Transformations de spécifications incluant du contrôle en spécification flot de données pour implantation distribuée

Nicolas Pernet, Yves Sorel

Projet AOSTE
tel : 01 39 63 55 80
fax : 01 39 63 51 93
INRIA Rocquencourt
78153 Le Chesnay, FRANCE
Email : Nicolas.Pernet, Yves.Sorel@inria.fr

RÉSUMÉ. *Parce qu'un systèmes temps réel combine fonctionnalités de contrôle et traitement de données, il est souvent spécifié à l'aide de plusieurs langages adaptés à ces deux aspects. La plupart de ces systèmes étant aujourd'hui distribués le problème est ensuite d'obtenir une implantation de ces spécifications distinctes. En effet une distribution de ces spécifications par production séparée de code ne permet pas d'obtenir une implantation cohérente. Nous proposons donc d'unifier toutes les spécifications en une seule. Cette unification conduit à un graphe flot de données conditionné qui explicite le parallélisme potentiel nécessaire à une exploitation efficace des ressources distribuées. Enfin on utilise le logiciel SynDEx pour obtenir automatiquement une implantation distribuée et cohérente à partir de la spécification obtenue.*

ABSTRACT. *Because a real-time system combines control and data processing designers specify it using different languages. Such systems are often distributed and the problem is to obtain a distributed implementation from these distinct specifications. Indeed, the method based on separated code generation and manual distribution leads to incoherent implementation. We propose to unify all these specifications into a unique one. The resulting specification is a conditioned data flow graph which exhibits the potential parallelism necessary to an efficient use of distributed resources. Finally, we use the SynDEx software in order to automatically produce a distributed and coherent implementation from the resulting specification.*

MOTS-CLÉS : *spécification, flot de contrôle, flot de données, implantation distribuée*

KEYWORDS: *specification, control flow, data flow, distributed implementation*

1. Introduction

Nos travaux de recherche sont dédiés aux systèmes temps réel distribués. Nous appelons "système" l'implantation de fonctionnalités, ou algorithmes, sur une architecture matérielle. De tels systèmes sont avant tout réactifs (Harel *et al.*, 1985) dans le sens où ils interagissent continuellement avec leur environnement en acquérant des signaux d'entrée, en effectuant des calculs sur ces signaux et en produisant des signaux de sortie. Les systèmes temps réel sont, quant à eux, des systèmes réactifs pour lesquels des contraintes temporelles sont imposées entre deux occurrences successives d'un signal d'entrée (période) ou entre un signal d'entrée et un signal de sortie (latence). Les systèmes temps réels nous intéressant, sont souvent distribués et leurs algorithmes consistent, à la fois, en du contrôle et du traitement de données. De tels algorithmes sont habituellement décrits à l'aide de langages de spécification graphiques. L'électronique embarquée que l'on rencontre aujourd'hui dans l'automobile est un exemple de système temps réel distribué sur des processeurs communiquant par bus afin d'exécuter des algorithmes de contrôle et/ou de traitement de données.

Cet article présente les caractéristiques des algorithmes de contrôle et de traitement de données et montre comment ces caractéristiques influent sur le choix des langages de spécification. En effet, la partie contrôle et la partie traitement de données conduisent généralement à utiliser des langages de spécification de types différents. Nous verrons que l'approche habituelle concernant la spécification de tels algorithmes, et consistant en la combinaison de plusieurs langages, pose problème lors d'une implantation sur architecture distribuée. Comme solution à ce problème, nous présentons d'une part un modèle de graphe flot de données conditionné qui permet de spécifier le contrôle, et d'autre part une transformation de ce graphe qui permet d'en obtenir une forme exploitable pour l'implantation distribuée. Ensuite nous proposons une alternative à l'approche usuelle utilisant des passerelles afin de transformer les spécifications incluant du contrôle en un graphe flot de données conditionné. Puis nous présentons le logiciel SynDEx qui permet d'obtenir une implantation temps réel distribuée à partir de ce graphe et d'une spécification d'architecture matérielle.

2. Les systèmes de contrôle et de traitement de données

2.1. *Aspects contrôle et aspects traitement de données*

Les fonctionnalités des systèmes temps réel embarqués combinent contrôle et traitement de données. Les fonctionnalités de traitement de données consistent en du calcul sur des données, alors que le contrôle consiste à séquencer ces calculs en choisissant lequel effectuer lorsqu'il y a plusieurs alternatives. Il est important de noter que le contrôle n'implique pas la notion d'état. En effet les choix effectués par le contrôle peuvent avoir lieu sans connaissance des calculs antérieurs. Le cas échéant c'est l'obligation de mémoriser des résultats de calculs précédents qui introduit la notion d'état.

2.2. *Langages de spécification*

Les langages de spécification utilisés pour spécifier les systèmes de contrôle et de traitement de données sous forme graphique sont de deux types.

Dans les langages flot de contrôle un sommet est un état et un arc décrit une transition entre deux états, c'est à dire la condition nécessaire pour passer de l'un à l'autre. Les calculs peuvent être déclenchés par une transition ou un état.

Dans les langages flot de données un sommet est une opération de calcul et un arc décrit une dépendance de données entre deux opérations. La façon dont le contrôle y est représenté diffère d'un langage flot de données à l'autre. Si la plupart utilise un sommet "retard" pour stocker l'état, certains utilisent une entrée booléenne sur une opération afin de choisir de l'exécuter ou pas. D'autres permettent de spécifier différents sous-graphes pour une même opération, une entrée spéciale permettant alors de choisir, à chaque occurrence de l'opération, le sous-graphe qui lui sera substitué.

Il est important de noter que dans le flot de contrôle, les arcs définissent un ordre total entre les états et donc les calculs qui leurs sont associés. D'autre part les états et les données utilisés par les calculs sont des variables globales, accessibles par l'ensemble des états et des transitions du graphe. Or ce concept de variable globale est très difficile à maintenir sur une architecture à mémoire distribuée. Enfin les calculs n'y sont pas détaillés sous forme de graphe.

Au contraire dans un graphe flot de données, les arcs ne définissent qu'un ordre partiel entre les opérations. Tous les échanges de données entre opérations sont explicités, y compris les données servant au contrôle. Enfin le contrôle y est représenté comme les calculs sous forme de graphe.

Lorsqu'une implantation distribuée est souhaitée, il est donc préférable de partir d'une spécification flot de données :

– l'ordre partiel des graphes flot de données fait apparaître du parallélisme potentiel. En effet, deux opérations sans dépendance de données entre elles pourront s'exécuter en même temps sur des processeurs différents si l'architecture matérielle le permet ;

– les dépendances de données étant exprimées, les problèmes d'accès concurrents aux données n'existent pas.

En effet bien qu'il soit possible de spécifier du parallélisme avec des langages flot de contrôle, l'absence de la spécification explicite des dépendances de données empêche l'exploitation directe de ce parallélisme. Sur la figure 1 page suivante, se trouve à gauche un programme Esterel (Boussinot *et al.*, 1991) et à droite son équivalent sous forme de graphe SyncCharts (André, 2003), tous deux langages flot de contrôle. La structure || en Esterel à gauche et les pointillés en SyncCharts servent à spécifier du parallélisme entre deux parties de la spécification. Mais cela n'implique pas forcément du parallélisme potentiel au sens où nous l'entendons. Si on regarde de plus près le comportement des deux parties, on s'aperçoit que la première produit une donnée

```
present A then
 emit B
end present;
||
present B then
 emit C
end present
```

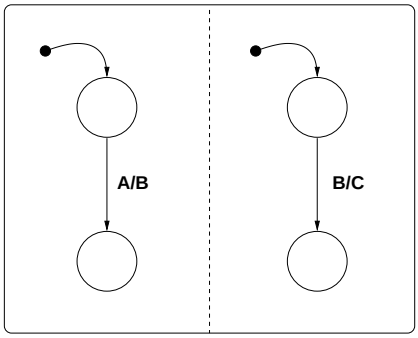

Figure 1. *Exemple de spécification de parallélisme en flot de contrôle*

attendue par la deuxième. En effet, la première attend A puis produit B alors que la deuxième attend B et produit C. Ainsi il existe implicitement une dépendance de données entre les deux parties rendant impossible l'exploitation du parallélisme spécifié. Ce dernier n'est que du parallélisme logique. Dans une spécification flot de contrôle du parallélisme potentiel ne peut être exhibé qu'après l'étude des dépendances de données existant implicitement à travers les variables utilisées.

2.3. *Spécification et implantation distribuée : l'approche usuelle*

La combinaison des fonctionnalités de contrôle et de traitement de données dans un même système conduisent leurs développeurs à utiliser plusieurs langages de spécification. Une fois les vérifications et simulations monoprocesseur effectuées se pose le problème de l'implantation distribuée. Il existe alors deux cas possibles.

Dans le premier les différents langages utilisés pour spécifier le système ne sont pas réunis sous un environnement unique. Dans ce cas chaque outil reposant sur un langage de spécification permet habituellement une génération de code monoprocesseur, et chaque spécification devient, après compilation, un code séquentiel que les développeurs doivent allouer à un des processeurs. Puisque certaines données sont communes ou échangées par les différentes spécifications, les développeurs doivent ensuite distribuer et ordonnancer les communications entre des processeurs. En plus des erreurs humaines dans l'ajout des communications, l'implantation à partir de codes obtenus séparément conduit généralement à un comportement d'ensemble différent de celui spécifié (Maffeïs, 1993).

Dans le second cas un environnement regroupe plusieurs langages de spécification. C'est le cas par exemple lorsque le langage flot de données Simulink est utilisé avec le langage flot de contrôle Stateflow[1]. On peut citer aussi Ptolemy (Liu *et al.*,

1. http ://www.mathworks.com

2003) un environnement de spécification hétérogène et l'environnement Scade[2] basé sur les langages synchrones Lustre et Esterel. Néanmoins, là encore, la plupart de ces outils n'offrent qu'une génération de code monoprocesseur, ou au mieux permettent à l'utilisateur de "découper" l'ensemble des spécifications en plusieurs parties, avant de générer les codes monoprocesseur correspondant à ce découpage. Si dans ce cas le comportement d'ensemble est plus cohérent il reste néanmoins le problème des communications.

De plus la distribution à la main des spécifications pose problème. Premièrement, dans les systèmes de contrôle et de traitement de données, les entrées sont des capteurs et les sorties des actionneurs. Or, avant même l'implantation, l'architecture des capteurs et des actionneurs est connue, les processeurs auxquels chacun d'entre eux est connecté aussi. Le choix de placer un des codes monoprocesseur obtenus sur un des processeurs est donc difficile, car il peut manipuler des entrées et des sorties dont les actionneurs et capteurs sont disséminés sur l'architecture. Deuxièmement, dans le cas de code obtenu à partir d'une spécification flot de contrôle, toutes les entrées ne sont pas nécessaires à chaque évaluation du graphe. En effet, en connaissant l'état courant, on peut réduire l'ensemble des entrées nécessaires à celle utiles pour évaluer les transitions sortantes par exemple. Un code monoprocesseur centralisant le contrôle implique donc un surplus de communications inutiles provenant des différents capteurs.

Il apparaît donc qu'un système nécessite plusieurs langages pour être spécifié. Puisque une distribution de ces spécifications par production séparée de codes monoprocesseur ne permet pas d'obtenir une implantation distribuée cohérente nous proposons d'unifier toutes les spécifications du système en une seule à l'aide de transformations. Comme les langages flot de données sont les seuls à expliciter directement le parallélisme potentiel cette unification des spécification doit se faire dans un langage flot de données. La spécification flot de données obtenue peut ensuite être exploitée pour obtenir une implantation distribuée.

Pour cela, nous présentons d'abord un modèle de graphe flot de données conditionné qui permet d'exprimer le contrôle et nous détaillons les transformations nécessaires pour l'implantation distribuée de ces graphes conditionnés. Nous verrons ensuite comment des passerelles permettent d'obtenir une spécification flot de données conditionnée unique à partir de spécifications incluant de contrôle et comment SynDEx permet d'obtenir une implantation distribuée à partir du graphe flot de données conditionné obtenu.

L'implantation distribuée à partir d'une spécification unique a fait l'objet d'un certain nombre de travaux. Dans (Maffeïs *et al.,* 1994) SynDEx est utilisé pour distribuer des programmes SIGNAL utilisant le modèle flot de données. Dans (Caspi *et al.,* 1999) un programme séquentiel est dupliqué sur les processeurs de l'architecture distribuée et chacun en exécute une partie. Les auteurs présentent des algorithmes permettant de reconstruire les dépendances de données afin de rajouter automatiquement

2. http ://www.esterel-technologies.com

les primitives d'envoi et de réceptions de données. Ces ajouts se font après distribution, ce qui signifie que les communications, et notamment les durées de celles-ci, ne sont pas prises en compte lors de la distribution et de l'ordonnancement. Dans (Caspi *et al.,* 2003) la spécification du système se fait sous Simulink qui permet la simulation, puis le graphe est transformé en graphe Scade (Lustre) pour validation, enfin le graphe est implanté sur une architecture distribuée de type TTA (Time-Triggered Architecture). Toute la chaîne est automatisée, les auteurs présentant une transformation Simulink/Scade, ainsi qu'une extension de Scade pour générer du code temps réel distribué. Cette chaîne est la plus proche de ce que nous présentons ici. Ce que nous apportons en plus est l'exploitation du parallélisme potentiel dans des graphes flot de contrôle. Car aussi bien dans Simulink que dans Scade la partie flot de contrôle est d'abord transformée en une opération atomique (fonction C le plus souvent) puis incluse dans la partie flot de données. Ainsi le parallélisme potentiel qu'elle pouvait contenir devient inexploitable.

3. Modèle flot de données conditionné

Le système que l'on cherche à spécifier étant réactif, un graphe flot de données est dans notre cas un sous-graphe infiniment répété, chaque répétition correspondant à un instant logique des langages synchrones (Benveniste *et al.,* 1991). Les sommets du graphe sont des opérations qui doivent être exécutées avant que ne débute la prochaine répétition infinie du graphe. On parle ici de répétition infinie par rapport à répétition finie où une opération est répétée plusieurs fois comme dans une boucle. Nous préférons le terme répétition au terme itération car cette répétition n'est pas nécessairement temporelle (séquentielle) mais peut être aussi spatiale (parallèle). Ces deux aspects peuvent être exploités conjointement si nécessaire. Chaque opération possède des ports d'entrée et/ou des ports de sortie et chaque arc, appelé aussi dépendance, relie un port de sortie à un ou plusieurs (diffusion) port d'entrée. Chaque opération ne peut être exécutée que lorsque toutes ses entrées sont disponibles. Un sommet spécifique appelé *delay* est nécessaire si une opération a besoin d'une donnée produite lors d'une précédente répétition infinie du graphe. On étend aussi le modèle flot de données de manière à rendre possible la spécification hiérarchique, une opération peut donc être décrite par un sous-graphe d'opérations.

Pour permettre de spécifier le contrôle dans le cas d'implantation distribuée nous étendons la notion de *conditionnement* déjà présent dans le modèle flot de données de (Dennis, 1974). Nous détaillons ici le conditionnement et aussi les transformations à apporter à ce graphe flot de données conditionné pour l'implanter sur une architecture distribuée.

Une opération est *conditionnée* lorsque son exécution dépend de la valeur d'une de ses données d'entrée appelée *donnée de conditionnement*. Un graphe flot de données comportant au moins une opération conditionnée est appelé graphe flot de données conditionné. Lorsque des opérations conditionnées sont conditionnées par la même donnée et s'exécutent pour la même valeur de cette donnée, elles peuvent être spé-

cifiées sous la forme d'un sous-graphe décrivant une opération *conditionnante*. Tous les sous-graphes d'une opération conditionnante sont constitués d'opération conditionnées par la même donnée, la valeur de cette donnée pour laquelle les opérations s'exécutent diffère d'un sous-graphe à l'autre. En plus de la donnée d'entrée correspondant à la donnée de conditionnement, une opération conditionnante possède des entrées et des sorties qui peuvent être partagées par les opérations des différents sous-graphes. Toujours grâce à la hiérarchie, une opération conditionnée peut être à son tour une opération conditionnante.

Le graphe de la figure 2 montre un exemple de graphe flot de données conditionné. Il est constitué de 5 opérations non-conditionnantes (A, B, D, E et F) et d'une opération conditionnante C. La donnée de conditionnement de C est la sortie de l'opération A :

– **si sortie(A)=1 :** l'exécution de C est équivalente à l'exécution de l'opération conditionnée C_{11} ;

– **si sortie(A)=2 :** l'exécution de C est équivalente à l'exécution de l'opération conditionnée C_{21} suivie de celle de l'opération conditionnée C_{22}.

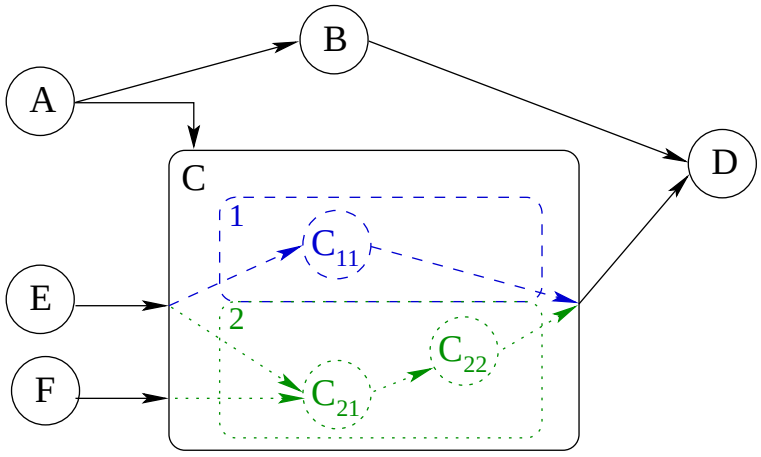

Figure 2. *Graphe conditionné*

3.1. *Opération conditionnée*

Avant l'implantation distribuée on effectue une transformation du graphe d'algorithme appelée *la mise à plat*. Dans le graphe mis à plat, chaque opération conditionnante est remplacée par les opérations conditionnées de ses sous-graphes. La *condition* d'une opération conditionnée est un booléen correspondant au test "est-ce que la donnée de conditionnement est égale à la valeur spécifiée pour le sous-graphe auquel elle appartient". Une opération conditionnée est exécutée lorsque sa condition est vraie.

Deux opérations conditionnées du graphe mis à plat sont dites en *exclusion mutuelle* si leurs conditions respectives ne peuvent être vérifiées simultanément. On parle alors de *conditions exclusives*. Cela signifie que lors de l'exécution temps réel l'une d'entre elle seulement sera exécutée. Lors de la distribution et de l'ordonnancement cette propriété est intéressante car deux opérations en exclusion mutuelle peuvent alors être ordonnancées sur le même processeur en même temps. C'est le cas par exemple ici pour C_{11} et C_{21}. Par l'utilisation de plusieurs niveaux de hiérarchie, une opération peut-être conditionnée par plusieurs données de conditionnement. Ainsi pour être plus exact, la condition d'une opération est une liste de conditions. Par exemple, une opération C appartenant à un des sous-graphes de l'opération conditionnée B, elle-même appartenant à un des sous-graphes de l'opération conditionnée A, est conditionnée par la donnée de conditionnement de A et par celle de B.

De plus cette liste de conditions est ordonnée. On parle d'imbrication de conditions et non d'ensemble de conditions. En effet à l'exécution les conditions doivent être testées dans l'ordre hiérarchique du graphe d'algorithme. Si on reprend l'exemple de l'imbrication précédente A, B, C, lors de l'exécution on testera d'abord l'entrée de conditionnement de A (la condition pour laquelle B a lieu) puis l'entrée de conditionnement de B (la condition pour laquelle C a lieu). Ceci permet d'effectuer des regroupements, sous un même test, d'opérations conditionnées par la même condition. Il existe des techniques pour optimiser le regroupement de ces tests comme l'emploi de DDD (Couvreur *et al.*, 2002).

3.2. *Dépendance conditionnée et de conditionnement*

Si l'une des opérations connectées par une dépendance est conditionnée, on parle de *dépendance conditionnée*. Lors de l'implantation certaines dépendances impliquent des communications quand l'opération productrice et l'opération consommatrice sont implantées sur deux processeurs distincts. Si la dépendance correspondante est conditionnée, on parle de *communications conditionnées*. La condition d'une communication est la condition de l'opération consommatrice de la donnée communiquée. Par exemple, la condition d'une communication pour la dépendance $A \rightarrow B$ est la condition de B.

Une dépendance conditionnée est :

– soit entrante (d'un prédécesseur du sous-graphe conditionné vers une des opérations du sous-graphe comme c'est le cas pour la dépendance entre E et C_{11} sur la figure 2 page précédente),

– soit interne au sous-graphe conditionné (et relie donc deux opérations conditionnées par la même condition comme c'est le cas pour la dépendance entre C_{21} et C_{22} sur la figure 2 page précédente),

– soit sortante (d'une des opérations de sortie du sous-graphe conditionné vers un des successeurs du sous-graphe comme c'est le cas pour la dépendance entre C_{11} et D sur la figure 2 page précédente).

Une *dépendance de conditionnement* est une dépendance d'une opération produisant une donnée vers une opération pour laquelle cette donnée sert de donnée de conditionnement.

3.3. *Ajout de dépendances et d'opérations pour implantation distribuée*

La principale difficulté concernant le conditionnement est de s'assurer que, sur les processeurs effectuant une opération conditionnée, la ou les données de conditionnement de cette opération sont bien présentes avant d'effectuer cette opération (pour pouvoir tester si l'opération doit être exécutée ou non). Pour résoudre ce problème, la mise à plat d'un graphe conditionné ajoute des opérations et des dépendances.

Dans le cas des opérations conditionnées et afin d'assurer que le processeur exécutant une telle opération disposera de l'ensemble des données de conditionnement par lesquelles cette opération est conditionnée, on ajoute une dépendance de conditionnement pour chaque opération conditionnée d'une opération conditionnante. Il sera ainsi possible de tester la valeur de ces données pour savoir si l'opération doit être effectuée ou non. En effet lors de l'ordonnancement de cette opération conditionnée sur un processeur donné, les dépendances de conditionnement impliqueront des communications de la donnée de conditionnement vers ce processeur.

Dans le cas des dépendances conditionnées entrantes, on ajoute des opérations *CondI*. Elles permettent d'assurer que le processeur source d'une communication conditionnée entrante disposera de l'ensemble des données de conditionnement de cette communication et pourra donc tester si la communication doit avoir lieu ou non. Le processeur récepteur de la communication conditionnée dispose aussi des données de conditionnement car l'opération de calcul destinataire de cette communication, qui est ordonnancé sur ce processeur récepteur, requiert les mêmes données de conditionnement que cette communication. Le nombre d'opérations *CondI* rajoutées pour une opération de conditionnement est égal au nombre de prédécesseurs de cette opération, c'est à dire ayant une dépendance conditionnée entrante sur cette opération. Un exemple est donné dans la figure 3.

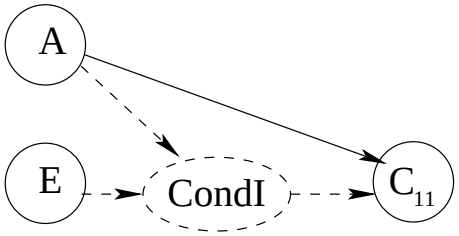

Figure 3. *Transformation de la dépendance entrante $E \rightarrow C_{11}$ de la figure 2 page 7*

Le cas des dépendances conditionnées internes n'entraîne aucun rajout d'opérations ni de dépendances. Les processeurs où sont ordonnancées l'opération productrice

et l'opération consommatrice disposeront obligatoirement des données de conditionnement nécessaires grâce aux dépendances de conditionnement qui ont été ajoutées à toutes les opérations conditionnées lorsque l'opération conditionnante a été remplacée par ses sous-graphes.

Dans le cas des dépendances conditionnées sortantes on ajoute lors de la mise à plat une opération *CondO* pour chaque sortie d'opération conditionnante. Pour chaque sortie d'une opération conditionnante, dans chaque sous-graphe de l'opération conditionnante, il existe une opération ayant une dépendance via cette sortie. Par exemple les opération conditionnées C_{22} et C_{11}, en exclusion mutuelle, ont une dépendance vers D via une sortie de C. Suivant la valeur de la donnée de conditionnement l'opération *CondO* choisit la dépendance provenant du sous-graphe correspondant et la transmet en sortie. Un exemple est donné dans la figure 4.

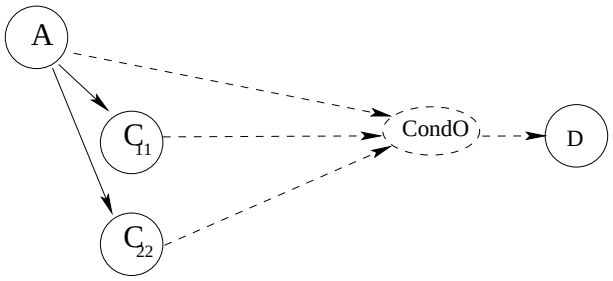

Figure 4. *Transformation pour les dépendances sortantes $C_{11} \rightarrow D$ et $C_{22} \rightarrow D$ utilisant la même sortie de C dans la figure 2 page 7*

La figure 5 page suivante montre le graphe mis à plat correspondant au graphe conditionné de la figure 2 page 7.

4. Implantation distribuée avec SynDEx de spécifications incluant du contrôle

SynDEx[3] est un logiciel libre de CAO niveau système qui repose sur la méthodologie Adéquation Algorithme Architecture (AAA) (Sorel, 1994). Elle est fondée sur des modèles de graphes, autant pour spécifier les algorithmes applicatifs et les architectures matérielles distribuée, que pour déduire les implantations possibles en termes de transformations de graphes. Le modèle de graphe utilisé est le modèle de graphe conditionné précédemment décrit. L'adéquation revient à résoudre un problème d'optimisation consistant à choisir une implantation dont les performances, déduites des caractéristiques des composants matériels, respectent les contraintes temps réel et d'embarquabilité. La première étape de l'adéquation correspond à la mise à plat du graphe d'algorithme conditionné décrite précédemment. Ensuite une heuristique qui tend à minimiser le temps d'exécution total de l'algorithme (Grandpierre *et al.*, 1999)

3. http ://www.syndex.org

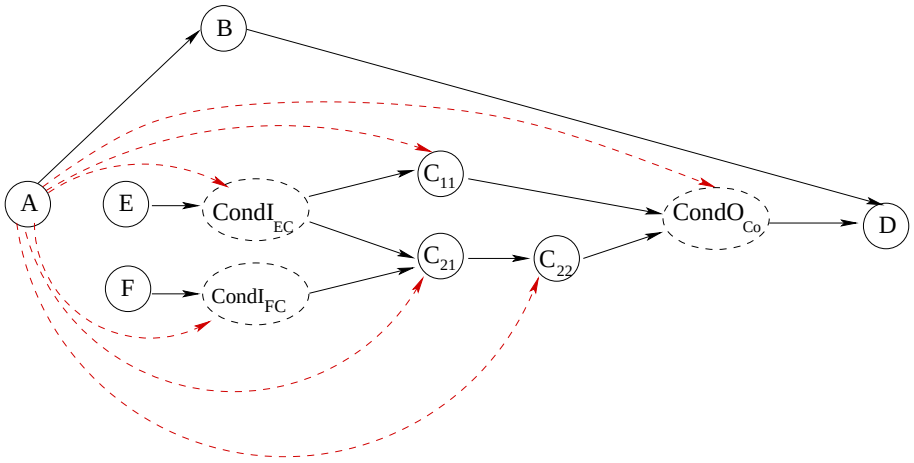

Figure 5. *Transformation du graphe conditionné de la figure 2 page 7*

est utilisée pour la distribution et l'ordonnancement. Cette heuristique est une heuristique gloutonne d'ordonnancement par liste (Yang *et al.*, 1993) prenant en compte l'ordonnancement et le routage des communications.

Afin d'utiliser SynDEx pour obtenir une implantation distribuée tout en gardant les langages de spécification habituels, nous proposons des passerelles permettant la transformation automatique de spécifications incluant du contrôle en graphe flot de données conditionné SynDEx. Nous présentons ici deux de ces passerelles.

4.1. *Passerelle SyncCharts/SynDEx*

SyncCharts (André, 2003) est un langage flot de contrôle, proche du langage Statechart (Harel, 1987) de David Harel, ayant l'avantage d'être totalement déterministe. Ce langage permet la description hiérarchique, un état pouvant contenir un sous-graphe, et la composition pour spécifier le parallélisme logique.

La figure 6 page suivante présente un graphe SyncCharts. Le graphe ABRO comporte un état ABO qui est décrit par un graphe comportant deux états dont un, AB, est décrit par la composition de deux graphes possédant chacun deux états. Son comportement est le suivant. Le système attend une occurrence de A et de B. Lorsque celles-ci ont eu lieu, O est émit. Toute occurrence de R ré-initialise le système.

La transformation de graphes SyncCharts en graphes SynDEx a déjà été présenté dans (Pernet *et al.*, 2003). La figure 7 page suivante montre le graphe flot de données conditionné SynDEx obtenu par transformation automatique du graphe SyncCharts ABRO. Les quatre sommets "delay" (reconnaissables au croisement de leurs dépendances de données) correspondent aux états de chacun des quatre graphes (ABRO,

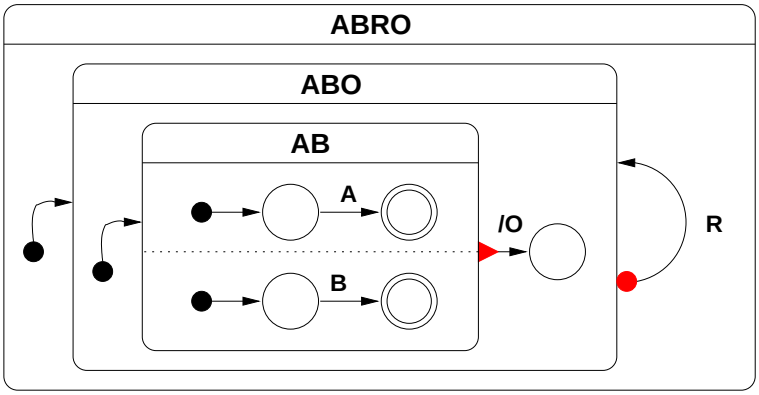

Figure 6. *Exemple de graphe SyncCharts : ABRO*

ABO, A et B), quatre opérations conditionnantes décrivent, pour chaque graphe, les transitions concernant chacun des états du graphe. Enfin les deux sommets de droite spécifient l'émission de O.

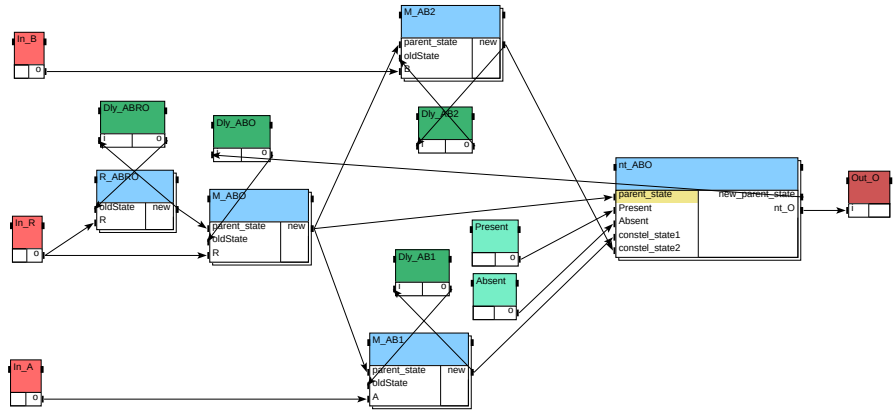

Figure 7. *ABRO après transformation en graphe SynDEx*

4.2. *Passerelle Scicos/SynDEx*

Scicos[4] (Nikoukhah *et al.*, 1996) est une boîte à outil de Scilab qui permet la spécification sous forme de flot contrôle et de données, puis d'en faire la simulation. Ce

4. http ://www.scicos.org

langage est une alternative à Simulink. Dans les graphes flot de données Scicos, le contrôle est spécifié à l'aide de port d'entrée permettant, via une donnée booléenne, d'activer ou pas l'opération. Cette forme de contrôle est différente de notre approche dans SynDEx mais on peut, avec des opérations conditionnantes, représenter sous forme d'un graphe flot de données conditionné l'équivalent d'un graphe Scicos comportant une hiérarchie d'activations.

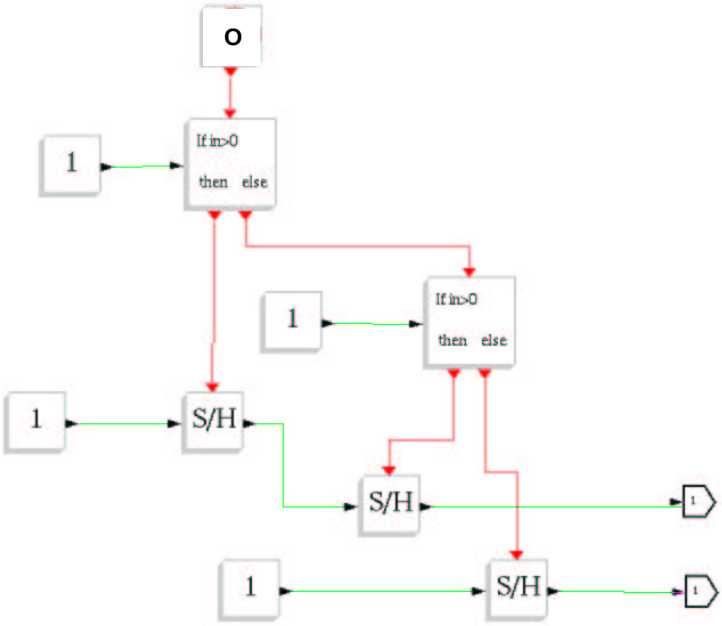

Figure 8. *Spécification Scicos*

La figure 8 montre une spécification Scicos comportant des sommets à entrées d'activation et des sommets manipulant et générant des données booléennes servant à ces activations. Sa transformation en graphe flot de données conditionné SynDEx, est illustré dans 4.3.

4.3. *Implantation distribuée avec les passerelles*

Afin de pouvoir faire coopérer plusieurs spécifications, il faut combiner les différents graphes flot de données conditionnés obtenus à l'aide des passerelles en un seul. Si les spécifications échangeaient des données, des dépendances sont à rajouter entre les composantes non connexes du graphe. C'est le cas par exemple si la spécification Scicos de la figure 8 utilise une donnée d'entrée O qui est la sortie de la spécification SyncCharts ABRO. L'usage de SyncCharts couplé avec Scicos est une alternative

"logiciel libre" à l'utilisation de Stateflow dans Simulink et l'interfaçage avec SynDEx offre une fonctionnalité supplémentaire de génération de code distribuée, là où les outils Mathworks ne permettent que de la génération de code monoprocesseur.

La figure 9 page suivante montre le graphe flot de données conditionné SynDEx obtenu après la transformation de la spécification SyncCharts ABRO et de la spécification Scicos. Après avoir connecté la sortie O de la partie SyncCharts à l'entrée O de la partie Scicos, l'utilisateur peut grâce à SynDEx :

– spécifier une architecture,

– spécifier des contraintes de distribution inhérentes à la réalité physique de l'architecture,

– réaliser différentes adéquations avec différentes architectures en vue de minimiser les ressources (exploration d'implantation),

– générer des exécutables temps réel pour une des implantations obtenues par l'adéquation.

L'utilisateur n'a plus aucune décision à prendre sur l'ordonnancement et la distribution des communications, évitant ainsi toute possibilité d'interbloquage. De plus il est sûr que le parallélisme potentiel de ses spécifications a été exploité.

5. Conclusion

Nous avons vu combien obtenir une implantation distribuée posait problème dans le cas de système de contrôle et de traitement de données. En effet l'utilisation de plusieurs langages de spécification amène les développeurs à choisir eux-même les processeurs devant exécuter chaque code obtenu par génération de code monoprocesseur. Ensuite, les développeurs doivent distribuer et ordonnancer les communications entre ces processeurs, sans aucune garantie de la cohérence de l'ensemble. Cette approche est source d'erreur, donc d'un allongement de la durée de développement de tels systèmes. De plus elle ne permet d'obtenir, au mieux, qu'une implantation distribuée correcte, c'est à dire satisfaisant les spécifications, à défaut d'être efficace, c'est à dire minimisant les ressources (taux d'utilisation des média, nombre de processeur). Pour que l'implantation soit efficace il faut d'une part que l'ensemble des dépendances de données soient explicitées et d'autre part que les spécifications expriment tout le parallélisme potentiel des fonctionnalités du système. Cela ne peut être réalisé qu'à partir d'une spécification flot de données. Nous avons donc présenté un modèle graphe flot de données conditionné pouvant être celui utilisé pour cette unification de spécification. En effet nous avons vu qu'il permettait de décrire le contrôle et, une fois mis à plat, était exploitable pour obtenir une implantation distribuée tirant profit du parallélisme potentiel. Pour unifier les spécifications nous avons ensuite proposé des passerelles permettant de transformer des spécifications incluant du contrôle, SyncCharts et Scicos, en spécification SynDEx. Ce logiciel permet d'obtenir, à partir d'un graphe d'algorithme basé sur notre modèle et d'un graphe d'architecture, une distribution et un ordonnancement optimisé des fonctionnalités sur l'architectures. Enfin

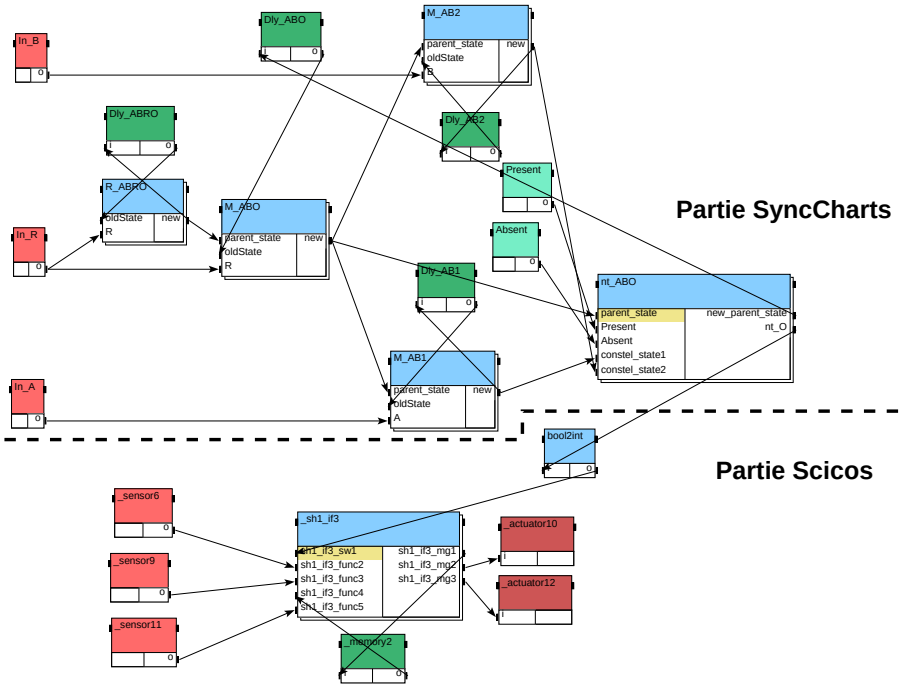

Figure 9. *Graphe SynDEx obtenu par transformation des graphes SyncCharts et Scicos*

l'emploi conjugué de SyncCharts, Scicos et SynDEx permet de profiter de la vérification sur les aspects contrôle (SyncCharts), de faire de la simulation sur la partie traitement de données (Scicos) et d'obtenir une implantation distribuée efficace de l'ensemble (SynDEx). Ceci raccourcit la durée de développement et minimise les ressources nécessaires.

6. Bibliographie

André C., « Computing SyncCharts reactions », *Synchronous Languages Applications and Programming*, Porto, Portugal, July, 2003.

Benveniste A., Berry G., « The Synchronous Approach to Reactive and Real-Time Systems », *Proceedings of the IEEE*, vol. 79, n° 9, p. 1270-1282, September, 1991.

Boussinot F., De Simone R., « The Esterel language », *Proceedings of the IEEE*, p. 1293-1304, September, 1991.

Caspi P., Curic A., Maignan A., Sofronis C., Tripakis S., Niebert P., « From Simulink to SCADE/Lustre to TTA : A layered approach for distributed embedded applications », *Proceedings of the LCTES 2003*, San Diego, Ca, US, June, 2003.

Caspi P., Girault A., Pilaud D., « Automatic Distribution of Reactive Systems for Asynchronous Networks of Processors », *IEEE Trans. on Software Engineering*, vol. 25, n° 3, p. 416-427, 1999.

Couvreur J., Encrenaz E., Paviot-Adet E., Poitrenaud D., Wacrenier P., « Data Decision Diagram for Petri Net Analysis », *Proceedings of the 23rd International Conference, ICATPN 2002, Lecture Notes in Computer Science*, Adelaide, Australia, June, 2002.

Dennis J., « First Version of a Dataflow Procedure Language », *Lecture Notes in Computer Sci.*, vol. 19, Springer-Verlag, p. 362-376, 1974.

Grandpierre T., Lavarenne C., Sorel Y., « Optimized Rapid Prototyping For Real-Time Embedded Heterogeneous Multiprocessors », *CODES'99 7th International Workshop on Hardware/Software Co-Design*, Rome, May, 1999.

Harel D., « Statecharts : a visual formalism for complex systems », *Science of Computer Programming*, vol. 8, p. 231-274, 1987.

Harel D., Amir P., « On the Development of Reactive Systems », *in* , K. R. Apt (ed.), *Logics and Models of Concurrent Systems*, Springer Verlag, New York, 1985.

Liu X., Liu J., Eker J., Lee E. A., « Heterogeneous Modeling and Design of Control Systems », *Software-Enabled Control : Information Technology for Dynamical Systems*, Tariq Samad and Gary Balas edn, Wiley-IEEE Press, April, 2003.

Maffeïs O., Ordonnancements de graphes de flots synchrones ; Application à la mise en œuvre de SIGNAL, PhD thesis, Université of Rennes, 1993.

Maffeïs O., Le Guernic P., « Distributed implementation of SIGNAL : Scheduling & graph clustering », *FTRTFT*, p. 547-566, 1994.

Nikoukhah R., Steer S., « SCICOS-a dynamic system builder and simulator », *Proceedings of the 1996 IEEE International Symposium on Computer-Aided Control System Design*, September, 1996.

Pernet N., Sorel Y., « Optimized implementation of distributed real-time embedded systems mixing control and data processing », *Proceedings of the ISCA 16th International Conference : Computer Applications in Industry and Engineering (CAINE-2003)*, Las Vegas, Nv, US, November, 2003.

Sorel Y., « Massively Parallel Systems with Real Time Constraints, the Algorithm Architecture Adequation Methodology », *Proceedings of Conf. on Massively Parallel Computing Systems*, Ischia, Italy, May, 1994.

Yang T., Gerasoulis A., « List Scheduling With and Without Communication Delays », *Parallel Computing*, vol. 19, n° 12, p. 1321-1344, 1993.