Préface

Au cours des trente dernières années, le temps réel s'est progressivement établi comme une discipline à part entière qui rassemble aujourd'hui une forte communauté issue à la fois du monde académique et de l'industrie. Ce traité en deux volumes a pour objectif de mieux faire connaître cette discipline : ses enjeux, les méthodes et formalismes qui lui sont spécifiques, les outils existants, les résultats connus et, naturellement, les recherches encore à mener.

L'informatique temps réel est une discipline qui concerne le contrôle ou le suivi de systèmes qui évoluent dynamiquement. Ces systèmes peuvent être des équipements de production ou de transport de produits, de matières, d'énergie ou d'informations. A ce sens, indubitablement, le temps réel entretient des liens étroits avec d'autres disciplines : l'automatique qui définit les cahiers des charges sous-jacents, la recherche opérationnelle pour les techniques de maîtrise et d'organisation de systèmes discrets, l'ingénierie du logiciel pour la mise en œuvre d'un développement sûr et de qualité, les processus stochastiques en raison de la connaissance le plus souvent incomplète ou imparfaite des systèmes et de leur environnement. En fait, ce qui fait l'essence de l'informatique temps réel est une exigence de garantir le niveau de performances temporelles requis par la dynamique et la sûreté des systèmes contrôlés ou surveillés : les résultats doivent non seulement être corrects en valeur mais ils doivent aussi être produits et délivrés aux bons instants. Le plus souvent cette exigence se traduira en des contraintes sur les dates de fin d'exécution des activités, on parle alors de contraintes d'échéances. Garantir

le respect de ces contraintes repose sur un ensemble de techniques de vérification formelle dont la plupart sont présentées dans ce premier tome du traité.

Les systèmes temps réel sont généralement « embarqués », c'est-à-dire situés à l'intérieur même de l'environnement avec lequel ils doivent interagir, comme un calculateur dans une voiture ou un avion. En plus des exigences de performances temporelles, ils sont alors assujettis à de fortes contraintes d'encombrement, de coût et parfois d'énergie, ce qui limite d'autant la puissance de calcul et l'espace mémoire disponibles et constitue une différence fondamentale avec l'informatique traditionnelle où globalement les ressources disponibles augmentent exponentiellement au rythme de la loi de Moore.

Les domaines d'applications intégrant de manière concrète l'informatique temps réel sont nombreux et de natures variées : contrôle de systèmes automatisés de production, surveillance de systèmes, aide au pilotage de véhicules, comme par exemple des systèmes électroniques de freinage offrant une meilleure sécurité, ou contrôle de flux dans les réseaux tels l'internet pour des applications critiques de télécommande, etc. Et l'on voit que le temps réel est loin d'être une discipline désincarnée de la réalité et qu'il peut être source de progrès dans notre quotidien.

Ce traité est organisé en deux volumes et vingt et un chapitres, rédigés pour la plupart par des conférenciers de la quatrième École d'été Temps Réel 2005 (ETR'05). Cette manifestation, soutenue par le GDR ARP (Architecture, Réseau et système, Parallélisme) du CNRS et l'INRIA, est un événement phare pour la communauté temps réel francophone qui a rassemblé, lors de sa dernière édition à Nancy, 110 chercheurs et industriels du 13 au 16 septembre 2005.

Le second volume du traité, intitulé Systèmes *Temps Réel : Ordonnancement, Réseaux, Qualité de Service*, est consacré aux mécanismes exécutifs permettant l'obtention d'une « qualité de service » temps réel. En particulier, le problème crucial du choix, de la configuration et de l'implémentation des stratégies d'ordonnancement de tâches est largement traité. Une seconde partie du tome 2 est dédiée aux réseaux et protocoles de communication temps réel qui permettent à des entités distantes d'un même système de s'échanger des informations dans des délais contraints.

Ce premier volume du traité, intitulé Systèmes *Temps Réel : Techniques de Description et de Vérification*, a pour objet de faire connaître les principaux formalismes, et les outils logiciels associés, qui offrent des solutions pour spécifier un système temps réel et vérifier, avant le déploiement du système, le respect des propriétés attendues. Les méthodes formelles constituent à l'évidence un facteur de progrès considérables vers une informatique « sûre de fonctionnement » mais leur utilisation ne pourra se faire à grande échelle que si les outils logiciels

correspondants sont développés. C'est pourquoi ce premier tome est délibérément orienté « outils » de façon à proposer des solutions concrètes à l'utilisateur potentiel de méthodes formelles. Au-delà des outils, le but est également de faire mieux connaître les formalismes sous-jacents et, c'est notre souhait, de montrer comment les méthodes formelles peuvent améliorer la qualité et la sûreté des systèmes.

Classiquement, on distingue deux grandes familles de techniques pour vérifier formellement la correction d'un système. En premier lieu, les techniques de preuve (*theorem proving*) qui sont des démonstrations mathématiques au sens classique du terme où la vérification des propriétés est effectuée par déduction à partir d'un ensemble d'axiomes et de règles d'inférences. Si l'utilisateur est généralement aidé par un assistant logiciel de preuve, l'automatisation n'en reste pas moins que très partielle et des résultats un tant soit peu complexes ne peuvent être généralement obtenus sans de fortes compétences mathématiques de la part de l'utilisateur.

La seconde famille de techniques est appelée vérification de modèles, ou *model-checking*, et consiste à construire, à partir d'une description formelle d'un système, l'espace des états atteignables puis, en parcourant l'ensemble de ces états, à vérifier le respect des propriétés attendues du système. Le *model-checking* est une technique de vérification automatique, ou « presse-bouton », ce qui explique sans doute en partie le succès grandissant qu'il rencontre dans l'industrie depuis une vingtaine d'années. Une des principales limites de cette approche réside en l'explosion combinatoire du nombre des états du système mais nous verrons au travers des différents chapitres de ce livre qu'il existe toute une panoplie de stratégies qui permettent souvent de limiter efficacement la taille de l'ensemble des états à considérer.

La vérification débute par la phase de description formelle du système, par exemple à l'aide de réseaux de Petri temporels (chapitre 1) ou d'automates temporisés (chapitre 4). Ensuite vient la phase de spécification des propriétés de bon fonctionnement que le système doit respecter. Ces propriétés relèvent de deux grandes catégories : les « propriétés de sûreté », c'est-à-dire intuitivement le fait qu'aucun événement indésirable ne se produira pendant toute la durée de vie du système (par exemple une contrainte d'échéance non respectée) et les « propriétés de vivacité » qui expriment qu'un comportement attendu se produira nécessairement (comme le fait que le système continuera à évoluer pendant toute sa durée de vie sans jamais être bloqué par des spécifications contradictoires). Typiquement pour des systèmes temps réel, les propriétés sont exprimées dans des logiques temporelles ou par des automates temporisés. Enfin, la troisième phase est l'étape de vérification à proprement parler où des algorithmes de parcours de l'espace des états permettent de décider ou non du respect des propriétés, et le cas échéant, de mettre en évidence des contre-exemples.

L'essentiel de ce premier volume du traité sera consacré à la vérification par *model-checking* et, comme nous pourrons le constater au travers des différentes contributions, la communauté francophone est très active dans ce domaine que ce soit sur les aspects théoriques ou la réalisation d'outils logiciels.

Le premier chapitre, rédigé par Bernard Berthomieu et François Vernadat, porte sur la vérification de systèmes modélisés sous forme de réseaux de Petri temporels. La forte expressivité de ce formalisme et la complémentarité des différentes techniques de vérification sont illustrées sur des exemples traités à l'aide de l'outil logiciel Tina.

Dans le second chapitre, Camille Constant, Thierry Jéron, Hervé Marchand et Vlad Rusu proposent une approche méthodologique de validation couplant vérification de la spécification du système et tests de son implantation sur plate-forme réelle d'exécution. Le test est en pratique éminemment complémentaire de la vérification sur modèles. En effet, d'une part, le test est indispensable pour nous renseigner sur la correction d'une implantation, et, parfois, vérifier la spécification ne permet que de conclure partiellement quant au respect des propriétés de bon fonctionnement. C'est le cas en particulier lorsque la complexité des modèles nécessite l'emploi de techniques de résolutions approchées.

En partant de l'exemple de la vérification du contrôleur d'un ascenseur, Stephan Merz, dans le chapitre 3, présente les fondements du *model-checking* : formalismes de transition d'états pour la modélisation, logiques temporelles pour la spécification des propriétés, algorithmes de vérification, classes de complexité des problèmes, stratégies pour maîtriser l'explosion combinatoire du nombre des états et, enfin, thématiques de recherche actuelles. Le chapitre 4, rédigé par Patricia Bouyer et François Laroussinie, poursuit cette problématique du *model-checking* dans le cas où les systèmes sont modélisés sous forme d'automates temporisés, qui permettent de manipuler des contraintes de temps explicites dans un cadre sémantique bien défini. Les propriétés qu'il est possible de vérifier et les algorithmes associés seront au cœur du propos. Ce chapitre introduit également certaines extensions des automates temporisés qui offrent plus d'expressivité dans la phase de spécification. Inversement, il sera aussi question de restrictions sous lesquelles il est possible de vérifier des propriétés hors d'atteinte dans le cadre général des automates temporisés, et d'utiliser des algorithmes plus efficaces en termes de complexité. Enfin, une présentation est faite de l'environnement de modélisation et de vérification UPPAAL.

Au travers de l'exemple d'un système industriel de perçage de pièces métalliques développé tout au long du chapitre 5, Radu Mateescu présente les possibilités de modélisation et de vérification offertes par la boite à outils CADP, conçue pour fournir une aide au développement dans le cadre général des systèmes asynchrones

qui évoluent en parallèle et communiquent par l'échange de messages. Comme données en entrée, CADP accepte des modèles dans plusieurs formalismes dont les réseaux d'automates communicants ou des spécifications de plus haut niveau en langage Lotos. CADP implémente un ensemble de techniques de transformation de modèles, comme des réductions, des techniques de vérification, allant de la simulation à la vérification de formules de logique temporelle, et fournit la possibilité de générer des tests de conformité de l'implantation.

Le chapitre 6, rédigé par Pascal Raymond, est consacré à la vérification de programmes écrits dans le langage synchrone Lustre à l'aide du *model-checker* Lesar qui a été spécifiquement développé à cet effet. Les langages synchrones connaissent un succès grandissant pour le développement des systèmes réactifs, c'est à dire des systèmes en interaction continue avec leur environnement, qui englobent en particulier les systèmes ayant des contraintes temps réel strictes. Fondés sur des modèles mathématiques de la concurrence et du temps, les langages synchrones se prêtent bien aux vérifications formelles et ce chapitre en est la meilleure illustration. Le chapitre 7, rédigé par Paul Caspi, Grégoire Hamon et Marc Pouzet, traite du langage Lucid Synchrone qui étend Lustre avec des constructions de haut niveau issues des langages fonctionnels, et permet ainsi de gagner en expressivité. Les auteurs expliquent en premier lieu pourquoi les approches synchrones offrent des éléments de solutions pertinents pour la conception de systèmes critiques. Ils dressent ensuite un historique des développements du langage puis, présentent de façon détaillée ses primitives et les concepts théoriques sous-jacents, en les illustrant sur plusieurs exemples.

On assiste progressivement depuis une quinzaine d'années à l'émergence de techniques de vérification probabiliste, en lien étroit avec les travaux sur les processus stochastiques réalisés dans la communauté de l'évaluation de performances. Ces approches sont motivées par l'intérêt pratique qu'il y a d'obtenir, pendant la phase de conception d'un système, une évaluation probabiliste de la satisfaction d'une propriété, par exemple pour dimensionner au plus juste en fonction des objectifs de sûreté de fonctionnement ou pour évaluer les performances relatives de différentes solutions en concurrence. D'autre part, une modélisation probabiliste permet généralement de mieux capturer la réalité de systèmes non déterministes, par exemple, ceux soumis à des aléas de fonctionnement. Serge Haddad et Patrice Moreaux, dans le chapitre 8, dressent un panorama des techniques de vérification probabiliste : chaînes de Markov en temps discret et continu, réseaux de Petri à loi exponentielle et à loi générale, logiques temporelles sur les chaînes de Markov et les processus de décision markoviens, etc. Enfin, sont présentés les principaux outils du domaine que sont GreatSPN, ETMCC et PRISM.

Le chapitre 9, rédigé par Marius Bozga, Susanne Graf, Laurent Mounier et Iulian Ober, présente la boîte à outils IF qui est un environnement de modélisation et de

vérification dédié aux systèmes temps réel. Au cœur de l'outil se trouve un langage de description interne, basé sur un formalisme d'automates temporisés communicants, dans lequel sont traduites des spécifications d'entrée en UML ou SDL fournies par l'utilisateur. Une fois cette traduction effectuée, peuvent être appliquées les techniques implantées dans IF, comme la réduction de modèles par analyse statique, la simulation ou la vérification par *model-checking*. L'utilisation de IF est illustrée sur une étude de cas du domaine de l'aérospatiale issue d'une collaboration avec EADS.

Pendant la phase de conception, la description architecturale du système peut servir de référentiel à tous les acteurs impliqués et fournir les informations nécessaires pour simuler, voire vérifier formellement certaines propriétés ou générer tests de conformité et code source. Le dixième et dernier chapitre de ce premier tome, rédigé par Anne-Marie Déplanche et Sébastien Faucou, traite des langages de description d'architectures (*Architecture Description Language* - ADL) en illustrant avec le langage AADL conçu et standardisé pour une utilisation dans l'avionique et l'aérospatiale, mais qui, à la date d'impression de cet ouvrage, se présente comme un bon candidat pour les systèmes temps réel en général. Après avoir examiné les besoins spécifiques rencontrés dans le contexte du temps réel, le langage AADL et les outils logiciels associés sont présentés puis appliqués sur l'exemple d'un système de contrôle échantillonné en boucle fermée.

Pour conclure, je voudrais exprimer ma gratitude à M. Jean-Charles Pomerol, directeur éditorial, pour la confiance qu'il a accepté de me témoigner et remercier chaleureusement les auteurs et les relecteurs des chapitres qui ont offert de leur temps et partagé leur expertise dans la rédaction de ce livre. Je remercie également Françoise Simonot-Lion et Stephan Merz pour leurs remarques et suggestions sur ce chapitre introductif.

Enfin, j'espère que cet ouvrage sera, pour vous, chers lecteurs, une source d'information complète, fiable et bien documentée sur les systèmes temps réel. Je vous en souhaite une excellente lecture,

Nicolas NAVET