Optimisation de l'extensibilité lors du déploiement des systèmes temps-réel distribués

Asma Mehiaoui, Sara Tucci-Pergiovanni
CEA LIST DILS
Point Courrier n 174
91191 Gif-sur-Yvette Cedex, France
asma.mehiaoui@cea.fr,sara.tucci@cea.fr

Jean-Philippe Babau
LISyC, UBO, UEB
20 avenue Le Gorgeu
29200 Brest, France
jean-philippe.babau@univ-brest.fr

Résumé

Dans le processus de conception des systèmes temps-réel, un modèle de déploiement doit être synthétisé à partir d'une architecture logique et d'une plateforme d'exécution. Durant cette synthèse, les fonctions/signaux sont tout d'abord placés sur les processeurs/-bus de communication et sont ensuite regroupés dans des tâches OS/messages. Enfin, des priorités sont affectées aux tâches OS/messages. Plusieurs choix de conception sont possibles et des contraintes dures telles que (1) les contraintes temporelles, (2) les contraintes de capacité et (3) les contraintes d'allocation doivent être prises en compte, le problème de déploiement devient alors NP-difficile. Dans ce travail nous proposons une extension d'une méthode de placement et d'affectation de priorités pour des fonctions activées selon le modèle Event-triggered. Cette méthode est fondée sur la programmation linéaire et tend à optimiser deux objectifs : l'extensibilité et le temps de réponse du système. Un cas d'étude montre les résultats de comparaison entre les modèles d'activation Time-triggered et Event-triggered.
MOTS-CLES :systèmes temps-réel distribués ; optimisation ; conception à base de modèles ; analyse d'ordonnançabilité ; exploration architecturale.

In the design process of real-time systems, a deployment model must be synthesized from a logical architecture and an execution platform. During this synthesis, the functions/-signals are first placed on processors/communication bus and they are then grouped into OS tasks/messages. Finally, priorities are assigned to OS tasks/messages. Many design choices are possible and hard constraints must be considered such as : (1)timing constraints and (2)capacity constraints, the problem of deployment becomes then NP-hard. In this work, we propose an extension of a placement and priorities assignment method to functions activated according to Event-triggered model. This method is based on linear programming and tends to optimize two objectives : extensibility and system's response time. A case study shows the results of the comparison between Time-Triggered and Event-triggered activation models.
KEYWORDS : distributed real-time systems ; optimization ; model-based design ; scheduling analysis ; architectural exploration.

1 Introduction

Les systèmes temps-réel sont présents dans plusieurs domaines tels que l'avionique, l'automobile, le spatial, le multimédia sur le web, etc.

Le développement logiciel de tels systèmes reste à la fois complexe et coûteux. La complexité est due à la criticité qui consiste à respecter un ensemble de contraintes dures fortement liées, afin d'assurer le bon fonctionnement du système. Ces dernières années ont montré que l'ingénierie dirigée par les modèles (IDM) [4] représente un domaine prometteur pour d'une part la réduction de la complexité, en raisonnant sur un niveau d'abstraction élevé qui est le modèle, et d'autre part la diminution du coût en validant le modèle d'implémentation durant la phase de conception. Cependant, les méthodologies à base de modèles préconisent la séparation

entre *le modèle fonctionnel* et *le modèle d'architecture*. Le modèle fonctionnel décrit le comportement logique du système en terme de fonctions et leur interaction via des signaux. Le modèle d'architecture représente la plateforme ciblée pour l'exécution du système. Ensuite, un modèle de déploiement est obtenu en projetant le modèle fonctionnel sur le modèle d'architecture et en configurant les différentes ressources du modèle d'architecture. La définition du modèle de déploiement nécessite le placement des fonctions/signaux sur les processeurs/bus de communication et leur regroupement dans des tâches/messages. De plus, une affectation de priorités aux tâches/messages est indispensable pour la validation par analyse d'ordonnançabilité, qui est l'une des techniques utilisées pour la détection précoce des problèmes de conception. L'ordonnançabilité du système représente une contrainte fondamentale pour la phase de déploiement, cela consiste à ce que le temps de réponse à un événement externe ne dépasse pas l'échéance de bout-en-bout, autres contraintes existent : les contraintes d'allocation qui sont données en entrée durant la spécification du système, et les contraintes de capacité qu'impose la plateforme d'exécution.

Dans ce travail, nous proposons une extension d'une méthode de placement fondée sur la programmation linéaire[9]. Dans cette extension nous considérons un modèle d'activation Event-triggered. Etant donné que l'optimisation joue un rôle important dans le déploiement des systèmes temps-réel, nous traitons deux critères d'optimisation : *la latence* et *l'extensibilité*. La latence d'une chaine de bout-en-bout est égale au temps de réponse de la dernière tâche exécutée dans la chaine. La métrique de l'extensibilité permet de mesurer les possibles futurs changements dans les budgets de temps des tâches et pour lesquels les ressources de la plateforme d'exécution ne sont pas dépassées.

Le reste du papier est organisé comme suit : la section suivante présente une vue d'ensemble sur le problème de déploiement. La section 3 est dédiée aux différentes techniques d'optimisation utilisées dans le cadre de la conception des systèmes temps-réel. La section 4, représente l'apport de ce papier dans l'aide au concepteur lors de la phase de déploiement. Dans la section 5, la méthode est appliquée à un cas d'étude et les résultats de comparaison des modèles Time-triggered (TT) et Event-triggered (ET) sont ainsi mentionnés. La dernière section conclut le papier en présentant une liste de perspectives.

2 Description du problème de déploiement

Le problème considéré dans ce travail consiste à générer un modèle de déploiement qui représente une combinaison du *modèle fonctionnel* et du *modèle d'architecture*. Cette combinaison réside dans la projection de la description logique du système (modèle fonctionnel) sur la plateforme d'exécution (modèle d'architecture) et aussi dans la configuration des ressources de ce dernier modèle.

Le modèle fonctionnel, décrivant le comportement du système, est représenté par un graphe acyclique orienté G. Ce graphe est composé d'un ensemble de transactions linéaires G=$\{\Gamma_1,\Gamma_2,...,\Gamma_p\}$. Les nœuds des transactions sont des opérations atomiques du système appelés fonctions et les liens sont les signaux échangés par les fonctions. Soit, $F = \{f_1,f_2,...,f_f\}$ l'ensemble des fonctions du système et $S = \{s_1,s_2,...,s_s\}$ l'ensemble des signaux. Une transaction Γ_i est déclenchée par un événement e_i qui peut être périodique ou sporadique avec respectivement une période d'activation ou un intervalle minimal entre deux activations P_i. De plus, chaque transaction a une durée maximale d'exécution D_i qui représente une contrainte du système. Une fonction f_i et un signal s_i ont un budget de temps ω_{fi} et ω_{si} qui déterminent respectivement la borne supérieure du temps d'exécution et du temps de transmission. Les fonctions/signaux peuvent être activés selon deux modèles : le modèle *Time-triggered* ou bien le modèle *Event-*

triggered. Le modèle Time-triggered est un modèle d'interaction purement périodique où toutes les fonctions en interaction sont activées périodiquement, de même pour les signaux. Dans le modèle Event-triggered les fonctions/signaux sont activés par des événements associés à l'arrivée des données d'entrée/du signal [7].

Le modèle d'architecture représente l'architecture matérielle et logicielle de la plateforme d'exécution du système. L'architecture matérielle est une topologie de la plateforme dédiée à l'exécution du système. Elle est représentée par un graphe G' constitué d'un ensemble de processeurs $C = \{c_1, c_2, \ldots, c_c\}$ connectés via des bus de communication $\beta = \{\beta_1, \beta_2, \ldots, \beta_b\}$. Les processeurs/bus de communication ont une capacité maximale (μ_c/μ_β) qui ne doit pas être dépassée, de plus, ils offrent un nombre maximal de ressources d'exécution/de communication appelées tâches/messages ($T = \{t_{c_1}, t_{c_2}, ..., t_{c_t}\}$ / $\Phi = \{m_{\beta_1}, m_{\beta_2}, ..., m_{\beta_m}\}$) et qui représentent les éléments de l'architecture logicielle. Une tâche t_{c_i}/un message m_{β_i} est l'entité permettant d'exécuter/de transmettre un ensemble de fonctions/signaux. Chaque tâche/message possède quatre paramètres : un budget de temps (ω), une priorité (π), une période d'activation (P) et un modèle d'activation. Les processeurs font tourner un système d'exploitation temps-réel dans le quel un ordonnanceur préemptif à base de priorités fixes est implémenté. Par ailleurs, un ordonnanceur non préemptif à base de priorités fixes est implémenté sur les bus de communication, ce qui correspond au bus CAN.

Le modèle de déploiement est une configuration dans laquelle les fonctions/signaux pour les tâches/messages sont définies, sachant qu'une tâche/un message vide n'est pas exécutable/transmissible. Suivant l'affectation donnée, les tâches/messages héritent : le budget de temps, la période et le modèle d'activation depuis les fonctions/signaux qui leur appartient. En outre, le paramètre de priorité pour les tâches/messages est ainsi.

3 Techniques d'optimisation pour le déploiement

Dans la littérature, plusieurs approches ont été proposées pour l'automatisation du déploiement des systèmes temps-réel. Dans ces travaux l'optimisation a été traitée selon une méthode d'optimisation qui est déterministe, stochastique ou heuristique.

Contrairement aux méthodes stochastiques et heuristiques pour lesquelles la qualité de la solution dépend des différents paramétrages aléatoires choisis pour l'exploration de l'espace de recherche et du choix de la solution de départ, certaines méthodes déterministes permettent de trouver la solution exacte en parcourant d'une manière exhaustive l'espace de recherche. La programmation linéaire est l'une des méthodes déterministes, bien qu'elle n'est pas adaptée à tous les problèmes d'optimisation, elle reste applicable pour le problème de déploiement optimisé des systèmes temps-réel. Dans un d'aide au concepteur, les contraintes et les critères d'optimisation sont souvent diversifiés par le concepteur. Par conséquence, la programmation linéaire est mieux adaptée au problème puisqu'elle permet très simplement d'ajouter de nouvelles contraintes et de modifier les critères d'optimisation, à l'inverse des méthodes stochastiques et heuristiques pour lesquelles une reformulation du problème est nécessaire.

Dans ce cadre, Davare et al [5] ont appliqué une procédure d'optimisation basée sur la programmation géométrique en nombres entiers afin d'affecter les périodes aux tâches et messages tout en respectant les échéances des transactions. Ultérieurement, Zheng et al [8] ont présenté une méthode d'optimisation basée sur la programmation linéaire en nombres entiers pour la synthèse des modèles d'activation des tâches et messages. Une contribution intéressante a été présentée dans [9] et [8], c'est une formulation linéaire pour le problème de placement des tâches/messages sur les processeurs/bus de communication et le problème d'affectation de priorités aux tâches/messages. Leur proposition est destinée au déploiement d'un système

temps-réel rapide et flexible. Toutefois, toutes les approches citées se basent sur l'hypothèse que le concepteur connait a priori le placement des fonctions/signaux ou le regroupement des fonctions/signaux dans les tâches/signaux. Aucune ne traite le problème de placement et le problème de regroupement en même temps. De plus, peu de travaux considèrent un modèle d'activation Event-triggered.

Dans ce travail, nous nous sommes basés sur les travaux [9] et [8] afin d'étendre la solution apportée pour le problème de placement et le problème d'affectation de priorités au modèle d'activation Event-triggered.

4 Méthode pour le déploiement

La génération du modèle de déploiement nécessite trois étapes : la première est le placement des fonctions/signaux sur les processeurs/bus de communication, la deuxième est le regroupement des fonctions/signaux dans des tâches/signaux, et enfin l'affectation de priorités aux tâches/messages. Cependant, durant cette génération, il est primordial de garantir le respect de certaines contraintes comme les : *contraintes de capacité*, *contraintes d'allocation* et *contraintes temporelles*.

- Les contraintes de capacité : la capacité maximale des processeurs/des bus de communication ne doit pas être dépassée.
- Les contraintes d'allocation : un signal est transmis sur un bus de communication si et seulement si la fonction émettrice et la fonction réceptrice de ce signal sont placées sur des processeurs différents. Une fonction peut aussi avoir une contrainte de localité. Par exemple une fonction qui est responsable de la collecte des données depuis un capteur doit être allouée sur le processeur lié au capteur.
- Les contraintes temporelles liées à l'exécution des transactions : consistent à vérifier le comportement temporel du système tel que toutes ses transactions doivent finir leur exécution avant l'écoulement de la durée maximale autorisée. La vérification des contraintes temporelles est faite à l'aide de l'analyse des temps de réponses.

Initialement, nous ne considérons pas l'étape de regroupement, ce qui signifie que chaque fonction/signal est vue comme une tâche/un message. Ensuite, nous étendons la solution dans [9] afin de supporter le placement des fonctions/signaux lorsque ceux-ci ne sont pas activés périodiquement. D'après [3] et [7] il existe des situations pour lesquelles l'Event-triggered est moins restrictif que le Time-triggered. La différence entre l'Event-triggered et le Time-triggered lors de la formulation du problème réside dans le calcul du temps de réponse des transactions, appelé latence L. La formulation suivante décrit le calcul de la latence pour le modèle d'activation Event-triggered. Cela représente le temps de réponse R de la dernière tâche activée dans la transaction :

1. $\forall \Gamma_i \in G : L_{\Gamma_i} \leq D_{\Gamma_i}$
2. $\forall \Gamma_i \in G, \forall t_i \in \Gamma_i : L_{\Gamma_i} \geq R_{t_i}$
3. $W_{t_i} = \omega_{t_i} + B_{t_i} + \sum_{t_j \in T/t_i} \theta_{t_i,t_j,c} * \omega_{t_j}$
4. $R_{t_i} = W_{t_i} + J_{t_i}$
5. $Y_{t_i,t_j} - M * \left(1 - \pi_{t_j,t_i}\right) \leq X_{t_i,t_j}$
6. $0 \leq X_{t_i,t_j} \leq Y_{t_i,t_j}$
7. $X_{t_i,t_j} \leq M * \pi_{t_j,t_i}$
8. $X_{t_i,t_j} - M * \left(1 - H_{t_i,t_j,c}\right) \leq \theta_{t_i,t_j,c}$
9. $0 \leq \theta_{t_i,t_j,c} \leq X_{t_i,t_j}$
10. $\theta_{t_i,t_j,c} \leq M * H_{t_i,t_j,c}$
11. $0 \leq Y_{t_i,t_j} - \left(\frac{W_{t_i}+J_{t_j}}{P_{t_j}}\right) < 1$
12. $J_{t_i} = \begin{cases} 0 \text{ si } t_i \text{ est activée par un événement externe} \\ R_{m_i} \text{ sinon,tel que } t_i \text{ est la tâche réceptrice de } m_i \end{cases}$

La variable booléenne $\pi_{t_j,t_i} = 1$ si t_j est plus prioritaire que t_i. $H_{t_i,t_j,c}$ est aussi une variable booléenne qui indique si t_i et t_j sont dans le même processeur. J_{t_i} représente le jitter d'activation de t_i. W_{t_i} est le temps de complétion de la tâche t_i, il est calculé en fonction des budgets de temps et des tâches plus prioritaires que t_i qui sont dans le même processeur. La formule 1 exprime les contraintes temporelles. La formule 2 décrit le calcul de la latence. Les contraintes 3, 4 et 12 permettent de calculer le temps de réponse des tâches selon l'analyse Holistic [6]. Le big M et les variables : X_{t_i,t_j}, Y_{t_i,t_j} et θ_{t_i,t_j} sont utilisés pour linéariser le calcul du temps de complétion, cette linéarisation est exprimée par les contraintes : 5, 6, 7, 8, 9, 10 et 11.

Le modèle de déploiement peut aussi subir certaines optimisations. Dans notre étude, nous considérons deux critères d'optimisation qui sont : la minimisation de la latence des transactions et la maximisation de *l'extensibilité des tâches*. L'extensibilité est définie comme la quantité maximale qui peut être ajoutée aux budgets de temps des tâches sans violer les contraintes de capacité. Le manque de place ne nous permet pas ici de détailler et fournir la formulation complète mise en œuvre.

5 Expérimentations

Afin d'illustrer notre étude comparative entre les modèles Event-triggered et Time-triggered par rapport au critère de l'extensibilité, nous considérons un système temps-réel distribué composé de 8 fonctions organisées en trois transactions. La plateforme d'exécution est composée de 3 processeurs et d'un bus de communication. Ce système est présenté dans [7]. L'objectif de cette étude est de montrer les cas où le modèle Time-triggered est restreint.

Résultats et discussion : la figure 1 indique la valeur de l'extensibilité par rapport aux budgets de temps des tâches pour les modèles Time-triggered et Event-triggered. Nous constatons que lorsque la charge des tâches de la même transaction est équilibrée, l'extensibilité reste identique pour les deux modèles ($facteur = 1$). Pour $facteur = 2$, $facteur = 3$ et $facteur = 4$ nous remarquons que plus le budget de temps de la tâche la moins prioritaire dans une transaction (dernière tâche activée) est supérieure aux budgets de temps des tâches plus prioritaires, meilleur est le gain de l'Event-triggered. Cela s'explique par le fait que le calcul de la latence pour le Time-triggered est trop pessimiste puisque celui-ci inclus la période d'activation [7]. Par conséquent, ce calcul empêche le programme linéaire de mettre les tâches de la même transaction dans des processeurs différents, ce qui limite la maximisation de l'extensibilité.

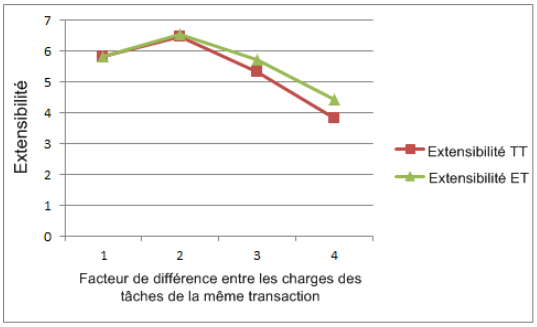

FIGURE 1 – Comparaison de l'extensibilité pour les modèles Time-triggered et Event-triggered

Il existe un autre inconvénient dans le modèle Time-triggered, dans le cas où la solution n'existe pas, contrairement à l'Event-triggered pour lequel une solution est fournie. Prenons par exemple des budgets de temps ordonnés par ordre décroissant (la tâche activée par l'événement externe, la plus prioritaire, a le plus grand budget de temps) d'un facteur de 3. Il n'existe pas de solution pour le Time-triggered car cette situation représente le cas où les tâches plus prioritaires de la transaction ont une charge plus importante que les tâches moins prioritaires. Dans ce cas, la la-

tence est trop pessimiste puisque des grandes valeurs sont considérés plusieurs fois dans le calcul des temps de réponses des tâches moins prioritaires, à la différence de l'Event-triggered qui a la possibilité de diminuer l'impact des tâches plus prioritaires sur l'exécution des tâches moins prioritaires en les plaçant sur des processeurs différents.

Pour information, nous utilisons le solveur de programmes linéaire GLPSOL [1] (un solveur non-commerciale) en sachant que les expérimentations sont réalisées sur une machine avec un processeur Intel dual-core i3-380m cadencé à 2.53GHz avec une mémoire de 7,68 Go. Ceci a comme contrainte de traiter que les systèmes de taille moyenne. Par contre l'utilisation d'un solveur commercial comme CPLEX [2] et d'une machine plus puissante permet de lever ces limites comme cela a été montré dans [9]. D'autres expérimentations ont été faites pour une application robotique et un sous système automobile (système régulateur de vitesse).

6 Conclusion et perspectives

La génération automatique d'un modèle de déploiement est très importante pour guider le concepteur vers une conception valide. Cela permet de réduire le temps de conception et de trouver une solution, si elle existe.

Nous présentons dans ce papier une première démarche apportant de l'aide au concepteur durant l'étape de placement. Cette démarche représente une extension pour les tâches déclenchées selon le modèle d'activation Event-triggered. En se basant sur un cas d'étude existant, nous montrons l'efficacité des résultats par rapport à l'optimisation de l'extensibilité et des latences de l'Event-triggered comparée au Time-triggered.

Actuellement, nous étendons la méthode pour prendre en considération la partie du regroupement, ceci permettant de réduire le nombre de tâches utilisées par le système. En ce qui concerne les travaux futurs, nous envisageons de considérer un système avec des modèles d'activation mixés Time-triggered/Event-triggered.

Références

[1] `http://catalogue.polytechnique.fr/site.php?id=122&fileid=1811`.

[2] Ilog cplex optimizer. `http://www.ilog.com/products/cplex/`.

[3] E. Azketa, L. Uribe, J.P. Gutiérrez, J.J., M. Marcos, and Almeida. Permutational genetic algorithm for the optimized mapping and scheduling of tasks and messages in distributed real-time systems.

[4] F. Huber J. Philipps B. Schätz, A. Pretschner. *Model based development of embedded systems.* Springer, 2002.

[5] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Proceedings of the 44th annual Design Automation Conference*, pages 278–283. ACM, 2007.

[6] J.C.P. Gutiérrez, J.J.G. García, and M.G. Harbour. On the schedulability analysis for distributed hard real-time systems. In *Proceedings of the 9th Euromicro Workshop on Real-Time Systems, Toledo, Spain.* Citeseer, 1997.

[7] M.D. Natale, W. Zheng, C. Pinello, P. Giusto, and AS Vincentelli. Optimizing end-to-end latencies by adaptation of the activation events in distributed automotive systems. In *RTAS'07. 13th IEEE.*

[8] W. Zheng, Q. Zhu, M. Di Natale, and A.S. Vincentelli. Definition of task allocation and priority assignment in hard real-time distributed systems. In *Real-Time Systems Symposium, 2007.*

[9] Q. Zhu, Y. Yang, E. Scholte, M. Di Natale, and A. Sangiovanni-Vincentelli. Optimizing extensibility in hard real-time distributed systems. In *RTAS 2009. 15th IEEE.* IEEE, 2009.