LA METHODE DARTS ET LA PROGRAMMATION MULTITACHE EN LABVIEW

NGO Khanh Hieu, GROLLEAU Emmanuel

Laboratoire d'Informatique Scientifique et Industrielle
ENSMA, Téléport 2 - 1, av. Clément Ader, BP 40109, 86961 Futuroscope
ngokhanhhieu2001@yahoo.co.uk, grolleau@ensma.fr

Résumé: Les systèmes temps réel sont de plus en plus présents dans la vie quotidienne. Ces systèmes de contrôle/commande sont soumis à des contraintes temporelles inhérentes à l'environnement. La méthode DARTS (*Design Approach For Real-Time System*) [Gom93] est une méthode de conception particulièrement bien adaptée à la conception de systèmes temps réel. LabVIEW est un langage de programmation graphique basé sur le formalisme flot de données ayant des fonctionnalités avancées, propres à la synchronisation et la communication. L'objectif de cet article est de présenter la bibliothèque DARTSVIEW, qui est une palette de fonctions LabVIEW orientée conception DARTS. L'utilisation de cette palette permettra à un concepteur d'application temps réel, à partir de DARTS, d'obtenir directement le modèle d'implémentation.

Mots clés: LabVIEW, mutiltâche, concurrence, synchronisation, communication, conception de systèmes temps réel.

1 CONTEXTE DE L'ETUDE

La "concurrence" est l'un des problèmes fréquemment posés quand on parle de systèmes temps réel. Dans un tel système, plusieurs activités, dont chacune est présentée par une tâche indépendante, peuvent s'exécuter en parallèle (ainsi, des tâches peuvent être chargées de l'acquisition de données à différentes périodes, d'autres tâches peuvent être dédiées au calcul, et d'autres à l'application de commandes sur des cartes d'acquisition). Lorsque ces activités se synchronisent et communiquent, le respect de l'exclusion mutuelle, de la synchronisation et de la communication est vraiment un point clé auquel il convient de s'adresser.

- l'exclusion mutuelle consiste à garantir un aspect exclusif à une donnée ou une ressource partagée par plusieurs tâches,
- la synchronisation permet de bloquer une tâche tant qu'une autre ne la réveille pas,
- la communication permet à des tâches d'échanger des données.

Le but de cet article est de présenter une bibliothèque facilitant l'implémentation de tâches partageant des ressources, communiquant et se synchronisant.

Le développement d'un système temps réel nécessite généralement différentes phases: la spécification, la conception, l'implémentation, les tests unitaires (tests fonctionnels de chaque tâche), les tests d'intégration, puis la validation temporelle. Dans la suite de cet article, nous présentons la méthode DARTS, un outil pouvant être utilisé lors de la phase de conception, puis sa traduction en LabVIEW, choisi comme langage de développement. pour créer une

bibliothèque, nommée DARTSVIEW, facilitant le passage de la conception DARTS à l'implémentation LabVIEW et permettant de simplifier l'implémentation de la concurrence.

2 PRESENTATION DE LA METHODE DARTS

Proposée par H. Gomaa, DARTS est une méthode de conception fournissant des outils qui permet de décrire les interactions de tâches concurrentes en terme d'interfaces entre elles. Ces interfaces dépendent du fonctionnement du système, et peuvent être exprimées sous la forme de messages, événements ou modules de données (généralement, les modules de données hébergent des données ou des programmes dont on doit garantir l'accès exclusif) (voir Fig. 1).

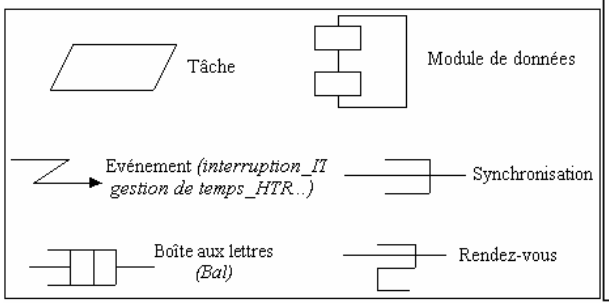

Figure 1: Eléments d'un diagramme DARTS

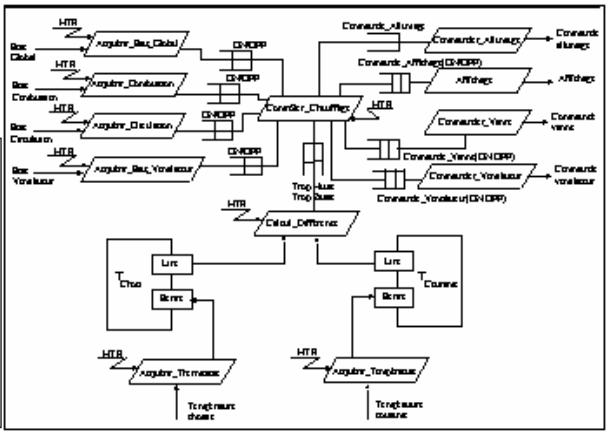

Figure 2: Un diagramme DARTS

Chaque tâche est présentée par un parallélogramme. Une tâche peut être soit une tâche matérielle, soit une tâche logicielle. L'activation des tâches matérielles est réalisée par un événement (une interruption interne ou externe provenant du procédé, pouvant être une horloge). Les tâches logicielles sont activées par des synchronisations, boîtes aux lettres, ou rendez-vous par le biais d'une autre tâche, elle-même matérielle ou logicielle.

La synchronisation est la communication sans données entre deux tâches, appelées "productrice" et "comsommatrice". Dans le sens le plus large, la tâche "consommatrice" ne peut s'exécuter que lorsqu'elle est déclenchée (i.e. la synchronisation a lieu) par la tâche "productrice". Les outils logiciels permettant d'implémenter la synchronisation peuvent être le Sémaphore et l'Evénement.

La communication est le passage d'information d'une tâche à une autre [Gev 98]. Elle peut de deux types: la communication synchrone et la communication asynchrone. La communication synchrone, représentée par le rendez-vous, est un mécanisme dans lequel les deux tâches qui communiquent le font de façon synchrone (on peut faire l'analogie avec le téléphone, où les deux interlocuteurs doivent être prêts à communiquer pour que la communication s'opère. Le langage Ada 83 était basé sur ce paradigme de communication.). Au contraire, la communication asynchrone, implémentée généralement par une boîte à lettre (BAL), est un mécanisme dans lequel l'émission et la réception des messages sont effectuées de manière asynchrone. Généralement, les messages sont stockés dans une file de type FIFO (premier arrivé premier sorti). Un blocage survient respectivement pour l'émettrice et la réceptrice lorsque la BAL est pleine ou lorsqu'il n'y a pas de message dans BAL. Cependant, il est possible d'implémenter des BAL à écrasement : dans ce cas, lorsque la BAL est pleine, l'émettrice n'est pas bloquée mais écrase le message le plus ancien.

Le module de données (MDD) est utilisé comme un système d'encapsulation et de protection de données. Le principe du module est de permettre à des tâches de partager des informations et d'y accéder en les protégeant contre des accès simultanés. Les données encapsulées ne

peuvent être accédées qu'à travers l'interface. Le MDD doit garantir l'atomicité des opérations de cette interface les unes par rapport aux autres (ie. une opération ne peut pas interrompre une opération en cours sur le même MDD). Le MDD peut être vu comme une BAL de taille 1, à écrasement, non bloquante en lecture (i.e. les messages ne sont pas consommés par leur lecture).

La figure 2 nous présente le diagramme DARTS dans la phase de conception d'une étude de cas: un système de chauffage d'une maison.

3 LabVIEW, UN LANGAGE MULTITACHE

LabVIEW est un langage naturellement parallèle: lorsque deux fonctions sont indépendantes, le "runtime" les exécute en parallèle en entrelaçant si nécessaire leur exécution. Cependant, la différence entre la notion de parallélisme dans LabVIEW et la sémantique flots de donnée associée au langage G ne permet pas de les faire communiquer directement à l'aide d'un flot de donnée (la seconde fonction serait forcée d'attendre la terminaison de la première pour commencer son exécution). LabVIEW nous offre cependant des outils avancés pour répondre au problème de synchronisation et de communication. Ces outils sont le Sémaphore, la Boîte à lettre, le Rendez-vous et l'Evénement. Ils se trouvent dans la sous-palette "Synchronisation" de la bibliothèque des fonctions. Les fonctions primordiales de ces outils sont présentées ci-dessous (le langage offre bien entendu de nombreuses variantes d'utilisation des outils, mais la description exhaustive de ces fonctions n'est pas l'objet de cet article). Il est à noter que dans LabVIEW, il existe d'autres outils permettant d'implémenter des communications. Parmi de ceux-ci, on peut citer la variable globale, ainsi que la combinaison registre à décalage/vi non ré-entrant.

Le Sémaphore est un concept introduit par Dijkstra (1965) pour protéger une section critique contre des accès simultanés. Les primitives de gestion de sémaphore dans LabVIEW sont "Créer_Sémaphore (σ, n)", "Prendre_Sémaphore P(σ)", "Rendre_Sémaphore V(σ)" (σ: nom de sémaphore, n: nombre d'accès simultanés autorisés). Lorsqu'une ressource est protégée par un sémaphore σ, si une tâche veut y accéder, elle doit d'abord obtenir une permission d'accès en appelant la primitive P(σ). Et elle va rendre cette permission lors de l'achèvement de sa section critique par la primitive V(σ).

La boîte aux lettres (BAL) permet la communication asynchrone entre tâches. C'est une zone d'échange dans laquelle une tâche dite "émettrice" dépose des données et une tâche dite "réceptrice" retire ces données dans l'ordre d'arrivée. Les primitives de gestion de BAL sont "Créer_BAL (β, t)", "Déposer_BAL (β, δ)", "Retirer_BAL (β, δ)" (β: nom de BAL, t: nombre de messages maximal contenus dans BAL, δ: message).

Le rendez-vous dans LabVIEW est simplement un objet permettant à plusieurs tâches de s'attendre en un point donné dans leur exécution. Les primitives de rendez-vous sont "Créer_rendez-vous", "Attente_sur_rendez-vous", "Supprimer_rendez-vous".

L'événement permet une communication par signaux. Contrairement à la communication par BAL, il n'y a pas d'échange de données lorsque des tâches communiquent par événement. Mais, à certains moments de la vie de l'application, des tâches peuvent émettre ou attendre des événements. Les primitives de gestion de l'événement sont "Créer_événement", "Attendre_ événement", "Signaler_événement".

3.1 Implémentation des concepts DARTS par LabVIEW

En utilisant des fonctionnalités avancées offertes par LabVIEW, il est relativement aisé d'implémenter les concepts de la méthode DARTS. Cependant, la représentation graphique du

langage nous a permis de créer un modèle de programmation DARTS proche de cette méthode de boxologie.
Dans la méthode DARTS, les tâches s'exécutent généralement en boucle sans fin (boucle « Tant que vrai »), chaque itération étant déclenchée soit sur un événement comme une interruption matérielle, une horloge périodique (utilisation du vi « Attendre le prochain multiple de »), l'arrivée d'un message dans une BAL ou bien le relâchement d'une synchronisation.
Le problème de synchronisation peut être résolu à l'aide d'un Sémaphore zéro (un sémaphore ayant la valeur initiale des accès simultanés nulle). Le principe, illustré sur la figure 3, est le suivant: au début, il n'y a pas d'instance libre du sémaphore, et la tâche en attente de synchronisation se met en attente du sémaphore. Elle est donc endormie. La tâche souhaitant relâcher la synchronisation va simplement libérer une instance du sémaphore. La tâche en attente se réveille alors en prenant l'instance du sémaphore et peut exécuter le reste de son traitement. Elle se remet à l'itération suivante en attente du sémaphore, qui ne sera disponible que lorsque la tâche déclenchant la synchronisation le souhaitera. Ainsi, la contrainte de précédence engendrée par la synchronisation DARTS est satisfaite. Sur la figure 3, la tâche appelante est une tâche matérielle périodique et la tâche appelée est une tâche logicielle.

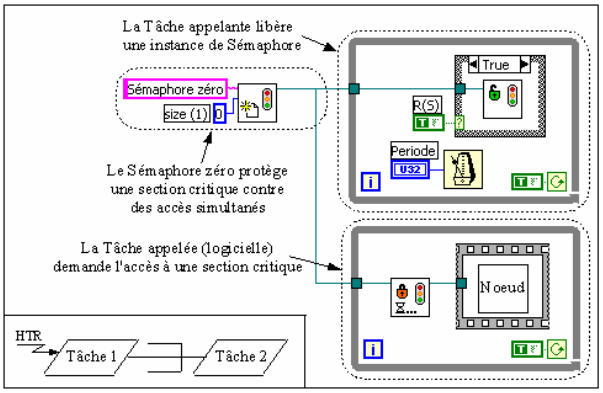

Figure 3: La synchronisation par sémaphore dans LabVIEW

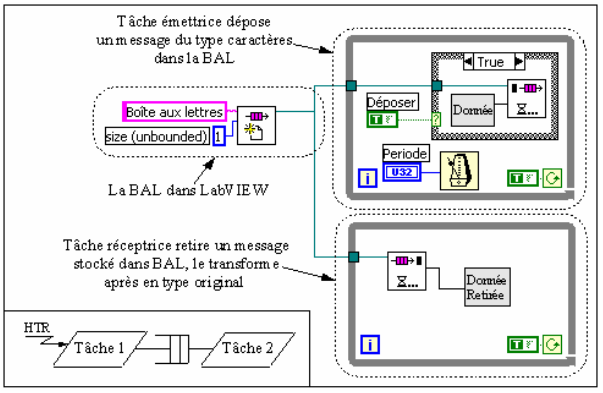

Figure 4: La communication par boîte à lettre dans LabVIEW

Le problème de la communication asynchrone peut être résolu à l'aide de l'outil "boîte aux lettres" LabVIEW. A chaque itération (voir figure 4), la tâche "réceptrice", qui utilise la primitive "Retirer_BAL", attendra un message envoyé par la tâche "émettrice", qui utilise la primitive "Déposer_BAL". L'émettrice n'a pas besoin d'attendre que la réceptrice soit à l'écoute pour lui envoyer des messages : l'émission de messages est asynchrone et la réceptrice est une tâche logicielle activée par BAL. Remarquer que dans LabVIEW, avant la version 6.1, pour envoyer ou récupérer un message par BAL, il faut d'abord le convertir en type caractères grâce à la fonction "Flatten to String" et le transformer après au type original grâce à la fonction "Unflatten from String". La version 6.1 a introduit des BAL polymorphes rendant cette manipulation inutile.

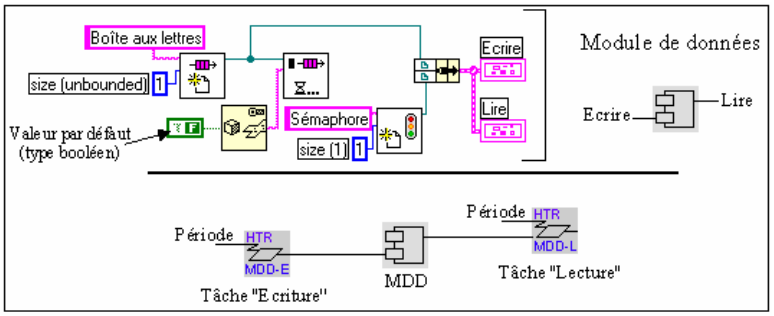

Figure 5: Le vi de module de donnée dans LabVIEW

Dans un diagramme DARTS, le module de données se sert d'un système d'encapsulation de l'information avec des opérations permettant d'accéder aux données stockées. Le MDD pourrait être implémenté à l'aide d'un vi non réentrant avec registre à décalage, mais cette implémentation ne permet pas l'encapsulation des vi à la DARTS. Le choix que nous avons fait propose une l'implémentation suivante: lors de l'écriture, on vide d'abord la BAL grâce à l'instruction "Flush Queue", puis on insère le nouveau message (c'est l'écrasement); lors de la lecture, on lit le message stocké dans la BAL sans l'enlever; le sémaphore est utilisé conjointement pour garantir qu'une opération sur le module ne peut en interrompre une autre (notamment entre le moment où l'écriture vide la BAL et le moment où elle insère le message). En fait, le vi de MDD est une combinaison d'une boîte à lettre de taille 1 et d'un sémaphore de même taille. Un connecteur de deux terminaux qui correspondent à deux opérations "Lire" et "Ecrire" est construit dans ce vi pour faire la liaison entre le MDD et les deux VIs de tâches "Lecture", "Ecriture" (voir Fig. 5).

4 LA BIBLIOTHEQUE "DARTSVIEW"

La bibliothèque DARTSVIEW est une palette de fonctions orientée conception DARTS. Cette palette propose des vi qui encapsulent les concepts présentés en section 3 dans des sous-vi ayant l'aspect des éléments DARTS. En utilisant cette palette, un concepteur d'application temps réel peut très simplement, à partir de la conception DARTS, obtenir directement le squelette opérationnel des tâches ainsi que les outils de communication/synchronisation à utiliser (voir Fig. 6).

La philosophie de DARTSVIEW est assez proche de celle de GRAFCETVIEW [Gev 98] : la sémantique flot de données n'est pas orientée dans le même sens que le flot de communications : en effet, si l'on considère les vi Contrôle_Chauffage (1), Commande_Allumage (2), Commander_Allumage (3), le flot de communication va de (1) vers (2) vers (3) alors que le flot de données part de (2) (création d'un élément de synchronisation émettant sur ses sorties une référence vers l'élément créé) et va vers (1) et (3).

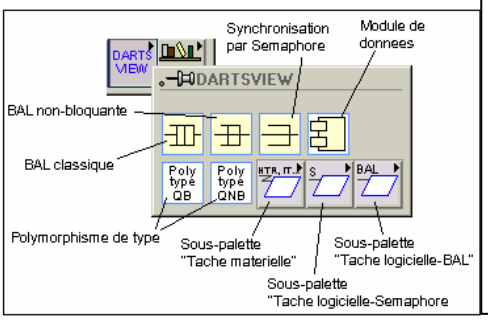

Figure 6: La palette DARTSVIEW

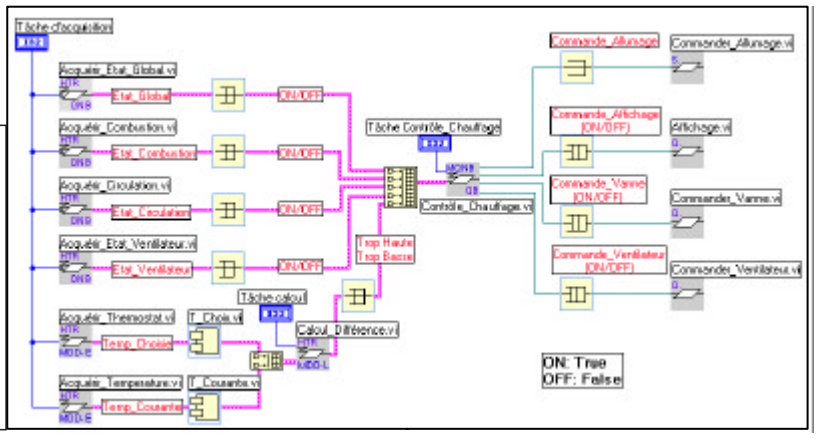

Figure 7: Diagramme DARTSVIEW

La figure 7 est l'implémentation du diagramme DARTS donné sur la figure 2. La palette DARTSVIEW propose donc des éléments de programmation "préfabriqués" implémentant les concepts multitâche DARTS.

5 CONCLUSION

Les avantages d'une telle approche sont les suivants:

- Utilisation d'un langage graphique (LabVIEW) pour concevoir directement le système à partir d'une méthode graphique de conception (DARTS): une véritable analogie est

proposée au concepteur (voir Fig. 7). De plus, le concepteur dispose d'une vue globale de l'application.

- Les éléments de la palette DARTSVIEW disposés implémentent automatiquement les communications et synchronisations mises en œuvre entre les tâches. Il ne reste plus alors au concepteur qu'à implémenter le comportement fonctionnel des tâches, sans se soucier de la mise en œuvre des communications.
- Facilité des tests unitaires des tâches grâce à l'instrumentation de l'environnement de développement (point d'arrêt, animation de l'exécution, espionnage des valeurs...).
- Facilité de mise en œuvre des tests d'intégration grâce à l'utilisation d'une tâche de simulation du procédé.
- Simplicité d'utilisation des cartes d'acquisition.

Enfin, en conclusion, nous présentons les perspectives de notre démarche, qui font actuellement le sujet d'une thèse de doctorat. Les objectifs à long terme sont de fournir une plateforme d'aide au développement d'application temps réel basée sur LabVIEW. Le passage de la conception à l'implémentation est facilité grâce à la bibliothèque DARTSVIEW, qui contient des prototypes de tâches. Différentes pistes sont abordées, comme la génération de code source Ada à partir de DARTSVIEW, ou bien la validation temporelle basée sur une étude d'ordonnançabilité des tâches générées (après mesure par le concepteur des pires durées d'exécution des tâches), ainsi que la simulation du fonctionnement du système piloté par un ordonnanceur.

6 BIBLIOGRAPHIE

[Bar 96] J. Barnes, "Programming in Ada 95", Addison-Wesley, Workingham, England, 702p., 1996.

[BC 98] J. P. Babau, F. Cottet, "Programmation et Validation des applications temps réel à contraintes strictes sur l'analyse schedulability analysis", Paris, 11-13 Janvier, 1995.

[CDKM] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri, "Ordonnancement temps réel", Hermes, Paris, 2000.

[CG 02] F. Cottet, E. Grolleau, "Développement des systèmes informatiques temps réel", Cours informatique Temps Réel à l'ENSMA.

[Cot 01] F. Cottet, "LabVIEW, Programmation et Applications", Dunod, 2001.

[Gev 98] E. Geveaux, "Conception d'un environnement de développement des applications de contrôle de procédé basé sur le modèle formel GRAFCET et fondé sur un langage graphique flot de données", rapport de Thèse, LISI – ENSMA, 29 Septembre 1998.

[Gol 93] S. Goldsmith, "A pratical guide to real-time systems development", Prentice Hall, 503p., 1993.

[Gom 93] H. Gomaa, "Software Design Methods for Concurrent and Real-Time System", Addison Wesley, SEI Series in Software Engineering, 1993.

[LL 73] C. L. Liu, J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, vol. 20, n°1, pp. 46-61, 1973.

[TE 97] J. P. Thomesse, J. P. Elloy, "Ecole d'été temps réel: applications, réseaux et systèmes", actes de l'école d'été temps réel, ERT'97, ENSMA, Futuroscope, pp. 2-3, 22-26 Septembre 1997.