Trampoline : un support pour le développement d'applications temps réel

Trampoline

Jean-Luc Béchennec[*] — Mikaël Briday[**] — Sébastien Faucou[**] — Pierre Molinaro[***] — Yvon Trinquet[**]

Laboratoire IRCCyN (UMR CNRS)
1, rue de la Noë
44321 Nantes Cedex 3

[*] *CNRS*
[**] *IUT de Nantes - Université de Nantes*
[**] *École Centrale de Nantes*

prenom.nom@irccyn.ec-nantes.fr

RÉSUMÉ. *Cet article présente une chaîne de développement logiciel, pour les systèmes embarqués temps réel, s'appuyant sur le standard OSEK/VDX de l'automobile. L'article a pour objectif de montrer comment des développements effectués dans le contexte recherche ont pu être réutilisés pour l'enseignement en IUT GEII, suite à l'évolution du Programme Pédagogique National il y a quelques années. La suite d'outils permet à partir d'une conception initiale d'architecture logicielle sous forme de tâches et d'ISR de mettre en œuvre l'application temps réel à l'aide de seulement deux fichiers : l'un pour décrire l'architecture applicative par description des objets (tâches, ISRs, alarmes, événements ...) en langage standard OIL, l'autre pour décrire les actions algorithmiques des tâches et ISRs en langage C. Diverses implémentations de l'exécutif temps réel permettent de travailler sur cible microcontrôleur (C167, ARM7, MPC565, H12, AVR ...) ou bien sur cible Posix. L'ensemble de la chaîne disponible sous forme de logiciel libre.*

MOTS-CLÉS : *systèmes embarqués, temps réel, Osek*

1. Introduction

Un système informatique temps réel est un système pour lequel les résultats produits ne doivent pas seulement être corrects en valeur mais également en temps, c'est-à-dire que le système doit respecter les contraintes temporelles spécifiées dans le cahier des charges. Un tel système est toujours en couplage fort avec son environnement et les contraintes temporelles peuvent s'exprimer sous diverses formes : temps de réponse à une sollicitation de l'environnement (simple, de bout en bout), contraintes de débit à respecter (en entrée ou en sortie). Les applications sont considérables et vont du contrôle d'une centrale électrique à la gestion des protocoles dans un objet communicant, en passant par les calculateurs embarqués dans une automobile ou encore les calculateurs de bord d'un avion.

L'analyse du cahier des charges d'une application conduit au recensement des actions élémentaires que doit effectuer le système de pilotage et que l'on regroupe en « tâches ». Ainsi une tâche rassemble des fonctions de l'application qui sont exécutées dans les mêmes conditions événementielles. La spécification du comportement de l'ensemble des tâches coopérantes, sous forme d'une architecture logicielle, est naturelle et lisible si on l'exprime sous formes d'actions parallèles, mais ce parallélisme d'expression repose sur l'hypothèse implicite que chaque traitement est réalisé sur un processeur dédié, ce qui ne sera pas forcément vrai à l'exécution.

On trouve dans l'architecture des variables d'entrées/sorties représentant des éléments physiques du procédé accessibles via les capteurs/actionneurs (niveau mesuré, commande de vanne), des liens de coopération entre les tâches ou entre les tâches et l'environnement (liens d'activation, de synchronisation, de communication). En laissant de côté les techniques de co-design (conception conjointe du matériel et du logiciel) on peut considérer, que l'implémentation d'une application passe par le choix de la configuration matérielle sur laquelle seront implantées les tâches de l'application, ce qui peut conduire à un système centralisé monoprocesseur (ou multiprocesseur), ou bien à un système réparti. Ce choix est complexe et est guidé par de multiples contraintes.

En ce qui concerne l'implémentation de l'architecture logicielle celle-ci est également étroitement conditionnée par un choix majeur : celui de la réalisation du « lien entre les tâches et leurs événements activeurs » qui peut conduire à deux techniques différentes relevant de deux principes fondamentaux : l'approche « synchrone » et l'approche « asynchrone ». La démarche synchrone a été à l'origine du développement de langages et environnements de développement pour les systèmes temps réel, comme par exemple [BER 87]. Elle n'est pas utilisée dans ce papier. La technique de mise en œuvre asynchrone consiste à observer en permanence les occurrences d'événement, y compris au cours de l'exécution des tâchesCette démarche tient également compte des temps d'exécution des traitements. Sa mise en œuvre s'appuie sur l'utilisation d'un logiciel spécifique chargé de gérer l'exécution en pseudo parallélisme des tâches : l'exécutif temps réel ou système d'exploitation temps réel (RTOS).

L'équipe Systèmes Temps Réel (STR) de l'IRCCyN a une longue expérience de développement des systèmes d'exploitation temps réel, ce qui l'a amené à participer au travers de contrats industriels au développement de standard comme VDX (Vehicle Distributed Executive) pour les constructeurs français. Au niveau européen les industriels de l'automobile (constructeurs et équipementiers) ont conçu le standard OSEK/VDX dont l ?équipe STR a produit une version Open Source, Trampoline, pour la partie système d ?exploitation et le compilateur OIL. L'évolution des programmes du département GEII avec le nouveau PPN, notamment l'introduction explicite d'un module sur les « Systèmes Temps Réel » (MC-II2) nous a conduit à transférer un savoir-faire « recherche » vers la pédagogie. L'objet de cet article est de présenter quelques concepts liés au développement d'application temps réel et l'exploitation d'une chaîne de développement dans un enseignement au niveau bac+2. Il n'est bien entendu pas possible de développer les problématiques liées aux systèmes temps réel telles que : les langage de description d'architecture, les OS temps réel, l'ordonnancement temps réel, la modélisation et la vérification des applications etc. [TRI 99b], [TRI 99a], [STA 04]. L'article présente en section 2 quelques concepts d'OSEK/VDX et OIL. Ils sont mis en œuvre au travers d'un exemple dans la section 3, puis la section 4 donne des informations sur les implémentations et la dynamique de dissémination.

2. Concepts d'OSEK/VDX et OIL

Au delà de son rôle premier d'ordonnancer les exécutions des tâches, et de celui de protéger l'accès aux ressources partagées, un exécutif joue un rôle centralisateur, un véritable rôle d'interface qui aiguille les événements reçus du procédé vers les tâches qui les attendent, déclenche le réveil des tâches en attente d'un délai ou d'une date de démarrage, reçoit et retransmet des signaux de synchronisation ou des données entre des tâches asynchrones. Ainsi en général, un exécutif offre des services accessibles directement par invocation dans les tâches. Ces services sont de différentes natures :

– la gestion des tâches : l'exécutif réalise tous les services qui contrôlent l'exécution des tâches comme leur activation, leur suspension, leur reprise, leur terminaison forcée ;

– la gestion des événements matériels (interruption) et de synchronisation : la synchronisation est réalisée en permettant aux tâches d'émettre ou de recevoir des signaux « internes ». L'exécutif offre pour cela des services de génération et d'attente de ces « événements logiciels » ;

– la communication de messages entre tâches : par boîtes aux lettres, via des ports d'échanges de données, ou sous forme d'appels du type client-serveur ;

– la gestion du temps : il s'agit essentiellement de permettre le réveil différé, daté ou périodique des tâches qui le sollicitent ;

– la gestion des ressources partagées, de gestion de la mémoire, de gestion des interruptions, des exceptions etc.

Le standard OSEK/VDX qui est la référence pour les applications embarquées dans le secteur automobile, implémente tout ou partie de ces concepts. Cette proposition est le fruit des travaux de constructeurs et équipementiers automobiles européens depuis 1995. OSEK a été étudié par les équipes principalement allemandes et VDX par le GIE PSA-Renault. Différents travaux ont été menés dans le cadre d'OSEK/VDX qui ont conduit à la fourniture de quatre spécifications principales [OSE] : OSEK/VDX OS : les services de base du noyau du système d'exploitation ; OSEK/VDX COM : les services pour la communication entre des nœuds d'un système réparti ou la communication locale ; OSEK/VDX NM (Network Management) : les services de gestion et de surveillance du réseau ; OSEK/VDX OIL (OSEK Implementation Language) : langage de description pour la mise en œuvre automatisée d'une application.

Seuls OSEK/VDX OS et OIL nous intéressent ici. Tous les objets de la spécification OSEK/VDX sont des objets « statiques », ce qui est naturel dans ce cadre d'application. On trouvera ci-après une description très succincte de quelques concepts de l'OS (hors communication).

2.1. *La gestion des tâches*

La tâche est l'agent actif de l'application. C'est une portion de code séquentiel correspondant souvent à la notion de procédure (ou fonction) dans le langage de programmation utilisé pour coder l'application. OSEK/VDX utilise deux types de tâches définis ci-après : 1- les tâches basiques sont des modules sans point bloquant. La tâche est activée, elle s'exécute puis elle doit se terminer. Elle ne possède que trois états : *suspended* (inactive), *ready* (attente du processeur), *running* (elle a le processeur) ; 2- les tâches étendues sont composées de un ou plusieurs modules séparés par des invocations de services éventuellement bloquants (WaitEvent). Le diagramme d'état possède donc un état supplémentaire : l'état *waiting*. Voici quelques exemples de services : `ActivateTask` active la tâche désignée ; `TerminateTask` termine la tâche appelante ;

2.2. *Ordonnancement*

Les tâches possèdent une priorité utilisée pour l'ordonnancement. La valeur de priorité est statique, donc non modifiable, sauf par l'exécutif lorsqu'il met en œuvre l'algorithme « Immediate Priority Ceiling Protocol » pour la gestion des ressources. L'ordonnancement peut être : complètement non-préemptif (une tâche ne peut pas être interrompue par une tâche de priorité plus élevée), complètement préemptif, ou bien mixte (par tâche).

2.3. *Synchronisation des tâches*

Chaque tâche peut posséder un certain nombre d'événements pour lesquels des occurrences seront signalées par d'autres tâches (de type basique ou étendu) ou des ISRs (Interrupt Service Routine). La tâche propriétaire peut se mettre en attente (OU implicite sur la liste des événements nommés), et l'effacement de l'occurrence est à sa charge. Voici quelques exemples de services utilisables par les tâches : `SetEvent` signale une ou des occurrence(s) pour une tâche ; `WaitEvent` met la tâche appelante en attente sur le(s) événement(s) nommé(s) ; `ClearEvent` efface le ou les événement(s) désigné(s).

2.4. *Partage de ressources et exclusion mutuelle*

Dans un contexte où plusieurs tâches coopérantes sont en concurrence, il faut contrôler et obtenir l'accès cohérent à des ressources partagées par les tâches. OSEK/VDX assure la gestion de l'accès concurrent aux ressources partagées par le protocole *IPCP* (Immediate Priority Ceiling Protocol, version simplifiée du protocole PCP [SHA 90]), ce qui garantit la non-inversion de priorité et l'absence de blocage par usage des services `GetResource` et `ReleaseResource` « encadrant » la section critique, sous réserve d'une gestion bien ordonnée (LIFO) des prises de ressources multiples.

2.5. *Les objets Alarme et Compteur*

Ces objets permettent principalement le traitement de phénomènes récurrents dans le temps. Ils permettent également la mise en œuvre des chiens de garde. Le mécanisme est à deux « étages » , associant les compteurs et les alarmes. Un compteur est un objet destiné à l'enregistrement de « ticks » en provenance d'un dispositif émettant des stimuli. Plusieurs alarmes peuvent être associées à un même compteur, ce qui permet de constituer facilement, par exemple, des bases de temps. À chaque alarme est associée une action, par exemple l'activation d'une tâche, ou encore la signalisation d'une occurrence d'événement. Une alarme peut être unique ou cyclique, absolue ou relative.

2.6. *Prise en compte des interruptions*

En réponse à une sollicitation externe se traduisant par une requête d'interruption, il y a exécution d'une routine de traitement (ISR : Interrupt Service Routine). Cette routine, va interférer avec les tâches via les appels de service (sauf ceux d'attente) pour, par exemple, activer une tâche ou encore signaler l'occurrence d'un événement.

2.7. *Le langage OIL*

Le langage OIL (OSEK Implementation Language) est un langage textuel de description d'une architecture logicielle pour un calculateur. Il décrit tous les éléments qui constituent une application et leurs relations, par exemple : les tâches et leurs attributs, les ressources partagées, les alarmes et compteurs etc. Le langage sera illustré au travers de l'exemple de la section suivante.

3. Utilisation pratique

Dans cette section, nous allons présenter un exemple concret d'utilisation de Trampoline tel qu'il peut être utilisé dans un contexte d'enseignement.

Considérons une application relativement simple, basée sur l'exécution de 2 tâches périodiques. Chaque tâche allume une diode LED lors de son activation (pour avoir un retour visuel) et mets une sortie logique à 1 (pour l'exploitation à l'oscilloscope). Le code « utile » de la tâche est ensuite simulé par une boucle d'attente active, puis la sortie logique est remise à 0 et la LED est éteinte. Les caractéristiques des tâches sont définies statiquement (priorité et périodicité).

Pour mettre en pratique cette application simple, la première étape consiste à définir un diagramme de contexte de l'application. Dans notre cas, le système est très simple car il admet 4 sorties (logique et Led) et aucune entrée, il n'est donc pas représenté ici.

Un diagramme représentant l'architecture fonctionnelle de l'application est ensuite réalisé. Il permet de distinguer directement quels sont les différents objets OSEK qui seront utilisés, ainsi que les dépendances éventuelles entre les tâches, les conflits d'accès sur les variables ou ressources matérielles.

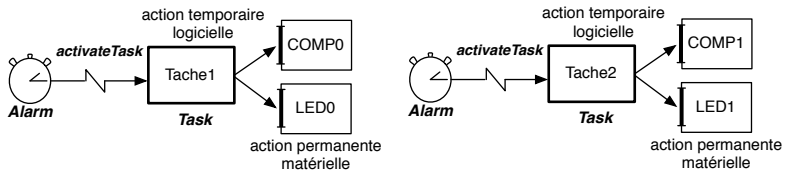

Figure 1. *Structure fonctionnelle.*

Sur la figure 1, les structure fonctionelles des tâches sont données, les 2 tâches étant fonctionnellement identiques. Un rectangle représente soit une action logicielle temporaire (tâche), soit une action matérielle (périphérique) ; dans ce dernier cas, le trait est plus fin. Les flèches indiquent le sens du transfert de données, en l'occurrence ici des tâches vers les périphériques. Le dessin du « réveil » représente l'objet ALARM

d'OSEK, objet qui est requis pour faire une tâche périodique. Enfin, le trait en zigzag sert à modéliser une action sur la tâche (ici une activation).

Dans le contexte OSEK/VDX, une tâche périodique utilise une alarme ainsi qu'un compteur. Pour une utilisation simple en TP, nous avons configuré Trampoline de manière à ce qu'un `timer` générant une impulsion à chaque milliseconde soit automatiquement associé à un compteur. Comme il n'y a pas d'appel système relatif au compteur, il n'apparaît pas dans le diagramme fonctionnel.

À partir de cette structure fonctionnelle, la traduction dans le langage OIL est relativement rapide, car les différents objets OIL apparaissent déjà dans le diagramme fonctionnel[1]. OIL est un langage relativement simple permettant de décrire les différents paramètres des objets OSEK, basée sur des associations clé/valeur. Quelquefois, une clé peut être associée à un sous attribut. Seul le code OIL relatif à la tâche 1 est indiqué ici :

```
COUNTER General_counter {
 TICKSPERBASE = 1;
 MAXALLOWEDVALUE = 65535;
 MINCYCLE =1;
};

TASK Tache1 {
 PRIORITY = 5;
 AUTOSTART = FALSE;
};
```

```
ALARM Active_Tache1 {
 COUNTER = General_counter;
 ACTION = ACTIVATETASK {
   TASK = Tache1;
 };
 AUTOSTART = TRUE {
   ALARMTIME = 1;
   CYCLETIME = 5;
   APPMODE=std;
 };
};
```

Le compilateur *Goil*, développé également dans le cadre du projet Trampoline, génère de manière automatique les structures de données internes propres à Trampoline, par compilation du fichier OIL de l'architecture. Il effectue aussi un certain nombre de vérifications sémantiques et d'optimisations (partage de piles pour les tâches de même priorité, optimisation de la taille des structures de données, ...).

Il ne reste à ce stade plus que la réalisation du corps de chaque tâche en C :

```
TASK(Tache1) {
 volatile int i;
 COMP0 = 0; LED0 = 0;
 for (i=0; i<6700; i++);
 LED0 = 1; COMP0 = 1;
 TerminateTask();
}
```

```
int main(void) {
 initES();
 StartOS(OSDEFAULTAPPMODE);
}
```

1. Certains paramètres sont définis globalement dans une autre partie du fichier OIL et ne sont donc pas indiqués ici

Seul le code de la tâche 1, de type basique, est décrit ici. On notera l'utilisation de l'appel système `TerminateTask()` à la fin de l'exécution de la tâche.

À partir d'un exemple simple, nous avons ici introduit les différentes étapes qui constituent la conception et le codage d'une application temps réel simple en se basant sur OIL et OSEK/OS. Cette approche est suffisamment simple pour être prise en main très rapidement par les étudiants. Au niveau pratique, il est possible par exemple ici d'observer le fait que la tâche de plus faible priorité est décalée dans le temps, car les 2 tâches sont réveillées en même temps. D'autre part, en modifiant les priorités et les temps d'exécution, il est possible de mettre en évidence les préemptions à l'oscilloscope et de mesurer les temps de commutations de tâches, ...

4. Dissémination

Le développement de Trampoline [IRC] a été commencé initialement avec un objectif de recherche, mais l'intérêt pour des retombées en enseignement est rapidement apparu. En effet, son utilisation s'intègre naturellement dans le cadre du module complémentaire MC-II2 qui introduit la notion de « Systèmes multitâches, systèmes temps réel » dans le cadre de la formation GEII à l'IUT (Programme Pédagogique National).

Depuis, Trampoline a gagné en maturité et est actuellement utilisé dans différents établissements de formation (IUT de Nantes, Licence Pro à Nantes et Rennes, écoles d'ingénieurs ESEO à Angers et INSA à Toulouse, Master EMARO à Nantes). On peut notamment remarquer qu'il est utilisé à différents niveaux de formation car il est aisé de moduler la difficulté des travaux pratiques en intervenant sur le niveau de conception demandée.

Trampoline a été développé dès le départ en dissociant la partie du code spécifique à une architecture de la partie générique. Il est ainsi relativement aisé de le « porter » sur une nouvelle cible. Les cibles embarquées actuelles sont *l'ARM7* (Arm), *l'AVR* (Atmel), le *H8300h* (Hitachi), *S12X* (Freescale), *PowerPC 565* (Freescale), *C167* (Infineon) et le *V850e* (Nec). De plus, une autre cible est proposée sur un système *POSIX* (Linux, MacOSX, ...). En utilisant cette cible, Trampoline est exécuté dans un *processus* Unix. Un autre processus est alors utilisé pour émuler l'envoi d'interruptions matériel et ainsi permettre l'utilisation de *timers*. Le fonctionnement de cet émulateur est décrit dans [BEC 06].

Au niveau recherche, le développement de Trampoline s'intègre dans le cadre des projets O4A puis O4A phase 2 (*Open for Autosar*[2]). Dans ce cadre, Trampoline est intégré dans l'offre Autosar de la PME Geensys et la PME Kereval intervient au niveau des tests fonctionnels. Ces projets sont soutenus par le pôle de compétitivité « Véhicules Haut de Gamme » des régions Pays de la Loire et Bretagne. Il est également utilisé au LAAS et en partenariat avec Renault sur des travaux concernant la robus-

2. Autosar (AUTomotive Open System ARchitecture) est un standard pour l'ensemble de l'architecture logicielle qui intègre une extension d'OSEK pour le logiciel de base.

tesse des systèmes embarqués, ainsi que dans la société Trialog dans le cadre du projet ANR Scarlet.

Trampoline est développé sous un modèle logiciel libre, avec une double licence BSD/LGPL. Ceci permet notamment à des tierces parties de participer à l'effort collectif, notamment par des nouveaux portages et des corrections directes grâce au caractère du logiciel à code source ouvert. Il est disponible sur le site de l'équipe STR de l'IRCCyN [IRC].

5. Conclusion

Nous avons développé *Trampoline*, une implémentation *libre* du standard OSEK/ VDX, un standard de système d'exploitation Temps Réel. Développé initialement pour des besoins de recherches, il est peu à peu utilisé dans l'enseignement supérieur, à tout niveau.

Après avoir abordé brièvement les grandes lignes du ce standard, nous avons donné un exemple d'utilisation pratique, en mettant l'accent sur la facilité de mise en œuvre en utilisant le langage OIL, permettant de décrire l'application OSEK. Il s'intègre ainsi naturellement dans le cadre du nouveau Programme Pédagogique National en IUT-GEII au niveau du module MCII2 dans le volume horaire prévu (30h).

Trampoline est téléchargeable sur `http://trampoline.rts-software.org`

6. Bibliographie

[BEC 06] BECHENNEC J.-L., BRIDAY M., FAUCOU S., TRINQUET Y., « Trampoline An Open Source Implementation of the OSEK/VDX RTOS Specification », *Emerging Technologies and Factory Automation - ETFA'06*, 2006.

[BER 87] BERRY G., COURONNE T., GONTHIER G., « Programmation synchrone des systèmes réactifs : le langage Esterel », *TSI*, vol. 6, n° 4, 1987, p. 305-316.

[IRC] IRCCYN, « Trampoline - OpenSource RTOS project », `http://trampoline.rts-software.org`.

[OSE] « OSEK/VDX Operating System Specification. OSEK Group. », `http://www.osek-vdx.org`.

[SHA 90] SHA L., RAJKUMAR R., LEHOCSKY J.-P., « Priority Inheritance Protocols ; an Approach to Real-Time Synchronisation », *IEEE Trans. on Computers*, vol. 39, n° 9, 1990, p. 1175-1185.

[STA 04] STANKOVIC J., RAJKUMAR R., « Real-Time Operating Systems », *The Journal of Real-Time Systems*, vol. 28, n° 2/3, 2004, p. 237-253.

[TRI 99a] TRINQUET Y., ELLOY J.-P., « *Systèmes d'exploitation temps réel : exemples d'exécutifs industriels* », chapitre Traité "Contrôle et Mesures" R8 052, Techniques de l'Ingénieur, 1999.

[TRI 99b] TRINQUET Y., ELLOY J.-P., « *Systèmes d'exploitation temps réel : les principes* », chapitre Traité "Contrôle et Mesures" R8 050, Techniques de l'Ingénieur, 1999.