Conception de logiciel en temps réel – Progrès actuels et défis à venir

par *Manas Saksena, Université Concordia, et*
Bran Selic, ObjecTime Limited.

In this article, a high-level overview of the state of the art of real-time software design is presented. In particular, attention is focused on standard techniques for dealing with the critical issues of concurrency and timeliness as well as the tools that support them. The various design styles that have evolved over time for constructing real-time software are also described. In conclusion, a brief review of the principal technological trends currently emerging in the field are described.

Dans cet article, on présente un survol des plus récents progrès technologiques accomplis dans le domaine de la conception de logiciels en temps réel. Une attention particulière est portée aux techniques normalisées servant à gérer les questions clés de la concurrence d'accès et de la synchronisation ainsi qu'aux outils utilisés pour leur mise en application. On décrit l'évolution des divers styles de conception des logiciels en temps réel. En conclusion, on présente un bref examen des principales nouvelles tendances technologiques dans ce domaine.

1.0 Introduction

Ces dernières années, le traitement en temps réel s'est imposée comme une discipline importante des sciences et du génie informatiques. Comme la puissance de calcul ne cesse d'augmenter, un nombre accru de systèmes sont réalisés par des logiciels qui tirent parti de la souplesse et du perfectionnement propres à cette solution. Mais au fur et à mesure que les logiciels en temps réel deviennent plus complexes, le style de conception de logiciels joue un rôle toujours plus grand dans l'élaboration des logiciels et des systèmes.

Le milieu de l'informatique en temps réel est habituellement très réticent à adopter de nouvelles technologies de conception de logiciels. La situation s'explique en partie par les exigences rigoureuses en matière de performance et de rapidité qu'imposent de tels systèmes, ce qui incite souvent les créateurs de logiciels à s'en tenir à des techniques de très bas niveau; de plus, les coûts élevés et les contraintes de sécurité entrent en jeu. C'est ainsi que les approches ayant fait leurs preuves remportent la faveur sur d'autres technologies susceptibles d'être supérieures sans toutefois avoir été éprouvées.

Les progrès technologiques réalisés dans le domaine de l'informatique moderne permettent toutefois de remettre en question ces hypothèses usuelles. Les ordinateurs sont de plus en plus rapides et les outils de création de logiciels, de plus en plus puissants. Qui plus est, comme la demande en fonctionnalités plus évoluées augmente et que les logiciels deviennent sans cesse plus complexes, les techniques de niveau bas communément appliquées ne sont plus à la hauteur. D'où le besoin criant, pour une foule d'applications en temps réel, de reposer sur des méthodes de pointe plus poussées en vue de satisfaire aux contraintes de sécurité.

1.1 Caractéristiques des systèmes en temps réel

En gros, le système en temps réel est un système qui réagit à point nommé. C'est pourquoi le « temps » en est une ressource de tout premier plan; les activités doivent être prévues et exécutées dans les contraintes de temps. Cette contrainte de temps (et de systèmes) est souvent classée en temps réel dur – où ne pas respecter l'échéance constitue une défaillance catastrophique du système – et en temps réel mou – où le dépassement occasionnel d'une échéance s'avère tolérable.

De nombreux systèmes en temps réel se caractérisent également par leur nature intégrée, c'est-à-dire qu'ils sont les composantes d'un système plus imposant doté d'interactions dans le monde physique. C'est souvent là que réside la principale source de complexité des systèmes en temps réel. Le monde physique se comporte généralement de manière non déterministe : les événements se produisent sans synchronisme, concurremment, dans un ordre imprévisible. Quelle que soit l'occurrence, le système en temps réel intégré doit réagir comme il convient et en temps voulu.

La concurrence dans le monde physique implique habituellement la concurrence d'accès du logiciel du système en temps réel. En fait, l'une des premières exigences de tels systèmes consiste à synchroniser les multiples activités concurrentes qui se déroulent dans l'environnement. Hélas, la concurrence complique encore davantage la complexité de la conception de logiciels puisqu'elle s'oppose au raisonnement intrinsèquement séquentiel de l'humain, qui va de la cause à l'effet.

Nombre de systèmes en temps réel partagent en outre un autre trait dominant : leur fiabilité. Ces systèmes jouent en effet un rôle crucial dans leur environnement et, s'ils ne réagissent pas correctement, les coûts risquent d'être élevés, voire de se mesurer en vies humaines. Par conséquent, pareils systèmes doivent souvent être très fiables (c'est-à-dire s'exécuter correctement) et disponibles (c'est-à-dire fonctionner sans interruption).

1.2 Aperçu de l'article

En raison des contraintes d'espace, nous nous concentrerons sur la concurrence d'accès et la contrainte de temps des systèmes en temps réel. Nous commencerons par un aperçu des techniques permettant de gérer ces deux aspects, nous présenterons ensuite deux styles de conception de systèmes en montrant comment chacun d'eux prend en charge ces contraintes, puis nous discuterons des atouts et des lacunes de chaque style. Après quoi nous offrirons un survol des technologies et des outils dont disposent les créateurs de logiciels en temps réel. Pour conclure, nous cernerons certains des défis que nous réserve l'avenir.

2.0 Gestion de la concurrence d'accès

Dans le domaine de la conception de systèmes en temps réel, l'un des grands problèmes consiste à gérer la concurrence d'accès. La concurrence d'accès sert souvent même dans les systèmes monoprocesseur, tant pour simplifier la structure du logiciel que pour améliorer la capacité de réaction du système aux événements critiques. Cette forme de concurrence d'accès est désignée sous le nom de fonctionnement multitâche et implique le multiplexage d'une foule de tâches concurrentes sur un processeur physique unique.

Toutefois, si la concurrence constitue un outil nécessaire à la gestion du temps de réaction du système, elle n'en reste pas moins bourrée de difficultés. Notre incapacité intrinsèque à faire des raisonnements convenables sur la concurrence risque de mener à l'insertion de bogues subtils dont le repérage et la correction posent problème, si bien qu'on atteint difficilement le degré de fiabilité voulu. La concurrence d'accès complique également l'évaluation de la performance.

C'est pourquoi on a mis au point des techniques spéciales applicables à la gestion de la concurrence d'accès de manière ordonnée. On compte trois grands types de mécanismes de base :

- représentation et suivi des activités concurrentes dans l'ordinateur;
- communication entre les activités concurrentes;
- synchronisation de l'exécution des activités concurrentes.

2.1 Abstractions d'activités concurrentes

Dans la plupart des cas, les mécanismes de représentation d'activités concurrentes fournissent à chaque activité un processeur « virtuel » qui exécute une thread (séquence d'instructions) et maintient son état pendant ce temps. On parle alors habituellement de processus. Normalement, chacun de ces processus a son propre espace d'adressage logiquement distinct de celui des autres processus. Malheureusement, faire passer le processeur physique d'un processus à l'autre exige souvent un temps système prohibitif. Il faut en effet changer les contextes et exécuter des transferts de registres dans le processeur physique, des opérations qui durent plusieurs dizaines, voire des centaines, de microsecondes, même avec les processeurs rapides aujourd'hui sur le marché.

Pour réduire ce temps système, nombre de systèmes d'exploitation (SE) donnent la possibilité d'inclure de multiples threads plus courtes dans un seul processus. Les unités intégrées à ce processus partagent le même espace d'adressage, ce qui diminue le temps système dû aux changements de contexte, mais multiplie du même coup les risques de conflits d'adressage de la mémoire.

On peut fréquemment réduire encore davantage le temps système dû aux changements de contexte en constituant des abstractions d'accès concurrent moins gourmandes à la couche application (soit celle au-dessus du système d'exploitation). Cette technique est fort courante dans les systèmes très imposants à nombreuses threads concurrentes. Elle repose sur le changement de contexte commandé par l'application à des points où il y a peu ou pas de contextes à sauvegarder. Ce type de changements de contexte n'exige pas de croisement de la frontière « robuste » entre la couche noyau et la couche utilisateur du processeur et, par conséquent, les threads de la couche application peuvent être multiplexées en une seule thread du système d'exploitation. Ces threads de la couche application sont ordonnancées par un gestionnaire propre à l'application qui fonctionne au sein de la thread du système d'exploitation.

L'abstraction d'un objet actif fournit un tel exemple de gestion de la concurrence d'accès à une application donnée. L'objet actif reçoit des événements de l'extérieur (généralement mis en file d'attente) et y réagit en complétant sa tâche. Autrement dit, les nouveaux événements ne peuvent évincer celui qui est en cours de traitement. Une fois le traitement terminé, les piles de données et d'appels de l'objet actif sont vidées sans avoir à être enregistrées au changement de contexte pour l'événement suivant. Les objets actifs multiples peuvent se combiner en une seule thread du système d'exploitation en partageant les mêmes piles. De cette façon, l'ordonnancement des objets actifs ne se produit qu'une fois terminé le traitement de l'événement précédent.

2.2 Mécanismes de communication

La communication des données se fait généralement par données partagées ou par passage de messages. Dans le cas des données partagées, deux threads ou plus peuvent accéder aux objets à données partagées et, de la sorte, communiquer indirectement l'une avec l'autre. Par contre, le passage de messages implique l'échange explicite de données entre les deux threads.

Lorsque plusieurs threads utilisent des données partagées, l'accès concurrent risque d'entraîner des mises à jour incohérentes. C'est pourquoi il faut souvent accéder à ces données aux sections critiques (soit des actions atomiques logiques) par un mécanisme de synchronisation nommé exclusion mutuelle. Les formes qu'emprunte ce mécanisme sont décrites à la section suivante. Les threads qui partagent un espace d'adressage peuvent automatiquement communiquer entre elles par la mémoire partagée. Même dans le cas des processus, une foule de systèmes d'exploitation permettent la création de zones partagées en mémoire entre ces processus.

La communication par passage de messages est tantôt asynchrone, tantôt synchrone. Si elle est asynchrone, l'activité émettrice ne se synchronise pas avec la réceptrice; elle transmet plutôt l'information à une file d'attente d'où la réceptrice l'extraira ultérieurement.

L'émettrice et la réceptrice travaillent indépendamment l'une de l'autre et ne peuvent en aucun cas attribuer une valeur implicite à l'état de l'autre. Si la communication est synchrone, l'émettrice et la réceptrice se synchronisent en plus d'échanger l'information. Pour la communication synchrone pure et simple, les activités interagissent par rendez-vous pour s'échanger l'information. La plupart des systèmes d'exploitation prennent en charge le passage de messages entre processus ou threads d'une manière quelconque. Ce mode de communication entre threads d'un même processus peut également se réaliser par mémoire partagée et synchronisation, décrites à la section suivante; cette option est souvent plus efficace puisque le système d'exploitation n'intervient pas.

2.3 Mécanismes de synchronisation

La synchronisation implique de coordonner l'exécution de deux activités ou plus de manière à éviter les conflits d'accès. L'exclusion mutuelle est un moyen d'y parvenir; elle est relativement simple à réaliser à l'aide de sémaphores ou de verrous « mutex » (mutual exclusion). Le problème avec cette méthode tient dans le fait qu'il s'agit d'abstractions de niveau bas indépendantes de la sémantique de l'application. C'est donc dire que le lien sémantique entre l'application et le mécanisme de synchronisation (par exemple, les circonstances propres à l'application dans le cadre desquelles il faut procéder à la synchronisation), doit être constitué par le programmeur de cette application. De plus, pareils mécanismes sont très sujets à l'erreur puisque le code de synchronisation se trouve généralement un peu partout dans le programme d'application, ce qui le rend souvent difficile à comprendre et à en faire la maintenance.

Un méthode plus sûre pour gérer les conflits d'accès consiste à encapsuler les données partagées comme des données privées d'objets protégés et à les rendre accessibles par les opérations appelées sur l'objet. En principe, un maximum d'une thread concurrente à la fois peut avoir accès à cet objet protégé. De la sorte, le code de synchronisation est encapsulé dans les objets à données partagées et diverses threads peuvent sans problème appeler des opérations sur des objets de données sans susciter de conflits d'accès.

Pour d'autres formes plus générales d'exigences en matière de synchronisation, une activité concurrente doit attendre la réalisation d'un événement avant d'entrer en jeu. En pareil cas, l'activité est bloquée par des abstractions de synchronisation telles des variables conditionnelles. L'activité peut être mise en éveil par l'indication de la variable conditionnelle. Normalement, ces abstractions sont utilisées avec la mémoire partagée et l'exclusion mutuelle pour résoudre les problèmes de synchronisation.

3.0 Analyse de la performance en temps réel

L'analyse de la performance en temps réel est un élément essentiel de la plupart des systèmes en temps réels et permet essentiellement d'évaluer si le système répond à ses contraintes de temps.

3.1 Évaluation de la durée d'exécution

Pour évaluer la performance d'un système concurrent, il faut d'abord se pencher sur une seule tâche séquentielle. L'évaluation de la durée d'exécution borne cette durée pour l'exécution du fragment de code d'une seule thread sur un processeur spécialisé (c'est-à-dire en dehors d'un fonctionnement multitâche). Cette opération pose deux problèmes : 1) la circonscription des chemins autorisés dans le code; 2) la détermination de la durée d'exécution dans chacun de ces chemins. La délimitation exacte de la durée d'exécution au moyen de techniques analytiques soulève des difficultés. D'abord, en général, on ne peut décider de déterminer tous les chemins autorisés dans un programme.

Ensuite, la durée d'exécution dans les chemins autorisés peut être fonction des données. Enfin, les caractéristiques architecturales des processeurs modernes, tels la mémoire cache et le traitement en pipeline, compliquent l'évaluation exacte de la durée d'exécution en raison d'effets secondaires globaux. En pratique, compte tenu des limites des techniques analytiques, il faut donc se rabattre sur les mesures et les essais étoffés pour évaluer cette donnée.

3.2 Ordonnancement en temps réel

Si l'analyse de la durée d'exécution d'une tâche individuelle est essentielle, elle ne saurait être suffisante. En cas de traitement multitâche, il faut également tenir compte de la gestion des tâches dans le processeur. Quand le travail s'effectue en temps réel dans un processeur unique, la difficulté de l'ordonnancement consiste à décider de l'ordre d'exécution d'un jeu de tâches (threads concurrentes) selon certaines caractéristiques connues (par exemple, la périodicité et la durée d'exécution).

La méthode la plus simple consiste alors à planifier l'ordonnancement hors ligne. Au moment de l'exécution, les tâches sont réparties en fonction de cet ordonnancement. L'ordonnancement statique en temps réel convient tout particulièrement quand les activités sont majoritairement appelées périodiquement. On peut créer un ordonnancement cyclique pour un ensemble de tâches périodiques en temps réel. L'exécution cyclique classique constitue un cas spécial d'ordonnancement statique.

L'ordonnancement statique cyclique ne convient pas lorsque les événements sont susceptibles de se produire de manière apériodique. Si l'on se contente de procéder par appels, l'utilisation du processeur est fort inefficace. On privilégie plutôt l'ordonnancement à l'exécution. Le mécanisme le plus commun repose alors sur la préemption des priorités, c'est-à-dire que les tâches sont réparties à l'exécution en fonction de leur priorité : la tâche de la plus haute priorité passe toujours à l'exécution la première.

L'ordonnancement à l'exécution doit toujours être assorti d'une analyse hors ligne (la simulation) afin d'assurer le respect de la contrainte de temps à l'exécution. L'analyse de la gestion des tâches permet de déterminer si ce sera le cas avec un système donné. Une solide théorie fondée sur l'ordonnancement par préemption des priorités et l'analyse déterministe permet de prédire le temps de réponse aux événements dans la pire éventualité, compte tenu de la cadence maximale de leur arrivée, de la durée d'exécution maximale et de l'ordre de priorité des tâches.

La synchronisation des tâches complique l'analyse de leur ordonnancement. En effet, si l'on applique l'exclusion mutuelle des tâches, on risque l'inversion des priorités, c'est-à-dire, une tâche à priorité faible peut bloquer l'exécution d'une autre à priorité plus élevée lorsque toutes deux partagent une section critique. Pire, cette inversion des priorités peut perdre ses bornes puisqu'un nombre arbitraire de tâches à priorité intermédiaire risque de retarder le traitement de la tâche à priorité faible. Heureusement, un certain nombre de protocoles d'héritage des priorités permet de borner de telles inversions et d'appliquer l'analyse de la gestion des tâches. Ces protocoles visent essentiellement à admettre d'abord la tâche à faible priorité se trouvant dans une section critique pour empêcher celles à priorité intermédiaire de l'écarter.

Les formes de synchronisation plus générales rendent l'analyse de la gestion des tâches encore plus difficile, le plus souvent impossible. La simple synchronisation de la préséance se prête toutefois à l'analyse déterministe et sert en communication synchrone.

L'analyse de la gestion des tâches constitue une technique au point, mais se fonde sur diverses hypothèses, dont celles relatives aux modèles de durée d'exécution dans la pire éventualité connue, de concurrence simple, de synchronisation, de communication, etc. S'il est vrai qu'on peut tout de même appliquer une variété de techniques d'analyse de la performance lorsque certaines de ces hypothèses ne tiennent pas (par exemple, contrôle du modèle, analyse stochastique de la performance, simulation), il n'en demeure pas moins que l'exercice s'avère ardu.

4.0 Paradigmes de conception

Dans cette section, nous aborderons deux paradigmes de conception utilisables dans la création de systèmes en temps réel. Par paradigme de conception, nous entendons ici la représentation d'un style d'élaboration d'applications en temps réel et, tout particulièrement, la représentation et la modélisation d'activités concurrentes.

Dans l'idéal, le paradigme de conception doit s'appuyer sur un jeu de notations standard (de préférence visuelles) et sur une sémantique formelle s'y rapportant. La sémantique formelle permet de simuler les caractéristiques de conception, d'en analyser la contrainte de temps et d'automatiser la génération du code. Mais rien de tout cela n'est faisable si nous souhaitons disposer de toute la généralité disponible en langage de programmation standard. Aussi, compte tenu des exigences propres aux systèmes en temps réel, la généralité est-elle souvent sacrifiée au profit de l'analyse automatisée, de la sémantique formelle et de la génération du code.

4.1 Conception axée sur le temps

Ce style de conception permet maintenant de modéliser les principaux scénarios fonctionnels d'un système en temps réel et de simplifier l'analyse de la contrainte de temps. Par conséquent, il se fonde sur un modèle de concurrence simple. Dans ce modèle, la tâche est activée par un événement déclencheur; à l'arrivée de ce déclencheur, la tâche effectue certains calculs puis attend l'événement déclencheur suivant. Ce déclencheur peut être relié à une source d'interruption externe, à une horloge, au signal provenant d'une autre tâche, etc. Comme l'horloge est souvent la principale source d'événements (de manière à rendre la charge de travail prévisible), on qualifie ce style d'« axé sur le temps ». Les communications entre les tâches se fondent sur le modèle à mémoire partagée par des objets protégés à données partagées.

Le modèle d'allocation de tâches est souvent élargi pour permettre non seulement le partage des données, mais également le passage de messages entre les tâches. Pour ce faire, on peut notamment s'assurer que chaque tâche lise un signal d'entrée à son déclenchement et produise un signal de sortie à la fin de son exécution. En reliant le signal de sortie d'une tâche au signal d'entrée d'une autre, on peut ainsi créer des graphes des tâches d'après les relations producteur-consommateur. Ces relations entre les tâches imposent d'ordonner l'exécution de chacune. On y parvient de diverses manières. Ainsi, dans un tel modèle, les signaux d'entrée externes circulent dans un graphe des tâches avant de produire les signaux de sortie externes.

Le style axé sur le temps se prête naturellement au traitement des signaux d'entrée réguliers et récurrents quand ce traitement peut circuler dans un certain nombre d'étapes (représentées par des tâches) avant de produire les signaux de sortie. Le traitement numérique des signaux et les boucles de réaction sont deux exemples qui correspondent à ce style, dont le principal atout réside en ce qu'il correspond directement aux modèles d'ordonnancement en temps réel et qu'il est très compatible avec l'analyse de la gestion des tâches (et avec l'ordonnancement hors ligne, dans certains cas).

Par contre, le modèle d'allocation de tâches est relativement simple et ne peut s'appliquer à la description du comportement de tâches complexes. Ainsi, les aspects relatifs aux commandes (initialisation du système, changements de mode, réaction aux exceptions et aux défaillances, etc.) ne sont pas formellement modélisés et, par conséquent, ne peuvent être analysés. Comme le style axé sur les événements, décrit à la section suivante, modélise explicitement ces phénomènes, il se prête plus aisément à l'analyse et à la simulation.

4.2 Conception axée sur les événements et modèles orientés objet

Contrairement au style axé sur le temps, le style de logiciels axés sur les événements a vu le jour en bonne partie pour traiter les événements asynchrones imprévisibles. C'est pourquoi ces logiciels sont structurés autour du code de traitement des événements. Un événement déclenche le code de traitement voulu et, sur achèvement de l'action, le logiciel se met en sommeil jusqu'à l'événement suivant. Puisque des événements

peuvent arriver pendant le traitement d'un autre, le logiciel permet leur mise en file d'attente pour traitement ultérieur. Le système doit réagir aux événements asynchrones en provenance de l'extérieur et cette réaction est fonction de l'état du système. Voilà qui explique pourquoi le comportement des logiciels est souvent modélisé sur celui d'une machine à nombre d'états finis dans laquelle l'arrivée d'un événement déclenche une transition d'état, et comprend le code de traitement des événements.

Un système logiciel axé sur les événements se compose souvent d'un parc de machines à nombre d'états finis et à accès concurrent qui communiquent entre elles.

Dans le cas de grands systèmes complexes, le logiciel axé sur les événements peut se combiner à l'orientation objet. L'abstraction de concurrence d'accès d'objets actifs convient particulièrement bien à la représentation de machines à états finis. Les caractéristiques de celle-ci donnent une caractéristique de comportement à l'objet actif. Ainsi, un système se compose d'un jeu d'objets actifs qui communiquent ensemble par passage de messages. On peut en outre ajouter des mécanismes de structuration supplémentaires, dont la décomposition hiérarchique et l'organisation en couches, pour créer des systèmes complexes.

Par comparaison au style de conception axé sur le temps, le principal atout de ce style-ci réside en ce qu'il possède des mécanismes de modélisation applicables au comportement de systèmes complexes découlant souvent des aspects de commande. On le constate par les caractéristiques de comportement plus générales des activités concurrentes qui se manifestent dans les objets actifs. Par contre, ce caractère général embrouille la compréhension du flux de données dans le système, ce qui complique l'analyse de la gestion des tâches.

5.0 Technologies et outils de temps réel et normes connexes

Les outils et technologies nécessaires au processus d'élaboration constituent un ingrédient majeur de l'élaboration de logiciels en temps réel. Dans la section qui suit, nous aborderons ce point ainsi que les normes dont disposent les créateurs de logiciels.

5.1 Systèmes d'exploitation en temps réel

Si, dans le passé, de nombreuses applications en temps réel ont été constituées sans recours à un système d'exploitation, l'utilisation d'un SE en temps réel en vue d'un logiciel mieux structuré s'avère indispensable lorsque l'application gère des tâches concurrentes multiples et des unités en fonctionnement concurrent.

Le SE en temps réel fournit les abstractions de tâches concurrentes ainsi que les mécanismes de communication et de synchronisation entre elles. Voilà qui simplifie grandement l'écriture de logiciels en temps réel. La plupart des SE en temps réel procèdent à l'ordonnancement par préemption des priorités tandis que la gestion des activités concurrentes visant à satisfaire la contrainte de temps est laissée à l'application.

Les SE en temps réel classiques fournissent leur propres API (application programming interface) en propriété exclusive. Néanmoins, l'arrivée de la norme POSIX a contribué à la normalisation chez plusieurs fournisseurs, qui prennent maintenant en charge les API POSIX, tout particulièrement celles qui se rapportent aux extensions en temps réel.

5.2 Outils de conception et de développement

Les outils de conception et de développement de logiciels en temps réel n'ont pas l'habitude d'être très perfectionnés. Encore une fois, cela découle partiellement de la nature intrinsèque du temps réel et des logiciels intégrés. Quoi qu'il en soit, on voit d'un bon œil que l'industrie commence enfin à investir dans les outils de développement. Ainsi, de nombreux fournisseurs offrent maintenant un environnement intégré comportant éditeur, compilateur, moniteur, débogueur, gestionnaire de configuration, outil de profil et autres. Nombre de ces outils résident sur une plateforme hôte (généralement une machine Windows NT ou UNIX), mais prennent en charge les communications avec l'environnement temps réel de la machine d'exécution.

Certains outils vont plus loin en fournissant un environnement de programmation permettant la conception en temps réel basée sur un modèle. Ces outils de génie logiciel assisté par ordinateur (GLAO) permettent maintenant d'élaborer des logiciels à l'aide de modèles évolués et apportent un soutien d'un bout à l'autre du développement, depuis l'analyse des besoins jusqu'à la mise en œuvre en passant par la conception. La plupart de ces outils sont compatibles avec les styles de logiciels axés sur les événements et orientés objet; de plus, il y a une certaine tendance à utiliser le langage de modélisation unifié mis au point depuis peu et normalisé par l'Object Management Group.

5.3 Outils d'analyse de la performance

Les outils d'analyse de la performance permettent de voir comment les logiciels en temps réel font face aux contraintes de temps. Dans l'idéal, ces outils devraient permettre d'évaluer rapidement la faisabilité de conceptions appropriées en fonction de l'estimation des coûts à l'exécution, puis d'analyser le cycle de conception jusqu'à la mise en œuvre en mesurant les coûts réels et en validant l'analyse de la performance à la lumière du comportement à l'exécution. Malheureusement, les outils de développement accusent là une relative faiblesse, bien qu'il existe aujourd'hui quelques outils s'appliquant à l'analyse de la gestion des tâches et à la simulation de modèles en ce qui concerne leurs propriétés de contrainte de temps.

6.0 L'avenir

Dans cet article, nous avons examiné la conception de logiciels en temps réel en nous concentrant sur deux aspects clés, à savoir la contrainte de temps et l'accès concurrent. Nous avons également présenté deux styles de conception et montré les caractéristiques de chacun d'entre eux sur ces aspects. Enfin, nous avons donné un aperçu de technologies et d'outils à la disposition des créateurs de logiciels en temps réel.

Il s'avère prometteur que la conception de logiciels en temps réel ne se confine plus à une simplicité indue pour enfin adopter des outils et des techniques modernes. Il reste néanmoins beaucoup à faire dans nombre de domaines. Certaines des plus grandes lacunes se rapportent à la contrainte de temps. Ainsi, les outils d'analyse de la performance sont encore relativement primitifs et ne sont pas intégrés à l'environnement de conception et de modélisation. De plus, il existe un conflit entre les modèles de conception plus généralisés et leur capacité d'analyser comment ils répondent à la contrainte de temps. Nombre de systèmes sont de type temps réel mou et on aurait grandement besoin d'outils applicables à la conception de systèmes adaptatifs dont on peut prévoir et contrôler la détérioration de la performance sous l'effet de la surcharge.

Nous prévoyons également que la conception heuristique présentera de nombreux défis. Le créateur d'un logiciel en temps réel complexe fait face à divers choix de conception. En l'absence de lignes directrices – l'heuristique – convenables, il fait des choix au cas par cas souvent susceptibles de mener facilement à une conception inefficace. La tendance aux abstractions et structures plus évoluées et plus proches du domaine à problème aggrave la situation en multipliant les choix à faire pour passer des abstractions de conception aux abstractions de mise en œuvre. Par exemple, dans le cas d'une application orientée objet, il faut faire des choix sur la façon de mettre la conception en concordance avec les threads du système d'exploitation, sur l'attribution des priorités à ces threads, etc.

Au fur et à mesure que la société développera une dépendance symbiotique à l'égard des systèmes informatiques, un nombre toujours d'entre eux croissant appartiendront à la vaste catégorie des systèmes en temps réel. C'est donc dire que les méthodes et techniques mises au point pour les logiciels en temps réel usuels, tout particulièrement les techniques relatives à la tolérance aux anomalies et à la prévisibilité, s'intégreront aux connaissances de base de la plupart des créateurs de logiciels et des ingénieurs en informatique. De plus, ces derniers recourront à des progiciels d'outils intégrés de plus en plus perfectionnés qui augmenter-

ont considérablement le degré d'automatisation du développement de logiciels en temps réel.

7. Lecture complémentaire

[1]. IEEE-CS Technical Committee on Real-Time Systems Home Page. Available at: http://cs-www.bu.edu/pub/ieee-rts/Home.html

Ce site Web renferme des liens vers des groupes de recherche, des conférences et des fournisseurs de produits commerciaux.

[2]. Burns, A. et A. Wellings, Real-Time Systems and Programming Languages, (2e éd.) Addison-Wesley, 1997.

Survol le plus complet des nombreux mécanismes et pratiques conçus spécialement pour les logiciels en temps réel.

[3]. Jalote, P., Fault Tolerance in Distributed Systems, Prentice Hall, 1994.

Excellent examen de diverses techniques liées à la fiabilité.

[4]. Klein, M., et coll., A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems, Kluwer Academic Publishers, 1993.

Traitement encyclopédique des techniques d'analyse de la gestion des tâches.

[5]. Object Management Group, « The Unified Modeling Language Specification (version 1.1) » OMG, Framingham, MA 1998.

Norme de langage de modélisation unifié disponible au site Erreur! Signet non défini.

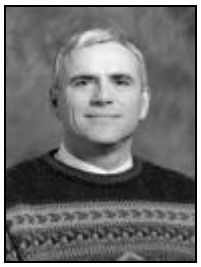
[6]. Selic, B., G. Gullekson et P. Ward, Real-Time Object-Oriented Modeling, John Wiley & Sons, 1994.

Exemple d'application du paradigme objet à un système en temps réel complexe axé sur les événements.

À propos des auteurs

Manas Saksena a reçu son doctorat de la University of Maryland, à College Park, en 1994.

Il est professeur agrégé au département d'informatique de l'Université Concordia, à Montréal. Il a fait de la recherche dans divers aspects des systèmes en temps réel, notamment les systèmes d'exploitation, l'ordonnancement, les communications et la conception. Il s'intéresse actuellement aux points liés à la conception de tels systèmes, sujet sur lequel il a publié des articles pour diverses revues et conférences de l'IEEE. Plus récemment, il s'est penché sur l'analyse de la gestion des tâches en conception de systèmes en temps réel orientés objet et sur l'élaboration d'une méthodologie de conception de systèmes en temps réel à l'aide du langage de modélisation unifié (UML).



Bran Selic est vice-président des technologies de pointe à ObjecTime Limited.