SYSTÈMES TEMPS RÉEL EMBARQUÉS

Francis Cottet • Emmanuel Grolleau
Sébastien Gérard • Jérôme Hugues
Yassine Ouhammou • Sara Tucci-Piergiovanni

SYSTÈMES TEMPS RÉEL EMBARQUÉS

Spécification, conception, implémentation et validation temporelle

2e édition

DUNOD

Illustration de couverture

© La station spatiale internationale (ISS)
CNES/ill./Ducros David, 2008

5 rue Laromiguière, 75005 Paris
www.dunod.com

ISBN 978-2-10-071331-8

Table des matières

Annexes

Avant-propos

Les applications informatiques embarquées ont envahi l'environnement industriel et notre vie quotidienne. Depuis quelques décennies, les besoins de plus en plus accrus en termes de technicité ont conduit à intégrer une très forte automatisation dans tous les produits industriels ou destinés à l'usage « grand public ». La liste infiniment longue des exemples contient des produits aussi divers qu'un avion civil ou militaire, un train, un véhicule automobile, un téléphone mobile, un four à micro-ondes, une console de jeu, un satellite d'exploration, etc. Le dénominateur commun à toutes les applications adressées par les méthodes proposées dans cet ouvrage est la fourniture de fonctionnalités toujours plus sophistiquées : interface homme-machine (écran couleur de haute définition, écran tactile, commande vocale…), nombre élevé de fonctions débordant largement l'utilisation de base du produit (visualisation des commandes, liaison Internet…), sûreté de fonctionnement (robustesse, tolérance aux fautes, répartition, maintenance rapide et aisée…).

Pour ces raisons, les trois grands domaines permettant ces développements, que sont l'électronique, l'automatique et l'informatique, ont dû progresser et s'adapter :

- **Électronique :** processeur multifonction (microcontrôleur), processeurs multi-cœurs, processeur à faible consommation, réalisation de circuits électroniques dédiés (FPGA), etc.
- **Automatique :** lois de régulations adaptées, régulation numérique, etc.
- **Informatique :** méthodes et méthodologies de développement, systèmes d'exploitation ou exécutifs embarqués, langages applicatifs, méthodes de tests et de validations, etc.

Cet ouvrage est résolument orienté informatique, et les domaines de l'électronique et de l'automatique ne sont pas ici le propos. Toutefois, étant donné le développement lié entre les deux parties « matériel et logiciel », une présentation succincte des aspects matériels nécessaires à l'appréhension des systèmes embarqués est faite. Nous allons nous intéresser particulièrement à l'aspect génie logiciel, c'est-à-dire la ou les méthodes permettant de développer correctement ces applications.

La signification du terme « correctement » est précisée dans le chapitre suivant, mais nous pouvons déjà annoncer que ces applications doivent être développées avec un grand souci de rigueur étant donné que leurs utilisations peuvent avoir un impact financier important, un effet nuisible sur l'environnement ou plus gravement mettre en jeu des vies humaines. Aussi la méthodologie de développement des applications de contrôle/commande doit assurer une qualité de réalisation en termes de fiabilité, d'efficacité, de maintenabilité, d'évolutivité, etc.

Il est important de noter qu'il n'est pas possible de parler des applications embarquées comme d'un ensemble homogène au sens de leur réalisation. En effet, entre les développements de l'application informatique gérant un four à micro-ondes et celle pilotant une navette spatiale, la distance est immense : criticité de l'application, taille du logiciel, évolution de l'application, etc. Le seul lien en commun est que ces applications mettent en relation un programme informatique avec un procédé externe.

D'autre part, le développement des applications de petite taille ou de taille moyenne a souvent été conduit par des professionnels du monde de l'automatique ou de l'électronique. Les régulations de type électromécanique ou électronique analogique ont rapidement fait place à des systèmes purement numériques ; seuls les capteurs et les actionneurs, faisant le lien vers le monde réel, sont toujours analogiques. Les concepteurs de ces applications ont des méthodes basées sur l'aspect fonctionnel : schémas blocs, grafcets, etc. En effet, la spécification et la conception de telles applications sont plus aisées lorsqu'elles sont pensées en termes de fonctions ou de traitements des données. Aussi les langages dédiés à ce type d'applications sont tout naturellement des langages fonctionnels à exécution séquentielle comme le langage C, les assembleurs ou le langage flot de donnée LabVIEW.

Dans le domaine informatique, depuis l'avènement du langage UML (*Unified Modeling Language*) à la fin des années 1990, accompagné de mécanismes d'extension permettant de l'étendre ou de le restreindre (profil SysML – *Systems Modeling Language*), d'en enrichir la sémantique, ou d'en spécialiser l'usage (DSL – *Domain Specific Language*), les technologies basées sur l'ingénierie des modèles se sont développées jusqu'à atteindre aujourd'hui le domaine des systèmes embarqués, et en particulier des systèmes embarqués critiques. L'utilisation de l'ingénierie des modèles permet, par le biais de modèles d'abstraction communs, de faciliter le passage d'un modèle de représentation à un autre, ou même de générer automatiquement des programmes à partir de leur modèle.

Connaître les différents diagrammes UML, SysML, ou même un ensemble de DSL, mais sans méthodologie de conception, est analogue à vouloir apprendre à programmer en apprenant la syntaxe d'un langage de programmation sans connaître l'algorithmique. Il faut en effet d'abord savoir mettre en œuvre un algorithme avant de pouvoir l'appliquer dans un langage donné. Cet ouvrage présente donc une

méthodologie complète de développement d'applications embarquées critiques. Les méthodes choisies l'ont été pour l'ensemble des concepts qu'elles permettent de manipuler, leur simplicité relative de prise en main, leur utilisation dans l'industrie et la facilité de se procurer gratuitement des outils permettant de les utiliser.

La méthodologie suppose l'existence d'un cahier des charges, éventuellement couplé à un ensemble d'exigences fonctionnelles et non fonctionnelles (notamment temporelles). La première étape consiste à modéliser l'aspect fonctionnel du système à développer en utilisant des diagrammes SysML permettant une décomposition fonctionnelle structurée des fonctionnalités à accomplir. Chaque fonctionnalité peut alors être exprimée formellement lorsque cela est nécessaire en utilisant un formalisme basé sur des machines à état (automates de Harel) présent dans UML et SysML. Étant donné que de nombreuses implémentations se basent sur une programmation multitâche qui permet de facilement prendre en compte le parallélisme inhérent à la plupart des systèmes de contrôle/commande, la phase conception est orientée multitâche. Le langage AADL (*Architecture Analysis and Design Language*) a été choisi pour exprimer dans un même langage l'architecture matérielle (processeurs, réseaux, périphériques) exécutant le système et son architecture logicielle parallèle (notion de tâches, processus, communications). Ce DSL est très utilisé en aéronautique.

Le premier chapitre présente l'environnement de développement des systèmes embarqués en décrivant la spécificité de ces applications en termes d'architectures logicielles et matérielles. Le second chapitre traite de la méthode de spécification fonctionnelle choisie SysML et ses IDB (*Internal Block Diagrams*), ainsi que des automates de Harel permettant d'exprimer formellement le comportement de certaines fonctions. La méthode de conception basée sur AADL est décrite dans le chapitre 3. Les environnements matériels et logiciels (exécutifs temps réel) très particuliers de ces applications sont présentés dans les chapitres 4 et 5 afin de mieux comprendre la partie implémentation de ces systèmes de contrôle/commande. Le chapitre 6 est dédié à l'implémentation des applications de contrôle/commande en déclinant trois environnements : noyau temps réel et langage de type langage C, langage Ada et enfin un environnement spécifique basé sur LabVIEW. Les précédents chapitres sont illustrés par des exemples simples mais réalistes ; en revanche le chapitre 7 propose le développement complet d'une application réelle industrielle. Enfin, le chapitre 8 ouvre le développement de ces applications vers des aspects avancés concernant la validation des exigences temporelles.

1 Le développement des systèmes embarqués temps réel

1.1 Introduction

1.1.1 Définitions

La plupart des systèmes embarqués temps réel sont avant tout des systèmes de **contrôle-commande**. Un système de contrôle-commande est un système informatique de contrôle de **procédé**. Le terme procédé est un terme générique désignant un système physique contrôlé.

Afin de contrôler le procédé, le système informatique est en relation avec l'environnement physique externe par l'intermédiaire de **capteurs** et/ou d'**actionneurs**. Les grandeurs physiques acquises grâce aux capteurs permettent au système de contrôle-commande de s'informer de l'état du procédé ou de son environnement. Le système de contrôle-commande, à partir de consignes décrivant l'état voulu du procédé, calcule des commandes qui sont alors appliquées sur le procédé par l'intermédiaire d'actionneurs. Donnons ainsi une définition générale d'un système de contrôle-commande (figure 1.1) :

« Un système de contrôle-commande reçoit des informations sur l'état du procédé externe, traite ces données et, en fonction du résultat, évalue une décision qui agit sur cet environnement extérieur afin d'assurer un ***état voulu*** *».*

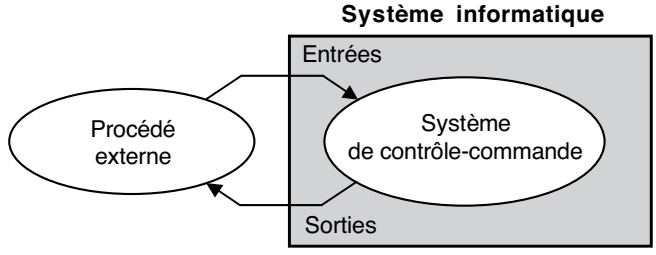

Figure 1.1 Représentation schématique d'un système de contrôle commande.

Exemple 1 : la figure 1.2 représente un exemple de système de contrôle-commande pour un drone. Un drone est un aéronef autonome, ou semi-autonome. L'un des buts principaux du système de contrôle-commande est de contrôler l'attitude (tangage, roulis) et la vitesse du drone : cela s'appelle la commande de vol. Le système de contrôle-commande est exécuté par un calculateur qu'on appelle **microcontrôleur**. Les capteurs utilisés sont une centrale inertielle, permettant de connaître les angles de tangage et roulis du drone, et un récepteur GPS (*Global Positionning System)* permettant notamment de connaître la position absolue (latitude, longitude), la vitesse par rapport au repère sol et la vitesse ascensionnelle. Un modem sans fil permet au drone de recevoir des consignes de la part d'un opérateur (par exemple un angle de roulis, une vitesse et une vitesse ascensionnelle), et de lui envoyer des informations sur le vol. À la fois capteur (lit des informations en provenance de l'opérateur) et actionneur (permet l'affichage ou l'enregistrement d'informations pour l'opérateur), ce modem est qualifié d'**organe de dialogue**. Enfin, trois actionneurs permettent de commander respectivement la puissance du moteur et la position des deux ailevons. Le différentiel des ailevons (l'un plus haut, l'autre plus bas), permet de donner une vitesse angulaire sur l'axe de roulis de l'aéronef, alors que leur position parallèle par rapport à l'angle d'équilibre permet d'appliquer une vitesse angulaire sur l'axe de tangage. La vitesse ascensionnelle dépend de la puissance du moteur et des angles de tangage et roulis.

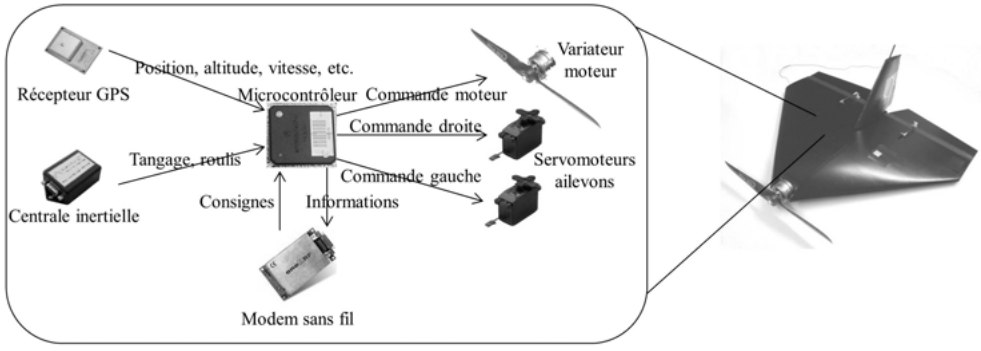

Figure 1.2 Système de contrôle-commande embarqué sur un mini-drone.

Un **système embarqué** est un système informatique dont les moyens de calcul sont embarqués sur le procédé contrôlé. Le fait d'embarquer les moyens de calcul implique, en plus des contraintes d'encombrement (taille, poids, forme), des contraintes de consommation d'énergie puisque l'alimentation électrique des éléments de calcul est soit embarquée (batteries, carburant, etc.), soit ambiante (panneaux solaires, etc.). À technologie équivalente, l'énergie consommée par un calculateur est fonction du carré de la vitesse de celui-ci ; en d'autres termes, plus le calculateur est performant,

plus l'énergie consommée est importante, avec une relation quadratique. Un système embarqué se caractérise donc souvent par des ressources de calcul dimensionnées (ou à dimensionner) au plus juste en fonction des besoins en calcul.

Un système **temps réel** est un système informatique soumis à des contraintes de temps. Une définition du temps réel peut être :

> *« Un système est dit temps réel si l'exactitude des applications ne dépend pas seulement du résultat mais aussi du temps auquel ce résultat est produit. ».*

Une autre définition du temps réel peut être :

> *« Un système est dit temps réel s'il doit s'exécuter suffisamment vite par rapport à la dynamique du procédé contrôlé. La notion relative de vitesse s'exprime par des* ***contraintes temporelles****. »*

La notion « suffisamment vite » est relative à la dynamique du procédé contrôlé. Par exemple, le drone de la figure 1.2, est considéré comme un système temps réel. De par sa forme d'aile delta, il est très instable en roulis ; par conséquent, il est important que la commande de vol se rafraîchisse fréquemment, par exemple toutes les 20 millisecondes. Si le drone avait une forme de planeur, plus stable, une fréquence de rafraîchissement plus faible (de l'ordre de 10 Hz, soit 100 millisecondes) pourrait peut-être suffire à assurer une commande de vol acceptable.

Un système **critique** est un système informatique pour lequel une défaillance, totale ou partielle, du système, peut entraîner des conséquences graves, par exemple économiques, humaines, écologiques, etc. La défaillance d'un drone d'observation entraîne des conséquences économiques (perte de l'engin) mais surtout peut entraîner des conséquences humaines en cas de collision, moteur allumé, avec une personne. Un drone est donc considéré comme un système critique lorsqu'il évolue dans une zone habitée. Un système est **temps réel critique** si une défaillance temporelle (le système ne réagit pas suffisamment vite par rapport à la dynamique du procédé) peut aussi entraîner des conséquences graves. Le drone peut être considéré comme temps réel critique, puisque le fait de ne pas être capable d'appliquer les commandes de vol à temps peut aboutir à une perte de contrôle. La plupart des systèmes de transport sont des systèmes embarqués de contrôle-commande, temps réel critique.

Une grande difficulté inhérente à la validation des systèmes de contrôle-commande provient du fait que les comportements ne sont pas reproductibles :

- Indépendance du résultat produit par rapport à la vitesse d'exécution. Le résultat d'un calcul effectué à partir de données d'entrée similaires est indépendant de la vitesse du calculateur. En revanche, l'état stable d'un procédé dépend de

la dynamique du procédé par rapport à la vitesse d'exécution du système de contrôle-commande.

- Comportement reproductible. Un calcul effectué à partir de données d'entrée identiques donne toujours le même résultat. En revanche, dans le cas de données d'entrée (grandeurs physiques) obtenues par des capteurs, le système de contrôle-commande travaille sur un domaine de données réelles approximées qui sont très rarement identiques.

Cette définition des systèmes de contrôle-commande ayant été faite, nous pouvons replacer ces systèmes par rapport aux autres systèmes informatiques en faisant trois catégories (voir figure 1.3) :

- Les **systèmes transformationnels** qui utilisent des données fournies à l'initialisation par l'utilisateur. Ces données, leurs traitements et l'obtention du résultat n'ont aucune contrainte de temps.
- Les **systèmes interactifs** dans le sens où les données sont produites par interaction avec l'environnement sous différentes formes (clavier, fichier, réseaux, souris, etc.). Mais le temps n'intervient pas en tant que tel si ce n'est avec un aspect confort de travail ou qualité de service.
- Les **systèmes de contrôle-commande ou réactifs** qui sont en relation avec l'environnement physique réel pour les données capteur/actionneur ; l'aspect « temps » a une place importante sous la forme d'un temps de réaction, d'une échéance à respecter, etc.

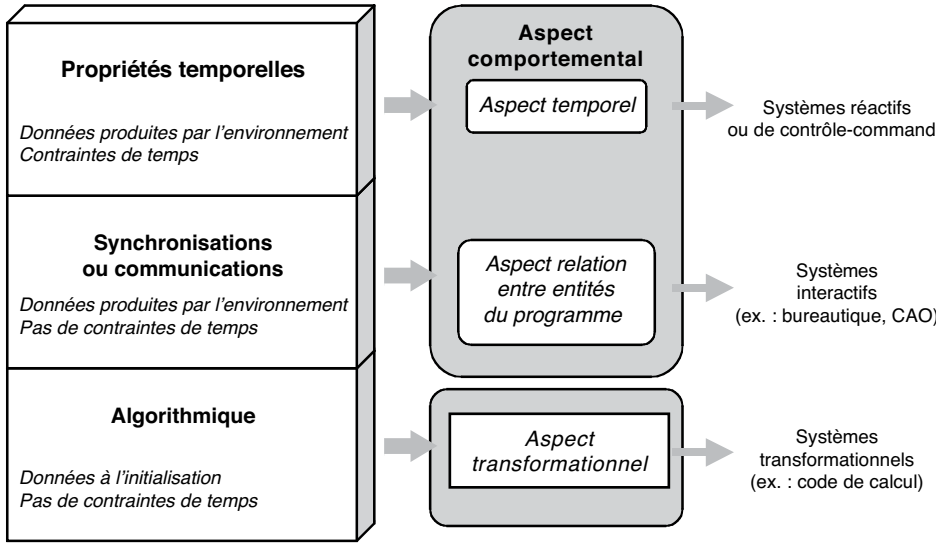

Figure 1.3 Comparaison des systèmes de contrôle-commande par rapport aux autres applications informatiques.

Nous terminons cette section par des définitions qualifiant des systèmes temps réel ayant des spécifications particulières. Ces contraintes temporelles peuvent être de plusieurs types :

- **Contraintes temporelles relatives** ou lâches (temps réel mou : *soft real-time*) : les fautes temporelles sont tolérables (Ex. : jeux vidéo, applications multimédias, téléphonie mobile…).
- **Contraintes temporelles strictes** ou dures (temps réel dur : *hard real-time*) : les fautes temporelles ne sont pas tolérables (Ex. : fonctions critiques avionique, véhicules spatiaux, automobile, transport ferroviaire…).
- **Contraintes temporelles fermes** (temps réel ferme : *firm real-time*) : les fautes temporelles sont autorisées dans une certaine limite, par exemple une erreur toutes les trois exécutions au plus.
- **Systèmes multi-critiques** : les sous-systèmes composant le système sont caractérisés par des degrés de criticité. Par exemple en avionique civile, la DO-178C caractérise les sous-systèmes par un niveau de criticité appelé DAL (*Design Assurance Level*), allant de niveau A (Ex. : commande de vol), catastrophique en cas de défaillance, à niveau E (Ex. : divertissement en vol), sans effet sur la sécurité.

1.1.2 Principales caractéristiques des systèmes temps réel

Considérons un exemple représentatif d'une application temps réel de contrôle-commande représenté sur la figure 1.4. Cet exemple de contrôle-commande d'un moteur à combustion est repris de façon détaillée dans le chapitre suivant. Le contrôle-commande de cette application est fait par l'intermédiaire d'un ensemble de capteurs et d'actionneurs (pédale d'accélérateur, température air, pression air, température eau, rotation vilebrequin, capteurs de pollution, injection essence, allumage, admission air, etc.) et d'une connexion au réseau interne à l'automobile. L'analyse de cet exemple d'application permet de mettre en exergue les principales caractéristiques des systèmes de contrôle-commande :

- **Grande diversité des dispositifs d'entrées/sorties** : les données à acquérir qui sont fournies par les capteurs et les données à fournir aux actionneurs sont de types très variés (continu, discret, tout ou rien ou analogique). Il est aussi nécessaire de piloter un bus de terrain pour les communications.
- **Prise en compte des comportements concurrents** : l'ensemble de ces données physiques qui arrivent de l'extérieur et le réseau qui permet de recevoir des messages ne sont pas synchronisés au niveau de leurs évolutions, par conséquent, le système informatique doit être capable d'accepter ces variations simultanées des paramètres.
- **Respect des contraintes temporelles** : la caractéristique précédente impose de la part du système informatique d'avoir une réactivité suffisante pour prendre

en compte tous ces comportements concurrents et en réponse à ceux-ci, de faire une commande en respectant un délai compatible avec la dynamique du système.

- **Sûreté de fonctionnement** : les systèmes de type contrôle-commande mettent souvent en jeu des applications qui demandent un niveau important de sécurité pour raisons de coût ou de vies humaines. Pour répondre à cette demande, il est nécessaire de mettre en œuvre toutes les réponses de la sûreté de fonctionnement (développements sûrs, tests, méthodes formelles, prévisibilité, déterminisme, continuité de service, tolérance aux fautes, redondance, ségrégation des moyens de calcul et de communication, etc.).

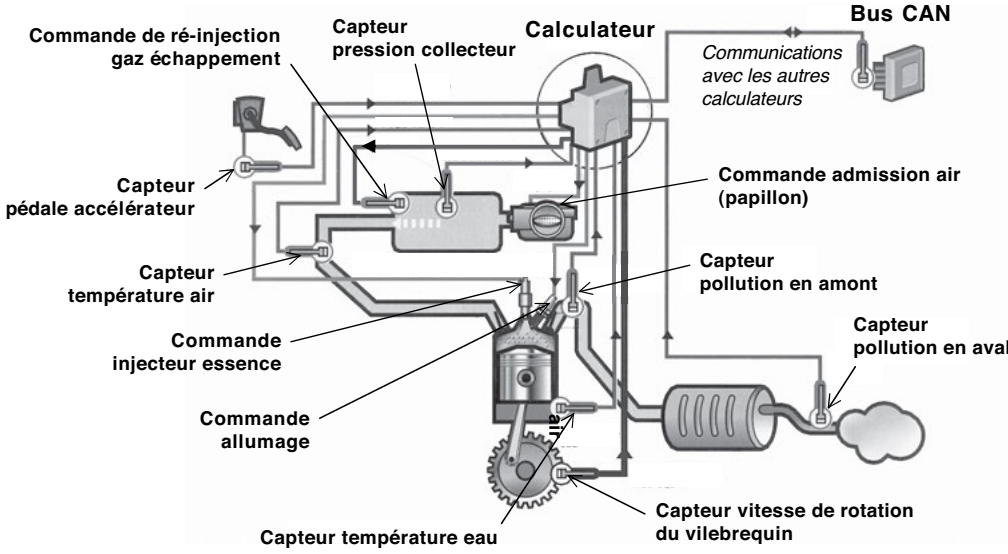

Figure 1.4 Exemple d'une application de contrôle-commande d'un moteur à combustion.

1.1.3 Caractéristiques temporelles des systèmes temps réel

Les contraintes temporelles qui sont classiquement présentées sont des **contraintes de bout en bout**, appelées aussi **contraintes de latence**. Ces contraintes représentent le délai maximal entre lecture de l'état du système (lecture des capteurs) et commande de celui-ci (commande des actionneurs). Par exemple, la commande de vol du drone présenté dans l'Exemple 1, puisqu'elle doit s'exécuter à 50 Hz, c'est-à-dire avec une période de 20 millisecondes, se verra vraisemblablement munie de contraintes temporelles de bout en bout sur la commande de vol, par exemple de 20 millisecondes, afin d'obliger le système à s'exécuter une fois par période.

Le respect du protocole de communication avec les capteurs, actionneurs, ou bus de communication est une autre source, très importante, de contraintes temporelles. Ainsi, à chaque fois qu'une trame est disponible sur un réseau, le système doit la lire et soit la traiter, soit la stocker pour traitement ultérieur, sous peine de la voir être remplacée (ou « écrasée ») par la trame suivante.

Comme nous le verrons dans le chapitre 8, il est nécessaire de préciser et de formaliser les caractéristiques temporelles d'un système. Cette caractérisation peut prendre de nombreuses formulations. Ainsi, nous pouvons définir de manière non exhaustive :

- **Durée d'exécution** d'une activité : l'activité d'une application, qui peut être l'enchaînement de plusieurs activités élémentaires (acquisition, traitement, commande, affichage…), possède une durée d'exécution qui peut être mesurée de diverses manières. Cette durée n'est pas constante à chaque occurrence de cette activité puisque les programmes et les enchaînements de programmes ne sont pas toujours identiques (branchement conditionnel, itération, synchronisation…).
- Cadence de répétition ou **périodicité** d'une activité : l'acquisition d'une donnée ou la commande d'un actionneur peuvent nécessiter une régularité liée par exemple à la fréquence d'échantillonnage.
- Date au plus tôt ou **date de réveil** : dans certains cas, un traitement doit être déclenché à une date précise relative par rapport au début de l'exécution de l'application ou absolue (plus rarement). Cette date de réveil n'implique pas obligatoirement l'exécution ; il peut y avoir un délai dû à l'indisponibilité du processeur.
- **Date de démarrage** : cet instant correspond à l'exécution effective de l'activité.
- **Date de fin** : instant correspondant à la fin de l'exécution de l'activité.
- Date au plus tard ou **échéance** : le traitement ou la commande d'un actionneur doit être terminé à un instant fixé par rapport au début de l'exécution de l'application. Dans le cas d'applications à contraintes temporelles strictes, cette échéance doit être respectée de façon impérative, sinon il y a faute temporelle et l'application est déclarée non valide.
- **Temps de réponse** : cette caractéristique peut s'appliquer à une activité de régulation ou à un ensemble d'activités de régulation ; elle est directement liée à la dynamique du système. Ce paramètre correspond à la différence entre la date de réveil et la date de fin de l'activité.
- **Gigue temporelle** : ce paramètre caractérise la répétabilité d'une activité au fur et mesure de ses occurrences. En effet, entre deux exécutions successives d'une même activité, ses caractéristiques temporelles peuvent changer : date d'activation, durée d'exécution, temps de réponse, etc.

1.1.4 Quelques exemples d'applications

Nous pouvons citer quelques exemples d'applications temps réel de contrôle-commande :

- **Système de transport** : que cela soit pour le transport terrestre (véhicule de tourisme, utilitaire, poids lourd, etc.), ferroviaire, aérien, ou spatial, les systèmes de transport sont caractérisés par une forte criticité et une forte complexité due à l'expansion du nombre de fonctions et la nécessité de tolérance aux fautes. Ils sont typiquement décomposés en sous-systèmes interconnectés par un ensemble de bus de terrains.
- **Drone volant, roulant, navigant, ou sous-marin** : ce type d'applications est de plus en plus répandu, et de plus en plus d'autonomie est donnée aux drones. Qu'ils soient d'observation ou tactiques, ce type d'application trouve une large place aussi bien dans les applications militaires que civiles (exploration de zone radioactive après un accident, planétaire, sous-marine, cinéma et télévision, etc.).
- **Robot de production** : un robot, réalisant une activité spécifique (peinture, assemblage, tri) sur une chaîne de production, doit effectuer son travail en des temps fixés par la cadence de fabrication. S'il agit trop tôt ou trop tard, l'objet manufacturier traité sera détruit ou endommagé conduisant à des conséquences financières ou humaines graves (oubli d'un ou plusieurs rivets sur un avion).
- **Téléphone mobile** : le système de contrôle-commande doit remplir plusieurs fonctions dont certaines ont des contraintes temporelles fortes pour avoir une bonne qualité de service (QoS : *quality of service*). Ainsi, la première fonction est de transmettre et de recevoir les signaux de la parole (577 µs de parole émises toutes les 4,6 ms et 577 µs de parole reçues toutes les 4,6 ms à des instants différents). En parallèle, il est nécessaire de localiser en permanence le relais le plus proche et donc de synchroniser les envois par rapport à cette distance (plus tôt si la distance augmente et plus tard si la distance diminue). Des messages de comptes rendus de la communication sont aussi émis avec une périodicité de plusieurs secondes. Les contraintes temporelles imposées au système doivent être imperceptibles à l'utilisateur.
- **Système de vidéoconférence** : ce système doit permettre l'émission et la réception d'images numérisées à une cadence de 20 à 25 images par seconde pour avoir une bonne qualité de service. Afin de minimiser le débit du réseau, une compression des images est effectuée. D'autre part la parole doit aussi être transmise. Bien que correspondant à un débit d'information moindre, la régularité de la transmission, qualifiée par une gigue temporelle, est nécessaire pour une reproduction correcte. De plus ce signal doit être synchronisé avec le flux d'images. Ces traitements (numérisations images et parole, transmission, réception, synchronisation…) sont réalisés en cascade, mais avec une cohérence précise.

- **Pilotage d'un procédé de fabrication** (fonderie, laminoir, four verrier…) : par exemple la fabrication d'une bobine d'aluminium (laminage à froid) exige un contrôle en temps réel de la qualité (épaisseur et planéité). Cette vérification en production de la planéité nécessite une analyse fréquentielle (FFT) qui induit un coût important de traitement. Le système doit donc réaliser l'acquisition d'un grand nombre de mesures (246 Ko/s) et traiter ces données (moyenne, FFT…) à la période de 4 ms. Ensuite, il affiche un compte-rendu sur l'écran de l'opérateur toutes les 200 ms et enfin imprime ces résultats détaillés toutes les 2 s. Un fonctionnement non correct de ce système de contrôle de la qualité peut avoir des conséquences financières importantes : production non conforme à la spécification demandée.

1.2 Architecture des applications temps réel

1.2.1 Architecture logicielle des applications temps réel

1.2.1.1 Architecture multitâche

Le comportement concurrent des événements et grandeurs physiques externes amène à décrire l'environnement comme un système fortement parallèle. Cela conduit naturellement à adapter les méthodes de conception et de réalisation du système de contrôle-commande d'un tel environnement à ce parallélisme. Aussi, l'architecture la mieux adaptée pour répondre à ce comportement parallèle du procédé externe est une **architecture multitâche**. Ainsi au parallélisme de l'environnement, la réponse est le parallélisme de conception. Nous pouvons définir la tâche ou activité ou processus comme « une entité d'exécution et de structuration de l'application ». Cette architecture logicielle multitâche facilite la conception et la mise en œuvre et surtout augmente l'évolutivité de l'application réalisée.

D'une manière très générique, la figure 1.5 donne l'architecture logicielle d'une application de contrôle-commande multitâche. Nous pouvons ainsi découper cet ensemble de tâches ou activités selon les groupes suivants :

- Tâches d'entrées/sorties : ces tâches permettent d'accéder aux données externes par l'intermédiaire de cartes d'entrées/sorties et ensuite de capteurs et d'actionneurs directement liés au procédé géré. Ces tâches peuvent être activées de façon régulière ou par interruption.
- Tâches de traitement : ces tâches constituent le cœur de l'application. Elles intègrent des traitements de signaux (analyse spectrale, corrélation, traitement d'images, etc.) ou des lois de commande (régulation tout ou rien, régulation du premier ordre, régulation PID, etc.). Dans le cadre de cet ouvrage, nous considérerons ces tâches comme des boîtes noires, c'est-à-dire que le traitement effectué

par ces tâches relève des domaines comme le traitement du signal, le traitement d'images ou l'automatique, disciplines qui débordent largement le contexte de ce livre.

- Tâches de gestion de l'interface utilisateur : ces tâches permettent de présenter l'état du procédé ou de sa gestion à l'utilisateur. En réponse, l'opérateur peut modifier les consignes données ou changer les commandes. Ces tâches peuvent être très complexes et coûteuses en temps de calcul si l'interface gérée est de taille importante (tableau de bord) ou de type graphique (représentation 3D).

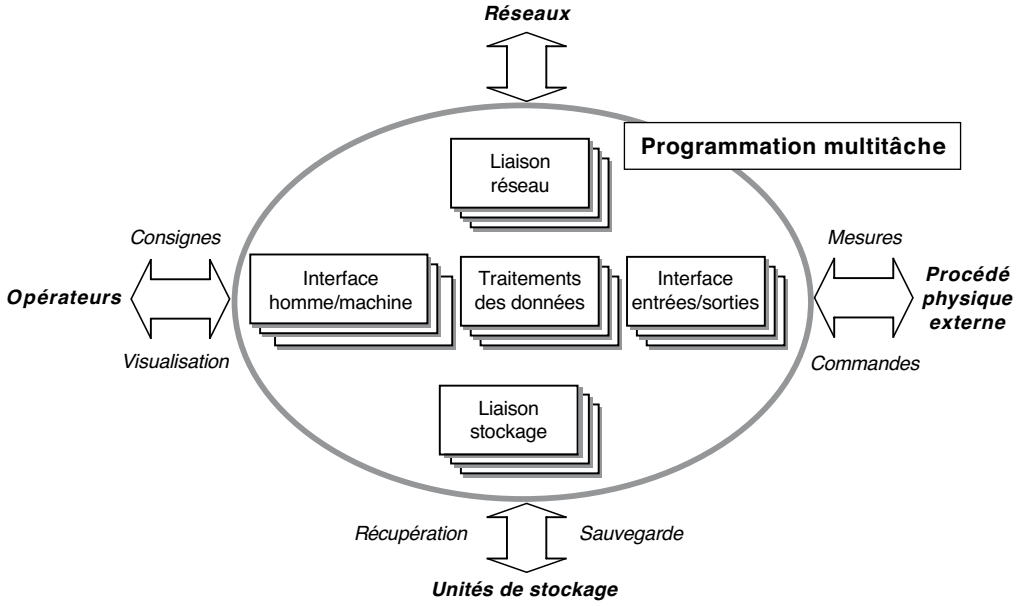

Figure 1.5 Architecture logicielle d'une application de contrôle-commande multitâche.

- Tâches de communications : ces tâches sont destinées à gérer les messages envoyés ou reçus à travers un ou plusieurs réseaux ou bus de terrain. Si ce type de tâches existe, l'application est dite distribuée ou répartie.
- Tâches de sauvegarde : ces tâches permettent de stocker l'état du système à des instants fixés. Cette sauvegarde peut être utilisée a posteriori pour analyser le fonctionnement de l'application ou lors d'une reprise d'exécution à une étape précédente.

Après l'analyse et la conception de l'application, nous obtenons un ensemble de tâches ou activités qui coopèrent afin de réaliser le contrôle-commande du procédé géré. Ces tâches appartiennent aux différents groupes listés précédemment : tâches d'entrées/sortie, tâches de traitement, tâches de gestion de l'interface utilisateur,

tâches de communications et tâches de sauvegarde. Ce découpage purement fonctionnel peut être modifié dans certains cas en utilisant une conception tournée vers les entités ou « objets » à contrôler. Cet aspect de la conception et de la mise en œuvre est présenté dans les chapitres suivants.

Les tâches obtenues, qui constituent l'application de contrôle-commande, ne sont pas des entités d'exécution indépendantes. En effet certaines tâches sont connectées vers l'extérieur pour les entrées/sorties. De plus elles peuvent être liées par des relations de type (voir figure 1.6) :

- **Synchronisation :** cela se traduit par une relation de précédence d'exécution entre les tâches ;
- **Communications :** à la notion de précédence, traduite par la synchronisation, s'ajoute le transfert de données entre les tâches ;
- **Partage de ressources :** les tâches utilisent des éléments mis en commun au niveau du système comme des zones mémoire, des cartes d'entrées/sorties, cartes réseau, etc. Certaines de ces ressources, comme par exemple les zones mémoire, ne sont pas ou ne doivent pas être accessibles, pour avoir un fonctionnement correct, par plus d'une tâche à la fois, elles sont dites **ressources critiques**.

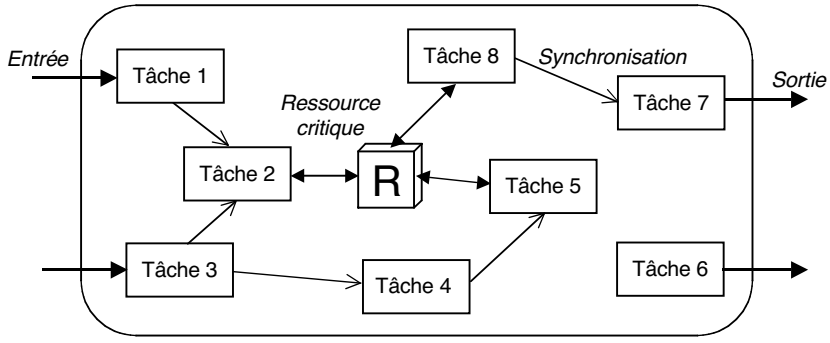

Figure 1.6 Représentation schématique de l'architecture multitâche d'une application de contrôle-commande.

Ces différents concepts sont étudiés de façon détaillée dans le chapitre 4.

1.2.1.2 Modèles d'exécution et ordonnancement

Cette architecture logicielle peut être vue comme un ensemble de tâches synchronisées, communicantes et partageant des ressources critiques. Le rôle essentiel du système informatique est donc de gérer l'enchaînement et la concurrence des tâches en optimisant l'occupation du processeur, cette fonction est appelée l'**ordonnancement**. L'ordonnancement est un point crucial des systèmes temps réel ; en effet

l'ordonnancement va déterminer le comportement temporel et être le garant du respect des contraintes de temps imposées à l'exécution de l'application.

Nous pouvons distinguer deux modèles d'exécution des systèmes temps réel : l'exécution dite **synchrone** et l'exécution **asynchrone**. Nous allons présenter ces deux modèles d'exécution à l'aide d'un modèle d'application très simple. Cette application, constituée d'un ensemble de tâches pour gérer le procédé, intègre en particulier les deux tâches suivantes :

- Tâche de lecture des données entrées par l'opérateur à l'aide d'un clavier, appelée « Lecture_consigne ». L'intervention humaine fait que cette tâche peut être longue.
- Tâche d'alarme qui se déclenche sur un événement d'alerte correspondant au dépassement d'un paramètre critique, appelée « Alarme ». Celle-ci doit s'exécuter au plus vite pour éviter l'endommagement du procédé.

Pour mettre en avant les différences entre les deux modèles d'exécution nous allons étudier la situation dans laquelle la tâche « Lecture_consigne » s'exécute et la tâche « Alarme » demande son exécution alors que la tâche « Lecture_consigne » n'est pas terminée.

Dans le modèle d'**exécution synchrone**, la perception de l'occurrence de tout événement par le système est différée du temps d'exécution de la tâche en cours. Dans l'exemple proposé, nous pouvons constater que la prise en compte d'un signal d'alerte n'est effective que lors de la fin de la tâche « Lecture_consigne » (voir figure 1.7). D'un point de vue du procédé, la réaction est perçue comme différée, alors que du point de vue du système informatique, elle est perçue comme immédiate. L'occurrence des événements externes a donc été artificiellement synchronisée avec le système informatique, d'où le nom d'exécution synchrone.

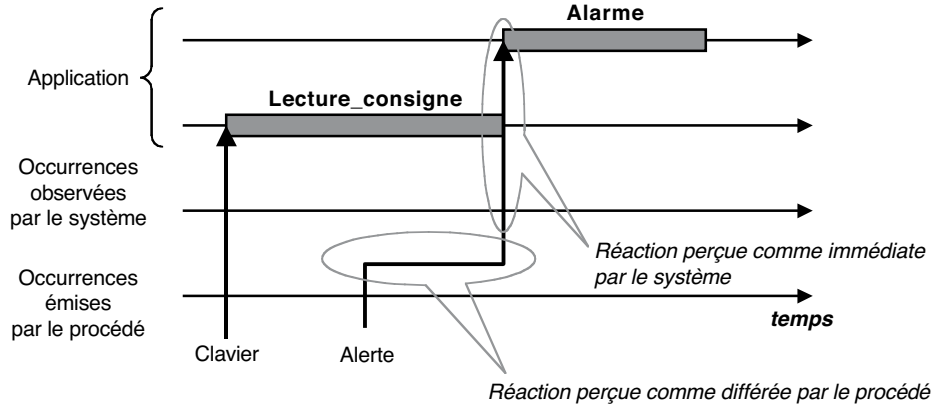

Figure 1.7 Modèle d'exécution synchrone d'une application de contrôle-commande.

Ce retard peut affecter la prise en compte de n'importe quel événement, quelle qu'en soit la gravité pour l'application. Il faut donc vérifier que l'architecture opérationnelle choisie permettra de prendre en compte les contraintes temporelles : hypothèse de la fenêtre de visibilité des événements ou d'instantanéité des actions. La capacité du système à appréhender un événement externe est caractérisée par la durée de la tâche la plus longue puisque les tâches sont non interruptibles ou **non préemtibles**.

Dans le cas du modèle synchrone d'exécution, nous avons un système d'ordonnancement complètement prévisible et, en conséquence, il est possible en faisant une analyse exhaustive de l'exécution de produire une séquence d'exécution qui est jouée de façon répétitive. Cette étude de la séquence est appelée analyse de l'ordonnancement **hors-ligne**. L'ordonnancement peut se réduire à un séquencement. Nous avons alors un environnement informatique très simple de l'application développée puisqu'il se réduit à une liste de tâches à exécuter. L'environnement informatique pour piloter cette liste de tâches se réduit à un système très simple : un **séquenceur**.

Dans le modèle d'**exécution asynchrone**, l'occurrence de tout événement est immédiatement prise en compte par le système pour tenir compte de l'urgence ou de l'importance. Dans l'exemple proposé, nous pouvons constater que la prise en compte d'un signal d'alerte est immédiate sans attendre la fin de la tâche « Lecture_consigne » (voir figure 1.8). La prise en compte de l'événement « alerte » est identique pour le procédé et le système informatique. L'occurrence des événements externes n'est pas synchronisée avec le système informatique, d'où le nom d'exécution asynchrone.

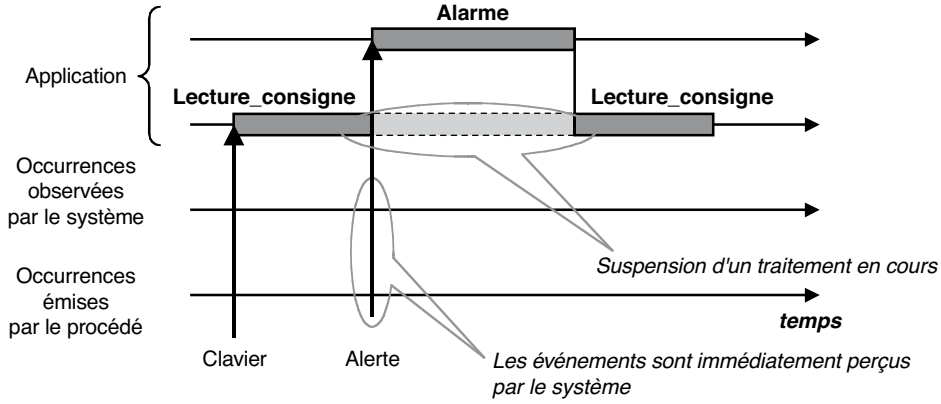

Figure 1.8 Modèle d'exécution asynchrone d'une application de contrôle-commande.

Dans ce contexte, nous avons des tâches qui sont interruptibles ou **préemtibles**. En conséquence, l'ordonnancement n'est pas totalement prévisible et l'analyse de l'exécution des tâches doit se faire **en ligne** par simulation ou par test. Cela nécessite

l'utilisation d'un gestionnaire centralisé des événements et de la décision d'exécution : **exécutif ou noyau temps réel**.

Pour terminer cette section nous allons rappeler trois définitions importantes que nous avons utilisées et fixer le contexte de cet ouvrage. Nous avons ainsi défini :

- Tâche non préemptible ou préemptible :
  - Une tâche non préemptible ne peut être interrompue qu'à des endroits spécifiques et à la demande de la tâche elle-même : fin_de_tâche, attente_signal… Dans ce cas, la programmation est plus simple et aucun mécanisme de partage de ressources critiques n'est à prévoir. En revanche, des temps de réponse longs peuvent se produire.
  - Une tâche préemptible peut être interrompue à n'importe quel instant et le processeur être affecté à une autre tâche. Dans ce cas, les temps de réponse à un événement externe peuvent être très courts ; mais nous avons alors une programmation plus complexe avec un besoin de mécanisme de partage de ressources critiques.
- Analyse de l'ordonnancement hors-ligne ou en ligne :
  - Une analyse de l'ordonnancement hors-ligne correspond à la construction d'une séquence d'exécution complète sur la base des paramètres temporels des tâches en utilisant une modélisation (réseaux de Petri…) ou une simulation (animation ou énumération du modèle). L'ordonnanceur nécessaire est simple puisque la séquence d'exécution est prédéfinie, il se réduit à un séquenceur. En revanche, l'application ainsi figée est peu flexible.
  - Une analyse de l'ordonnancement en ligne correspond à un choix dynamique de la prochaine tâche à exécuter en fonction des paramètres de la tâche. Si le système a des contraintes temporelles, on doit utiliser préalablement à la mise en exploitation du système des tests permettant de vérifier qu'en toutes circonstances les contraintes temporelles seront respectées. On appelle ces tests des **tests d'ordonnançabilité**.
- Exécution synchrone ou asynchrone :
  - Une exécution est dite synchrone si les tâches sont non préemtibles et s'exécutent les unes après les autres dans un ordre qui peut être défini par une analyse hors-ligne de l'ordonnancement.
  - Une exécution est dite asynchrone si les tâches sont préemtibles et s'exécutent selon l'ordonnancement. Une analyse de la séquence doit se faire obligatoirement en ligne.

Dans la suite de cet ouvrage, nous nous intéressons plus particulièrement aux systèmes asynchrones composés de tâches préemptibles avec un ordonnancement en ligne. Ainsi l'architecture logicielle de l'application est composée de plusieurs

tâches réalisées par le concepteur et d'un environnement spécifique, le noyau temps réel, que nous allons décrire. Le point central de cet environnement est l'ordonnanceur qui permet d'affecter à tout instant le processeur à une tâche afin de respecter l'ensemble des contraintes temporelles attachées à la gestion du procédé.

1.2.1.3 Exécutif ou noyau temps réel

Cet environnement particulier d'exécution, exécutif ou noyau temps réel, peut être assimilé à un système d'exploitation de petite taille dédié aux applications de contrôle-commande. La caractéristique fondamentale est son déterminisme d'exécution avec des paramètres temporels fixés (temps de prise en compte d'une interruption, changement de contexte entre deux tâches, etc.). Nous pouvons comparer les différences au niveau des objectifs fixés pour le noyau d'exécution d'un système informatique classique et d'un système informatique de contrôle-commande.

Un système classique n'a pas été conçu pour permettre de respecter des contraintes temporelles, mais il suit les règles suivantes :

- Politiques d'ordonnancement des activités basées sur le partage équitable du processeur : affectation identique du temps processeur à tous les processus en cours ;
- Gestion non optimisée des interruptions ;
- Mécanismes de gestion mémoire (cache…) et de micro-exécution engendrant des fluctuations temporelles (difficulté pour déterminer précisément les durées des tâches) ;
- Gestion des temporisateurs ou de l'horloge pas assez fine (plusieurs ms) ;
- Concurrence de l'application temps réel avec le système d'exploitation toujours actif ;
- Gestion globale basée sur l'optimisation d'utilisation des ressources et du temps de réponse moyen des différents processus en cours.

Un système informatique de contrôle-commande s'attache aux caractéristiques suivantes :

- Efficacité de l'algorithme d'ordonnancement avec une complexité limitée ;
- Respect des contraintes de temps (échéances…). Ces contraintes temporelles se traduisent plus en termes de choix d'une activité à exécuter à un instant donné plutôt que de rapidité d'exécution de toutes les activités ;
- Prédictibilité (répétitivité des exécutions dans des contextes identiques) ;
- Capacité à supporter les surcharges ;
- Possibilité de certification pour les applications de certains domaines comme l'avionique, l'automobile…

En général, les contraintes temporelles ne peuvent pas être garanties dans un système d'exploitation généraliste (Unix, Windows 8…) contrairement à un noyau temps réel.

- Une application temps réel étant par définition un système multitâche, le rôle essentiel du noyau temps réel est donc de gérer l'enchaînement et la concurrence des tâches en optimisant l'occupation de l'unité centrale du système informatique. Les principales fonctions d'un noyau temps réel peuvent être scindées en trois groupes :
  1. gestion des entrées/sorties (gestion des interruptions, gestion des interfaces d'entrées/sorties, gestion des réseaux de communications…) ;
  2. ordonnancement des tâches (orchestration du fonctionnement normal, surveillance, changements de mode, traitement des surcharges…) ;
  3. relations entre les tâches (synchronisation, communication, accès à une ressource critique en exclusion mutuelle, gestion du temps…).

Il est important de noter que les tâches sont les unités actives du système ; le noyau temps réel n'est actif que lors de son appel. Une tâche activée peut appeler le noyau temps réel par une **requête**. Les différentes requêtes sont servies par des modules du noyau temps réel appelées **primitives**. Ensuite le noyau temps réel réactive une tâche de l'application selon l'algorithme d'ordonnancement utilisé (voir figure 1.9).

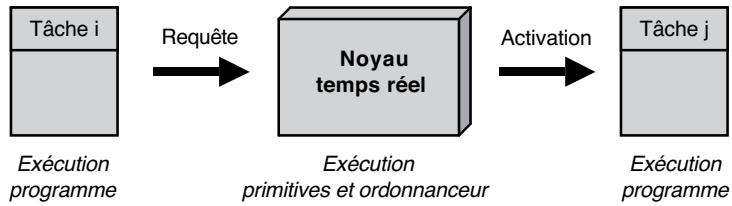

Figure 1.9 Interaction entre les tâches et le noyau temps réel.

Ainsi le noyau temps réel centralise toutes les demandes d'activation des tâches et gère des tables lui permettant de comparer les priorités (ou les urgences) et l'état de ces diverses tâches, ainsi que l'état d'occupation des ressources. La décision d'activation d'une tâche étant prise, le noyau temps réel lance les modules de programmes correspondant à cette tâche et lui alloue les ressources disponibles. La tâche activée occupe un processeur jusqu'à la fin de son exécution sous le respect des conditions suivantes :

- Elle ne réalise pas d'opérations d'entrées-sorties ;
- Les ressources utilisées sont disponibles ;
- Aucun événement extérieur ne revendique le déroulement d'une tâche plus prioritaire.

Nous pouvons donc décrire schématiquement le contexte complet d'exécution d'une application temps réel avec les deux parties : tâches et noyau temps réel (voir figure 1.10).

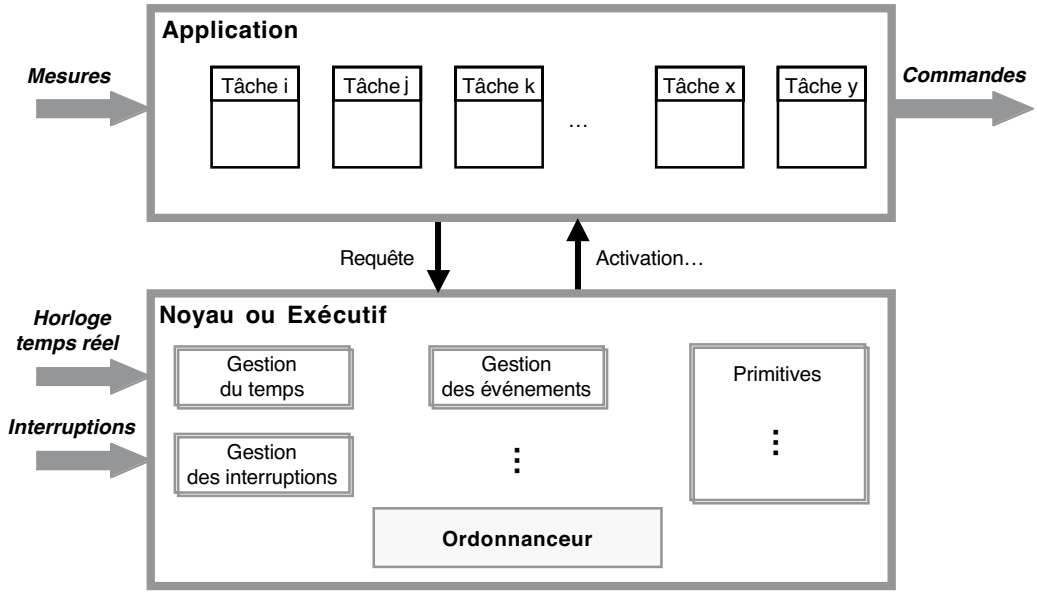

Figure 1.10 Architecture de l'application : tâches et noyau temps réel.

En conclusion de cette section sur l'ordonnancement qui est étudié de façon plus complète dans le chapitre 8, l'ordonnancement dans le cas des systèmes temps réel à contraintes temporelles strictes a pour objectif principal de répondre aux deux cas suivants :

- Fautes temporelles : cela correspond à un non-respect d'une contrainte temporelle associée à une tâche comme le dépassement de la date limite d'exécution ou échéance. Cela induit la notion d'**urgence** d'une tâche.
- Surcharge : lors de l'occurrence d'une ou plusieurs fautes temporelles, l'ordonnanceur peut réagir en supprimant une ou plusieurs tâches de l'application, ce qui amène à la notion d'**importance**, c'est-à-dire le choix d'une tâche à exécuter par rapport aux spécifications fonctionnelles de l'application.

1.2.1.4 Implémentation des applications de contrôle-commande

Comme nous le verrons au cours de cet ouvrage, les langages de développement des applications de contrôle-commande sont très divers. Cependant, par rapport à l'environnement d'exécution que nous venons de décrire (noyau temps réel avec les trois fonctions décrites : 1—gestion des interruptions, 2—ordonnancement,

3—relations entre les tâches), il possible de décliner les langages en trois groupes (voir figure 1.11) :

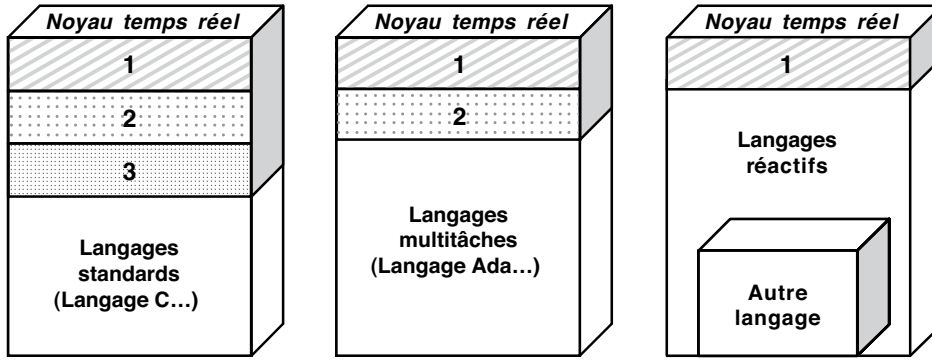

Figure 1.11 Langages utilisés pour développer les applications avec un noyau temps réel (1 : gestion des interruptions, 2 : ordonnancement, 3 : relations entre les tâches).

- Langages standards (langage C…) : le noyau temps réel qui supporte ce type de langage doit être complet puisque le langage n'intègre aucune spécificité multitâche.
- Langages multitâches (langage Ada, Java…) : ces langages permettent de décrire l'application en termes de tâches ; ainsi le noyau peut être plus réduit et ne comporter que les deux premières fonctions.
- Langages réactifs (langages Lustre, Esterel, Signal…) : ces langages donnent non seulement la possibilité de décrire les fonctionnalités du programme, mais aussi l'enchaînement des différentes parties. Le noyau est donc limité à une couche proche du matériel lié notamment à la gestion des interruptions. En revanche, étant donnée la possibilité très limitée d'expression de l'aspect fonctionnel, ils sont souvent associés à un langage standard pour pallier ce manque.

1.2.2 Architecture matérielle des applications de contrôle-commande

Comme nous l'avons vu en introduction, l'aspect matériel a une très grande importance dans les applications de contrôle-commande. Cette implication est liée d'une part à la connexion directe avec le monde physique réel à l'aide d'une grande diversité de systèmes d'entrées/sorties et d'autre part au matériel informatique parfois spécifique et développé pour une application donnée. Ce dernier point concerne les applications dites dédiées et embarquées ; le matériel a été conçu et créé spécifiquement pour une application, comme un téléphone portable, une caméra vidéo, etc.