record : Special Issue on Real-Time Databases*. 1988.

[9] S. Son and J. Liu. How well can data temporal consistency be maintained ? In *IEEE Symposium on Computer-Aided Control System Design*, 1992.

[10] J. Stankovic and K. Ramamritham. *Hard real-time systems : a tutorial*. IEEE Computer Society Press, 1988.

[11] F. Terrier and S. Gerard. Les concepts natifs d'uml pour le temps réel. In *Ecole d'été Temps Réel*, Toulouse, France, 2003.