Développement et intégration sous Linux/RTAI de composants logiciels temps-réel open-source génériques

Thibault GARCIA, Audrey MARCHAND et Maryline SILLY-CHETTO
IRIN (Institut de Recherche en Informatique de Nantes)
La Chantrerie BP 50609 Nantes Cedex 03 France
{maryline.chetto,audrey.marchand,thibault.garcia}
@iut-nantes.univ-nantes.fr

Résumé

Cet article décrit l'architecture open-source Cleopatre, ainsi que ses composants logiciels génériques et flexibles dédiés aux systèmes temps-réel embarqués. Cleopatre se base sur le système d'exploitation Linux/RTAI et ses composants sont issus de la recherche (des ordonnanceurs dynamiques basés sur l'algorithme Earliest Deadline, des mécanismes de tolérance aux fautes basés sur de la redondance temporelle...)

Les premières mesures de performances donnent des résultats intéressants notamment lorsque le nombre de tâches est important. La flexibilité permet à une même application de se dérouler différemment selon les composants qui sont chargés au sein de l'architecture Cleopatre.

Table des matières

Mots-clés

Temps-réel, Linux, RTAI, ordonnancement, synchronisation, tolérance aux fautes, serveurs de tâches apériodiques, matériel "sur étagères"

1 Introduction

Les travaux[1]décrits dans cet article consistent à améliorer Linux/RTAI (Real-Time Application Interface) en ajoutant de nouvelles fonctionnalités au noyau d'origine. On dote ainsi Linux/RTAI de mécanismes novateurs tels que des ordonnanceurs à priorités dynamiques, des protocoles d'accès aux ressources et des serveurs de tâches apériodiques. En particulier, une version de l'ordonnanceur Earliest Deadline First, le protocole d'héritage de priorité et des serveurs de tâches apériodiques ont été implémentés.

Un utilisateur sélectionne ces fonctionnalités dans des bibliothèques de modules selon les besoins de ses applications. De plus, un certain nombre d'outils ont été développés de manière à faciliter le débuggage et à tester l'implémentation en fournissant des informations textuelles[1].

2 Cleopatre et Linux/RTAI

2.1 Le projet Cleopatre

Ce travail s'inscrit dans le projet français Cleopatre (Composants Logiciels sur Etagères Ouverts Pour les Applications Temps-Réel Embarqués)[2]. L'objectif du projet est d'une part, de créer une librairie de modules logiciels open-source pour la conception de systèmes temps-réel et d'autre part, de participer à l'évolution de la communauté open-source Linux.

L'objectif principal est à la fois de démontrer la validité d'application et l'interopérabilité de ces composants logiciels par le biais de tests sur une plateforme de robotique mobile. Le travail est réalisé grâce à la collaboration d'utilisateurs finaux potentiels et d'équipes de recherche qui possèdent les qualités requises en matière de technologie temps-réel à savoir : ordonnancement et tolérance aux fautes, communication, vision temps-réel et contrôle/commande. Les innovations sur tous ces aspects auront un impact direct non seulement sur les performances mais aussi sur la fiabilité de tout système embarqué.

Le partenaire IRIN du projet a modifié l'ordonnanceur natif du système d'exploitation Linux/RTAI (développé au Dipartimento di Ingegneria Aerospaziale Politecnico di Milano) de sorte à ce qu'il soit plus souple et plus adapté aux applications des autres partenaires. Pour cela, l'IRIN a développé un module appelé TCL (Task Control Layer) qui est le seul à dépendre explicitement du système d'exploitation Linux/RTAI. Ce module supporte les autres modules d'ordonnancement développés par l'IRIN de sorte à ce qu'ils soient génériques et flexibles. Ces autres modules d'ordonnancement servent les applications des autres partenaires du projet.

[1]Travaux financés par le Ministère de la Recherche, Grant number 01 K 0742.

2.2 Le concept de RTAI

RTAI est un système d'exploitation temps-réel open-source basé sur le noyau Linux classique. Il consiste principalement en un dispatcher d'interruptions : RTAI capture les interruptions issues des périphériques et si nécessaire, les redirige vers Linux[3]. En d'autres termes, RTAI utilise le concept RTHAL (Real-Time Hardware Abstraction Layer) décrit sur la figure ci-dessous :

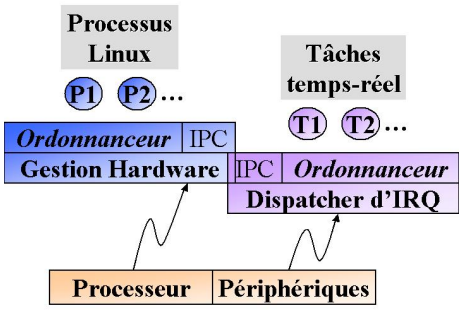

FIG. 1: Architecture de RTAI

RTAI est un patch pour Linux qui rend son noyau totalement préemptif, c'est à dire que les primitives du noyau peuvent être interrompues au profit de tâches très prioritaires dans le but d'améliorer la réactivité des applications temps-réel. RTAI considère d'ailleurs Linux et toutes les applications qui ne sont pas temps-réel comme des tâches de fonds activées lorsque aucune activité temps-réel n'est présente.

2.3 Des fonctionnalités modulaires

RTAI est composé de différents modules. Le module principal doit être chargé pour toute application et assure les fonctionnalités de bases. Pour des développements avancés, RTAI offre des fonctions pour l'ordonnancement, la gestion des FIFOs et l'utilisation de mémoire partagée. RTAI comprend trois ordonnanceurs temps-réel préemptifs à priorité fixe. Le projet Cleopatre propose une augmentation de la modularité de Linux/RTAI en modifiant son ordonnanceur de sorte à ce qu'il accepte de nouveaux composants novateurs.

En termes de performances, RTAI est très compétitif, offrant typiquement des temps de changement de contexte de 4μs et des temps de réponse aux interruptions de 20μs. Il est par ailleurs possible de créer des tâches périodiques de fréquence maximale 100KHz en mode "periodic" et 30KHz en mode "one-shot"[3]. Cleopatre apporte une optimisation des gestions logicielles des listes de tâches pour profiter au mieux de ces performances.

3 L'architecture du système d'exploitation

3.1 Description globale

Les ordonnanceurs Cleopatre sont modulaires et intègrent des fonctionnalités de recherche novatrices. La commande basique 'insmod' de Linux, charge les modules en mémoire, les place dans le noyau Linux et construit les dépendances entre modules, en utilisant les noms de toutes les fonctions impliquées dans l'application.

Les différents modules sont complètement indépendants mais nécessitent un module bas-niveau appelé TCL qui fournit une API minimale indépendante du système d'exploitation et qui coordonne les autres modules entre eux. Ensuite, les applications accèdent aux interfaces des modules intermédiaires dont elles ont besoin pour utiliser leurs services. Tout module peut être remplacé par un autre module avec des fonctionnalités similaires ou différentes, pourvu qu'il possède la même interface.

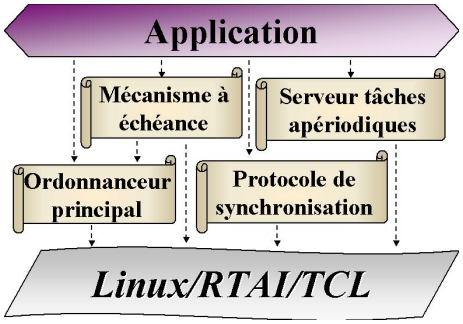

FIG. 2: Architecture globale

Un des objectifs importants du projet est de construire des composants aussi génériques que possible. C'est pourquoi certaines données liées directement au système d'exploitation ont été isolées. C'est notamment le cas pour tout ce qui est relatif à la gestion des tâches (création et destruction), aux changements de contexte et aux interruptions.

En effet, ces points ne peuvent pas être génériques car ils dépendent directement des choix effectués au moment de l'implémentation du système d'exploitation et également de l'architecture matérielle utilisée. Chaque tâche créée avec le système d'exploitation RTAI nécessite la définition des tailles de pile et de tas, ainsi qu'une fonction signalant la préemption de la tâche, alors qu'en général les systèmes d'exploitation ne laissent pas l'utilisateur gérer ces données.

Le logiciel d'application est donc le seul programme pouvant fournir ces données spécifiques, mais les modules intermédiaires doivent être capables de créer des tâches eux-mêmes. La question qui se pose alors est la suivante : comment construire un module intermédiaire qui soit indépendant du système

d'exploitation ? Les spécificités du système d'exploitation sont rassemblées dans une structure spéciale appelée TCLCreateType.

3.2 La couche TCL de contrôle des tâches (Task Control Layer)

TCL est au coeur de l'architecture du système. Il tient à jour l'horloge système, élit une tâche selon le paramètre de priorité dynamique et fournit une interface basique aux composants d'ordonnancement du système.

Primitives	Description
TCL_CREATE	crée une nouvelle tâche
TCL_DESTROY	détruit une tâche
TCL_KILL	réinitialise une tâche
TCL_READY	passe une tâche à l'état prêt
TCL_BLOCK	bloque une tâche
TCL_SCHEDULE	préempte une tâche

TABLE 1: *Interface de TCL*

Combinée à la structure TCLCreateType, TCL fournit une interface à Linux/RTAI, de manière à ce que les composants de plus haut niveau soient complètement indépendants du système d'exploitation. Pour adapter le système d'ordonnancement Cleopatre à un autre système d'exploitation temps-réel comme RT-Linux, ou à d'autres versions de Linux/RTAI, il suffit donc simplement d'adapter la couche TCL.

TCL est la clé de voûte de la portabilité car elle a été conçue de manière à rendre les autres composants indépendants du système d'exploitation. Aussi, TCL est la seule partie devant être modifiée pour exporter l'architecture Cleopatre à tout autre système d'exploitation temps-réel.

3.3 Mécanismes bas niveau

Pour réussir la phase d'intégration, la couche TCL (Task Control Layer) considère l'ordonnanceur natif de RTAI comme une tâche de fond. Ainsi, l'ordonnanceur natif de RTAI gère ses propres tâches, mais celles-ci sont ordonnancées lorsqu'aucune activité Cleopatre n'est présente. Le système d'ordonnancement est donc compatible avec les applications du système RTAI natif.

Une architecture flexible a été conçue pour utiliser les différents composants optionnels d'ordonnancement sur un même système. Les composants peuvent s'enregistrer pour accéder à une des listes de tâches qui seront exécutées. Chacune de ces listes est caractérisée par un paramètre de criticité statique et contient des tâches d'un même type telles que des tâches de fond, des tâches

normales ou des tâches de secours. Les tâches placées dans la liste la plus critique sont toujours exécutées en premier. Les listes sont ordonnées selon un paramètre de priorité dynamique fixée par les modules d'ordonnancement selon leurs protocoles. Le module TCL gère donc les listes de tâches pour tous les composants d'ordonnancement et les élit.

TCL facilite aussi l'ordonnancement avec un mécanisme de partage du signal d'horloge. C'est-à-dire que les modules d'ordonnancement peuvent enregistrer leurs propres vecteurs d'interruptions qui sont exécutés régulièrement pour mettre à jour les paramètres d'ordonnancement.

4 Les Composants du système

4.1 Composants d'ordonnancement

Un simulateur en couches a été implémenté pour préparer la phase d'intégration. Il teste les différents algorithmes d'ordonnancement de manière identique bien que son interface dépende de l'algorithme d'ordonnancement considéré. La couche intermédiaire établit le lien entre les interfaces génériques du simulateur et des algorithmes. Les algorithmes utilisant la même interface peuvent être simulés avec la même couche intermédiaire. La couche bas niveau utilisée pour chaque simulation, génère des événements aléatoires devant être gérés par les algorithmes (la couche de plus haut niveau).

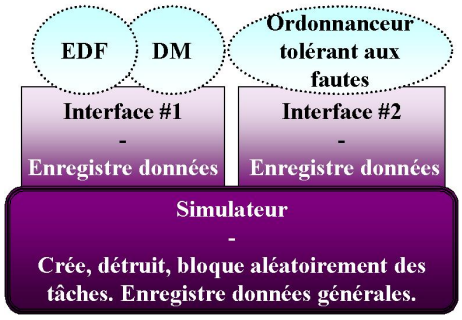

FIG. 3: Simulateur en couches

Le simulateur nous fournit un rapport de données avec lequel il est possible de vérifier manuellement toutes les listes à chaque fois qu'un événement s'est produit. Pour faciliter cette vérification, la couche intermédiaire a été pourvue d'une fonction qui vérifie la validité de la séquence d'ordonnancement actuelle. Elle surveille de nombreux paramètres tels que le nombre de tâches ordonnancées, le nombre d'appels aux fonctions de création et de destruction des tâches, etc.

4.2 Composants de contrôle d'accès aux ressources

Deux classes de protocoles de gestion de ressources ont été développées : des sémaphores mutex et des sémaphores génériques. Chaque classe est composée de plusieurs composants de gestion de ressources utilisant la même interface. Ainsi, un composant peut facilement en remplacer un autre de la même classe. Deux versions de chaque protocole ont été créées selon l'ordre dans lequel les tâches obtiennent une ressource après avoir été bloquées. Cet ordre dépend soit des priorités dynamiques des tâches soit de l'ordre de blocage (FIFO).

Protocoles	Classement FIFO	Classement par priorités
AUCUN	FIFO	priorité
HÉRITAGE DE PRIORITÉ	F_PIP	P_PIP
SUPER PRIORITÉ	S_SPP	S_SPP

TABLE 2: *Composants ressources*

Les interfaces des deux classes de composants s'apparentent à POSIX.

Sémaphores	Mutex	POSIX 1003.1
SEM_CREATE	MTX_CREATE	SEM_CREATE
SEM_DESTROY	MTX_DESTROY	SEM_DELETE
SEM_TAKE	MTX_TAKE	SEM_WAIT
SEM_TAKE_IF	MTX_TAKE_IF	SEM_TRYWAIT
SEM_RELEASE	MTX_GIVE	SEM_POST

TABLE 3: *Interface des composants*

4.3 Composants pour le service des tâches apériodiques

Les serveurs de tâches apériodiques sont très importants pour les systèmes temps-réel dur, car ils vérifient que les tâches apériodiques peuvent être ordonnancées sans entraîner de fautes temporelles. Trois composants de serveurs de tâches apériodiques ont été implémentés.

Le serveur Background (BGS) ordonnance les tâches apériodiques comme des tâches de fond et garantit le fait que les tâches apériodiques n'affectent pas les tâches périodiques.

Le serveur Total Bandwidth (TBS) [4] attribue une échéance virtuelle aux tâches apériodiques selon la charge du processeur. Les tâches apériodiques sont alors ordonnancées avec les tâches périodiques. TBS est un serveur optimal.

Lorsqu'il n'y a pas d'activité apériodique, le serveur Earliest Deadline Late (EDL) [5][6] ordonnance les tâches périodiques selon le protocole Earliest Deadline First. A chaque fois qu'une tâche apériodique est réveillée, il ordonnance

les tâches périodiques au plus tard de manière à récupérer un maximum de temps CPU pour exécuter les tâches apériodiques au plus tôt.

Serveurs	Composants
Serveur Background	BGS
Serveur Total Bandwidth	TBS
Serveur Earliest Deadline Late	EDL

TABLE 4: *Composants serveurs de tâches*

Primitives	Description
ATS_CREATE	Crée des tâches
ATS_BEGIN	Réveille toutes les tâches périodiques
ATS_DESTROY	Détruit toutes les tâches
ATS_WAKEUP	Réveille une tâche apériodique

TABLE 5: *Interface des serveurs de tâches*

4.4 Composant de tolérance aux fautes

Le Mécanisme à Echéance (Deadline Mechanism[7]) a été implémenté au sein d'un composant spécifique. Ce mécanisme améliore la fiabilité d'un système en incorporant de la redondance : chaque tâche est composée de deux versions, une version primaire et une version de secours.

Primitives	Description
FT_CREATE	Crée une nouvelle tâche
FT_DESTROY	Détruit toutes les tâches
FT_BEGIN	Réveille toutes les tâches ensemble

TABLE 6: *Interface*

5 Tests et exemples

5.1 Evaluation

Les composants Cleopatre ont été testés ensemble et séparément. Un mécanisme de débuggage a été implémenté pour les développeurs de nouveaux composants. Il rapporte toute modification survenue dans les listes de tâches prêtes et il peut être activé de façon précise. Ce mécanisme est très utile pour valider des composants.

Des mesures de performances sont en cours de réalisation pour quantifier les surcoûts par rapport au RTAI natif. Les premiers résultats indiquent que, contrairement à Linux/RTAI, le système IRIN ne dépend pas du nombre de

tâches créées (même celles qui ne sont pas utilisées). Les mesures ont aussi mis en évidence deux manières de programmer les applications Cleopatre : l'une est simple et claire, l'autre, proche de la manière de programmer dans les systèmes temps-réel classiques, est très performante.

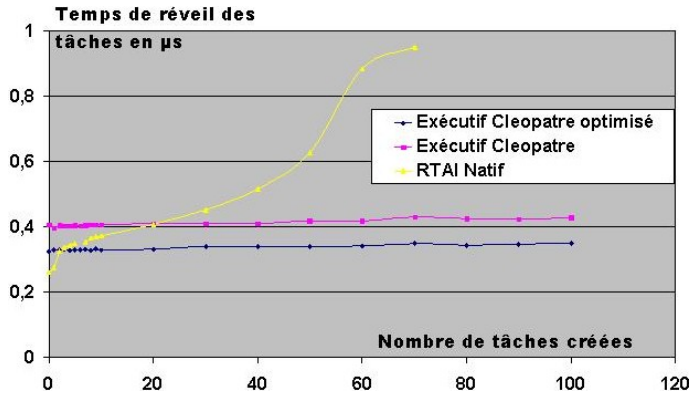

FIG. 4: Temps de réveil d'une tâche

Les primitives de synchronisation sont des primitives clefs de la programmation temps-réel. Comparées aux durées d'exécution des primitives similaires de Linux/RTAI, celles de Cleopatre concernant la réservation (wait et wait_if) et la libération (signal) des mécanismes de synchronisation sont meilleures.

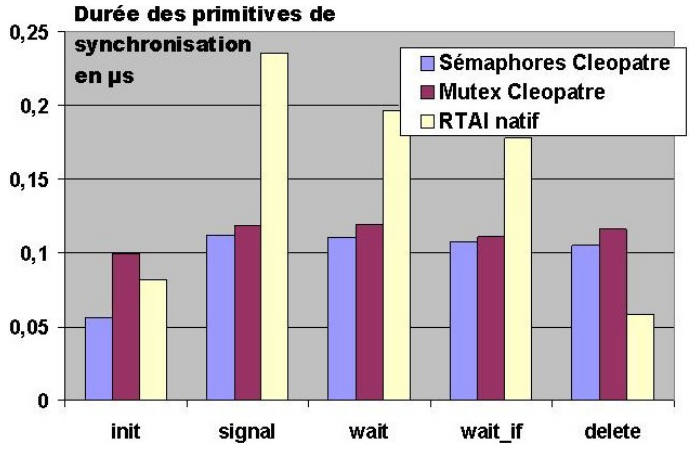

FIG. 5: Temps de réveil d'une tâche

En contre-partie, les primitives de création (init) et de destruction (delete) sont souvent plus longues. Il s'agit d'un bon compromis, car ces dernières primitives ne sont employées qu'une fois chacune et à des moments qui ne sont

pas critiques, c'est à dire lors de l'initialisation de l'application, avant que le système passe en mode temps-réel, et lorsque de la terminaison de l'application, une fois que le système est revenu du mode temps-réel.

5.2 Exemple

On considère deux tâches périodiques partageant une même ressource. La tâche 1 et la tâche 2 sont périodiques de périodes respectives 0.5s et 1s. La tâche 1 réserve la ressource lors de sa première activation, puis la libère à la période suivante. Lorsque la tâche 2 s'exécute, elle demande à son tour la ressource. Les tâches 3 et 4 sont des tâches apériodiques réveillées sur un signal d'interruption externe. Dans le but de simplifier l'exemple, la durée d'exécution de chaque tâche est fixée à 4ms.

Tâche	Délai critique	Période	WCET
1	16 ms	0.5 s	4 ms
2	512 ms	1 s	4 ms
3	8 ms	none	4 ms
4	12 ms	none	4 ms

TABLE 7: *Valeurs numériques*

Cette application a été testée avec deux combinaisons de composants. Tout d'abord, nous avons chargé l'ordonnanceur principal Deadline Monotonic (DM[8]) associé au serveur de tâches apériodiques Background (BGS).

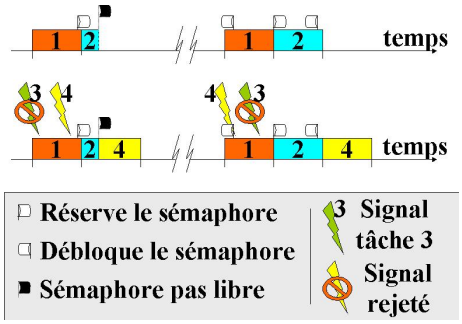

FIG. 6: Exemple : DM + BGS

On constate que BGS ne modifie jamais la séquence des tâches périodiques établie par l'ordonnanceur principal.

Ensuite, l'ordonnanceur principal Earliest Deadline First (EDF[9]) associé au serveur de tâches apériodiques Total Bandwidth a été utilisé. Cette combinaison nous a permis d'obtenir de meilleurs résultats, puisque la tâche apériodique 3 est rejetée une fois de moins.

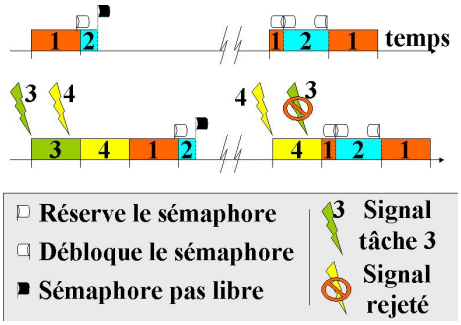

FIG. 7: Exemple : EDF + TBS

5.3 Code source

Le code source de cet exemple est relativement clair et donne un aperçut de la simplicité de programmation temps-réel sous Cleopatre.

```
struct AtsTaskType t1; /* Descripteurs tâches  */
struct AtsTaskType t2;
struct AtsTaskType t3;
struct AtsTaskType t4;

struct MtxType mtx;  /* Descripteur sémaphore  */

void fct1() {        /* -- Tâche 1 -- */
  simul_wait(4);     /* Simulation d'exécution */
  mtx_take(&mtx);    /* Verrouillage sémaphore */

  Dsch_wait();       /* Attendre période       */

  mtx_release(&mtx); /* Déverrouillage sem     */
  simul_wait(4);     /* Simulation d'exécution */
}

void fct2() {        /* -- Tâche 2 -- */
  mtx_take(&mtx);    /* Verrouillage sémaphore */
  simul_wait(4);     /* Simulation d'exécution */
  mtx_release(&mtx); /* Déverrouillage sem     */
}

void fct3() { simul_wait(4); } /* Tâches       */
void fct4() { simul_wait(4); } /* apériodiques */

int init_module(void) {
  struct TCLCreateType create = {0, 2000, 0, 0};

  mtx_create(&mtx);  /* Création du sémaphore  */
```

```
                    /* Création des tâches    */
  ats_create(&t1, fct1,  16,  500, 4, create);
  ats_create(&t2, fct2, 512, 1000, 4, create);
  ats_create(&t3, fct3,   8,    0, 4, create);
  ats_create(&t4, fct4,  12,    0, 4, create);

  ats_begin(1000);   /* Réveil après 1s        */

  start_rt_timer(nano2count(TIMERTICKS));
  return 0;
}

void cleanup_module(void) {
  stop_rt_timer();

  ats_destroy(&t1);  /* Destruction des tâches */
  ats_destroy(&t2);
  ats_destroy(&t3);
  ats_destroy(&t4);
  mtx_destroy(&mtx); /* Destruction sémaphore  */
}
```

6 Conclusion et perspectives

TCL (Task Control Layer) est une modification de l'ordonnanceur de RTAI qui augmente ses possibilités en gérant les composants logiciels Cleopatre. Ces composants sont sous forme de modules génériques et flexibles. Ils gèrent différents aspects de l'ordonnancement : L'ordonnanceur principal lui-même, deux mécanismes de synchronisation indépendants, des serveurs de tâches apériodiques et un composant de tolérance aux fautes. Ces composants sont optionnels mais apportent de grandes facilités pour la programmation des applications. De plus, plusieurs protocoles de chaque mécanisme sont souvent disponibles et ils peuvent être changés sans recompiler les applications.

Dans le but de démontrer que les nouveaux composants sont indépendants du système d'exploitation, un projet d'adaptation de la couche TCL à RTLinux s'achève. Dans un même temps, des protocoles de gestion de ressources plus sophistiqués tels que le protocole à priorité plafond (Dynamic Priority Ceiling Protocol[10]) ou la politique de gestion de ressource à pile (Stack Resource Policy[11]) sont en cours d'implémentation. Des mesures de performances de l'exécutif CLEOPATRE ainsi que leurs comparaisons avec de Linux/RTAI natif sont réalisées.

L'exécutif Cleopatre a été intégré à une plate-forme de robotique mobile et est en train d'être intégré à une nouvelle plate-forme plus performante et complexe. Les autres partenaires du projet vont apporter leurs propres composants (algorithmes de contrôle/commande, vision temps-réel) pour compléter les fonctionnalités et assurer un contrôle de navigation élaborée.

FIG. 8: Plate-forme robotique mobile Cleopatre

Références

[1] A. Marchand, C. Plot and M. Silly-Chetto, June 2002, *Real-time mobile robot navigation with LINUX/RTAI*, PROCEEDINGS OF INT. CONF. ON CONTROL AND AUTOMATION, XIAMEN, CHINA.

[2] http ://www.cleopatre-project.org/

[3] Lineo Inc., P. Mantegazza, 2000, *RTAI Programming Guide 1.0*, Lindon, Utah.

[4] M. Caccamo, G. Lipari, G. Buttazzo, December 1999, *Sharing Resources among Periodic and Aperiodic Tasks with Dynamic Deadlines*, PROCEEDINGS OF THE IEEE REAL-TIME SYSTEMS SYMPOSIUM, PHOENIX, ARIZONA.

[5] H. Chetto, M. Chetto-Silly, 1989, *Some results of earliest deadline scheduling algorithm*, IEEE TRANS. ON SW ENG., 18(8), PP.736-748.

[6] J.P. Lehoczky, L. Sacha, Y. Ding, 1992, *An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority pre-emptive systems*, PROCEEDINGS OF THE IEEE REAL-TIME SYSTEMS SYMPOSIUM, PP.110-123.

[7] A.L. Liestman, R.H. Campbell, 1986, *A fault tolerant scheduling problem*, IEEE TRANS. ON SOFTWARE ENGINEERING, VOL.12 PP.1089-1095.

[8] J.Y.T. Leung, M.L. Merril, 1980, *A note on pre-emptive scheduling of periodic real-time tasks*, INFORMATION PROCESSING LETTERS, VOL. 20 N°3 PP. 115-118.

[9] C. Liu, J.W. Layland, 1973, *Scheduling algorithms for multiprogramming in a hard real-time environment*, JOURNAL OF ACM, VOL.20 N°1 PP.46-61.

[10] M.I. Chen, K.J. Lin, 1990, *Dynamic Priority Ceilings : A Concurrency Control Protocols for Real-Time Systems*, REAL-TIME SYSTEMS JOURNAL, 2(4), PP. 325-346.

[11] T.P. Baker, 1990, *Stack-Based Scheduling of real-time Processes*, PROCEEDINGS OF IEEE REAL-TIME SYSTEMS SYMP., PP.191-200.