ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE DE LA PRODUCTION AUTOMATISÉE
M.Ing.

PAR
VINCENT POIRÉ

MIGRATION DE LOIS DE CONTRÔLE D'UN ENVIRONNEMENT DE SIMULATION VERS UN CADRE D'APPLICATION ROBOTIQUE

MONTRÉAL, LE 29 NOVEMBRE 2006

CE MÉMOIRE A ÉTÉ ÉVALUÉ
PAR UN JURY COMPOSÉ DE:

M. Sylvain Lemieux, directeur de mémoire
Département du génie de la production automatisée à l'École de technologie supérieure

M. Pascal Bigras, codirecteur de mémoire
Département du génie de la production automatisée à l'École de technologie supérieure

M. Ilian Bonev, président du jury
Département du génie de la production automatisée à l'École de technologie supérieure

M. Julien Beaudry, examinateur externe
Institut de Recherche d'Hydro-Québec (IREQ), Hydro-Québec

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC
LE 27 NOVEMBRE 2006
À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

MIGRATION DE LOIS DE CONTRÔLE D'UN ENVIRONNEMENT DE SIMULATION VERS UN CADRE D'APPLICATION ROBOTIQUE

Vincent Poiré

SOMMAIRE

Le prototypage rapide de contrôleur (PRC) est largement utilisé dans le milieu de la recherche pour sa facilité d'utilisation. L'utilisation d'un cadre d'application robotique (CAR) apporte une grande flexibilité mais demande un investissement important pour la programmation, ce qui n'est pas toujours souhaitable dans l'univers de la recherche. Combiner les deux stratégies en faisant migrer des lois de contrôle d'un environnement de PRC vers un CAR permet d'accélérer le processus de développement tout en conservant une grande flexibilité.

Ce mémoire propose un protocole de migration qui permet cette combinaison. Le développement de ce protocole, son guide d'utilisation ainsi que son évaluation sont présentés dans ce document.

Un exemple d'utilisation du protocole de migration est illustré à l'aide de la migration d'un contrôleur à couple précalculé pour un robot planaire à deux degrés de liberté. Cette migration est effectuée à partir d'un environnement de PRC composé de la chaîne d'outils Matlab / Simulink / RTW. Le contrôleur généré est encapsulé dans une bibliothèque partagée puis intégré au CAR Microb. La bibliothèque partagée peut être générée pour Windows XP ou pour QNX.

Les résultats obtenus montrent que le processus de migration est viable et prometteur.

MIGRATION PROCEDURE OF CONTROL LAWS FROM A GRAPHICAL SIMULATION ENVIRONMENT TO A ROBOTIC FRAMEWORK

Vincent Poiré

ABSTRACT

Rapid controller prototyping (RCP) is widely used in the research community for its ease of use. Robotic Frameworks (RF) bring great flexibility but require significant investments in programming, which is not desirable in the research world. Combining both strategies by migrating control laws from a RCP environment to a RF can speed up the development process while retaining the flexibility.

This thesis proposes a migration protocol that makes this combination possible. The development of the protocol, its user's guide and its evaluation is presented in this document.

We illustrate this approach with the migration of a computed torque controller for a virtual 2-DOF planar robot. The migration is carried out with a RCP consisting of a Matlab/Simulink/RTW toolchain. The computed torque controller is encapsulated in a dynamic library, and then integrated with a RF called Integrated Modules for Robot Control (MICROB). The shared library can be generated either for Windows XP or for QNX.

The results show that such a migration process is both practical and promising.

REMERCIEMENTS

Je dédie ce mémoire à Edwidge, ma grand-mère et mon modèle de persévérance.

Tout d'abord, je remercie Sylvain Lemieux et Pascal Bigras pour leur aide, leur disponibilité et leurs précieux conseils. Je mentionne que ce projet n'aurait pu être mené à terme sans le soutien financier généreux de l'ÉTS, du CRSNG et de la Caisse Populaire de Lévis. Je souligne ma gratitude aux gens de l'IREQ qui m'ont épaulé tout au long du projet, plus particulièrement Michel Blain, Régis Houde, Julien Beaudry ainsi que Philippe Hamelin et David Mercier. Merci également à Nicolas et François pour leurs efforts de covoiturage.

J'aimerais remercier spécialement Jérôme Loisel pour l'aide qu'il m'a apportée dans la traduction de mon article et pour ses commentaires divers, tant sur mon travail que sur d'autres éléments biscornus.

Je remercie chaleureusement ma mère et mon père pour leur sagesse, leur appui indéfectible et leurs encouragements. Un gros merci à Normand pour ses conseils et à mon frère pour m'avoir montré que c'était possible de réussir. Je remercie ma copine Amélie pour son support et qui, par sa simplicité, incarne pour moi le bonheur. Mention spéciale à Lucie, à Maître Mononque et à la petite Florence pour les activités ludiques pratiquées en leur compagnie qui m'ont permis de recharger mes piles.

Un gros merci à Wiwun pour ses conseils informatiques forts utiles. Merci aux festivaliers de la santé Listerine[MD]: Migo et Mélanie, Perks, Blais et Marianne. Merci à toutes ces personnes qui m'ont permis de me changer les idées: Lévesque, Martin, Dym, Caro, Gaëlle, Marc Guimond, Gabrielle, Delo et Maître Brochet. Merci à Marie-Eve qui, malgré qu'elle ne soit plus dans ma vie, m'a aidé à me rendre jusque là. Finalement, je remercie Boris Vian [1] pour l'inspiration, THQ pour DoW, St-Louis pour son existence et Fidel pour ses installations.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS ET SIGLES

ÉTS	École de technologie supérieure
IREQ	Institut de Recherche d'Hydro-Québec
CAR	Cadre d'Application Robotique
PRC	Prototypage rapide de contrôleur
IREQ	Institut de Recherche d'Hydro-Québec
Microb	Module Intégré de Contrôle Robotique
dll	Bibliothèque dynamique sous Windows (*dynamic link library*)
so	Bibliothèque partagée sous Unix (*shared object*)
RTW	Real-Time Workshop
TLC	Target Langage Compiler
TMF	Template MakeFile

INTRODUCTION

Certaines tâches peuvent difficilement être exécutées par des robots industriels disponibles sur le marché. Des applications spéciales, qui nécessitent des opérations en contact ou dans des environnements non-structurés, doivent être réalisées à l'aide d'algorithmes de contrôles avancés qui ne sont généralement pas implantés dans les contrôleurs de robots fournis par les fabricants. Le développement d'outils qui facilitent la validation et l'implantation de nouvelles lois de commande tout en fournissant une flexibilité suffisante pour assurer le développement de contrôleur de haut niveau est un aspect important de la recherche en robotique. Ce mémoire, qui est issu d'une collaboration entre l'École de technologie supérieure (ÉTS) et l'Institut de Recherche d'Hydro-Québec (IREQ), porte sur le développement d'un tel outil.

L'ÉTS forme des ingénieurs d'application, dont des spécialistes en robotique. Plusieurs chercheurs de l'ÉTS travaillent dans ce domaine en collaboration avec l'industrie. Ils utilisent généralement des outils de prototypage rapide de contrôleurs (PRC), plus particulièrement la suite Matlab / Simulink / Real-Time Workshop [2]. Ces outils graphiques permettent d'implanter des algorithmes complexes rapidement et sont parfaitement adaptés à l'univers académique.

Par contre, d'un point de vue pratique, l'utilisation d'outils de PRC comporte des désavantages majeurs. La compatibilité avec le matériel informatique peut parfois être limitée, particulièrement lorsque l'on fait appel à du matériel spécialisé. De plus, les outils de PRC sont davantage conçus pour faire du traitement de signal que de la gestion événementielle. Le traitement de signal est très bien adapté à la mise en œuvre de lois de commande, mais il n'est pas approprié lorsqu'il s'agit de concevoir un contrôleur de robot complet avec les mécanismes de haut niveau tel que la sécurité et la supervision.

Les chercheurs de l'IREQ effectuent également des travaux de recherche en robotique. Les projets de recherche de l'IREQ servent principalement à répondre aux besoins de la société d'État qui produit, transporte puis distribue l'électricité dans l'ensemble du territoire québécois. L'IREQ utilise un cadre d'application robotique (CAR) développé à l'interne appelé Microb [3]. Ce CAR permet d'avoir un maximum de flexibilité tout en permettant de conserver un temps de développement acceptable. Le CAR permet de mettre en œuvre des contrôleurs de robots complexes munis de mécanismes de sécurité et de supervision de haut niveau.

Par contre, l'utilisation du CAR a certains effets peu souhaitables. Apprendre à utiliser le CAR de façon efficace requiert des efforts considérables de par la nature peu intuitive de l'outil et de sa complexité. De plus, des connaissances avancées en programmation et en systèmes temps réel sont requises. Comme ce CAR est peu utilisé à l'extérieur de l'IREQ, les chercheurs externes sont peu enclins à apprendre ce logiciel peu répandu dont l'évolution est lente par rapport aux outils de PRC.

Un projet de meulage à sec (MAS) a été mis de l'avant à l'IREQ en collaboration avec Sylvain Lemieux [4], qui est professeur à l'ÉTS et également directeur de ce mémoire. Ce projet a pour but de prouver qu'il est possible d'arriver à effectuer des opérations en contact à partir d'une base mobile, en l'occurrence un sous-marin. Le manipulateur résultant, tel qu'illustré à la Figure 1, possède six degrés de liberté actionnés. Les trois premiers servent à simuler le mouvement oscillatoire du sous-marin et les trois derniers représentent le manipulateur qui est destiné à être fixé sur le sous-marin.

Le robot MAS, conçu pour développer et expérimenter de nouveaux algorithmes de commande de force, ne peut utiliser un contrôleur commercial standard car ce dernier comporte de trop grandes limitations techniques. Il est donc requis de concevoir un nouveau contrôleur. Ce contrôleur a été développé à l'aide du CAR Microb.

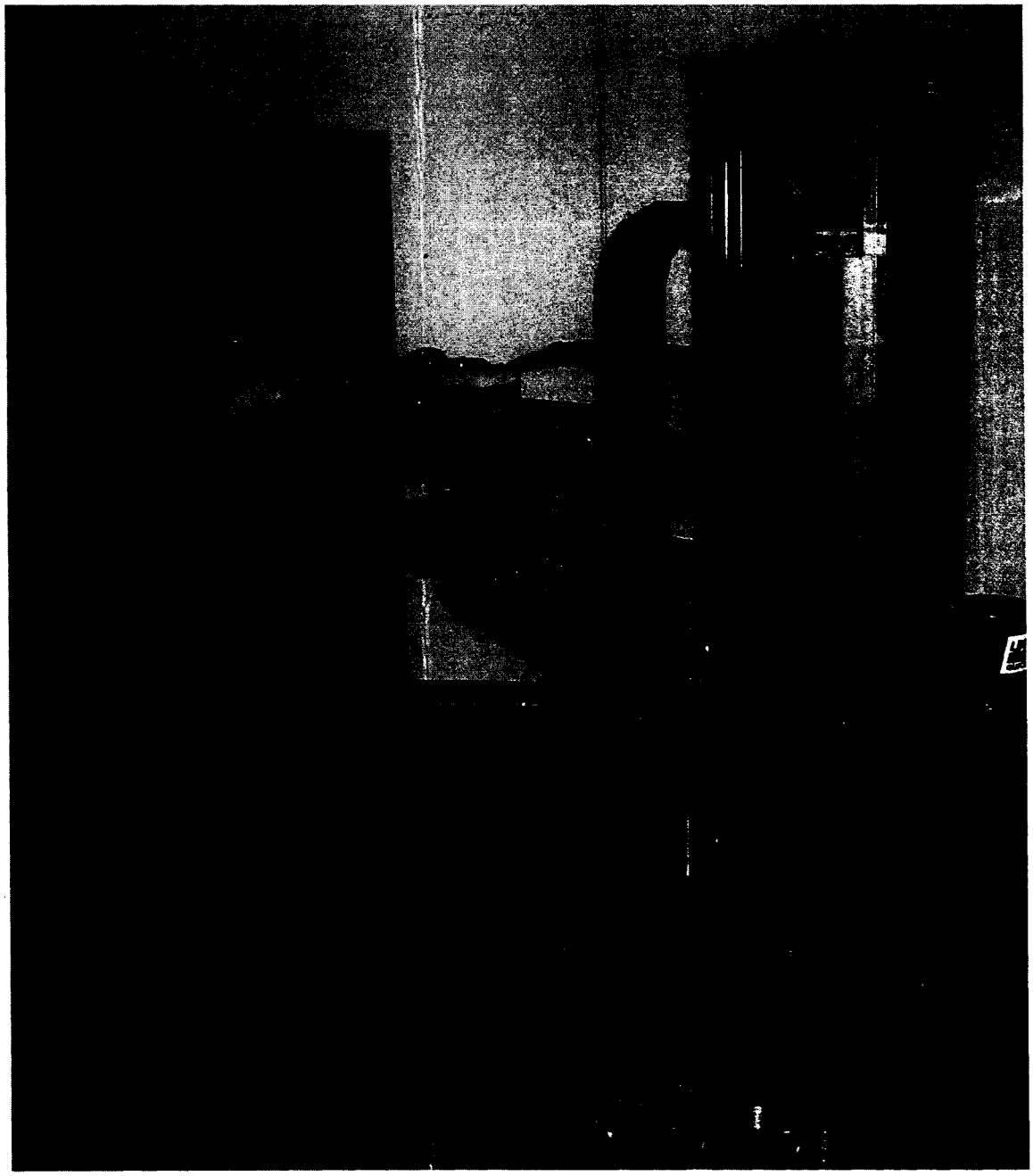

Figure 1 Manipulateur MAS
(source: IREQ)

Le processus de développement du contrôleur a débuté par l'interfaçage des composants informatiques par un expert en programmation Microb. Cet expert a ensuite mis en place une structure de contrôleur permettant d'ajouter des lois de commande et des interfaces utilisateur. Cependant, cette approche de développement nécessite que chaque version de l'algorithme de commande soit intégrée au contrôleur Microb par l'expert en programmation Microb.

Cet exemple montre bien le désavantage de l'utilisation du CAR qui nécessite absolument un expert en programmation. Une question se pose alors: peut-on bénéficier des avantages du CAR (comme la flexibilité) et du PRC (comme la simplicité et la rapidité) en optimisant l'utilisation des ressources?

Dans ce mémoire, il est proposé de combiner l'approche de développement basée sur un PRC avec celle basée sur un CAR, pour permettre le développement rapide de contrôleurs plus fiables, adaptables, sécuritaires et performants.

Dans un premier temps, une étude préliminaire à la réalisation est effectuée. Une revue de la littérature (CHAPITRE 1) est réalisée afin de présenter l'état de l'art du point de vue technique et scientifique. Par la suite, la problématique (CHAPITRE 2) est décrite afin de cerner les enjeux et les objectifs puis la méthodologie de recherche (CHAPITRE 3) est présentée.

Dans un deuxième temps, les décisions sont prises quant à la façon dont la combinaison est effectuée. L'analyse des besoins des usagers (CHAPITRE 4) est réalisée, ce qui permet d'effectuer les choix technologiques (CHAPITRE 5).

Dans un troisième temps, la combinaison du CAR et des outils de PRC est réalisée. La mise en œuvre (CHAPITRE 6), l'utilisation (CHAPITRE 7) et l'évaluation (CHAPITRE 8) de la nouvelle combinaison sont exposées.

L'utilisation combinée d'un CAR et d'outils de PRC ne semble pas documentée à ce jour, mis à part dans un article [5] issu des travaux de ce projet. Ce sujet mérite d'être traité en profondeur à l'aide d'un travail de recherche car il s'agit d'intégrer deux logiciels complexes qui peuvent avoir une incidence sur la sécurité et l'efficacité des systèmes robotiques.

CHAPITRE 1

REVUE DE LITTÉRATURE

1.1 Introduction

Cette section a pour but de présenter le contexte de la programmation des contrôleurs robotiques utilisés dans le cadre de la recherche appliquée. Premièrement, une mise en contexte de la recherche appliquée en robotique est effectuée. Deuxièmement, les différentes méthodes de programmation des robots sont exposées. Troisièmement, les différents types de contrôleurs disponibles sont décrits afin de mettre en relief quels sont les types de contrôleurs les plus appropriés dans un contexte de recherche appliquée.

1.2 Profil actuel de la robotique industrielle

Les robots industriels prennent de plus en plus de place dans l'industrie. Entre 1986 et 2002, le parc de robot installés aux États-Unis a plus que quadruplé [6]. Selon la Commission Économique de l'Europe des Nations Unies (UNECE) et de la Fédération Internationale de Robotique (IFR), le nombre de commandes de robots industriels, tout pays confondus, a augmenté de 93.9 % en 2004 par rapport à 1996 [7]. Ces informations montrent que la robotique industrielle est en plein essor, et ce, depuis un bon moment.

Plusieurs facteurs contribuent à rendre ce moyen de production davantage prépondérant. Selon Colestock [8], l'accroissement de la popularité des robots industriels est due à l'augmentation de leur précision et de leur puissance mécanique. De plus, l'auteur soutient que cet accroissement de performance a été conjugué à une baisse des coûts des robots de l'ordre de 70% par rapport à 1995: un robot de 100 000$ en 1995 se vendrait 30 000$ en 2005 [8].

À ce jour, les robots industriels fabriqués par les grands manufacturiers sont principalement utilisés pour faire de l'assemblage, de la soudure ou de l'emballage et de la palettisation [9]. D'autres applications existent, notamment dans l'industrie alimentaire et botanique, mais elles demeurent marginales.

Ces limites d'application peuvent toutefois être repoussées grâce à la recherche appliquée en robotique, qui est présentée à la section suivante.

1.3 La recherche en robotique industrielle

Les grands manufacturiers ne sont pas en mesure de combler tous les besoins de l'industrie, comme le prouvent les chiffres qui suivent. Selon l'Associated Press [10], l'investissement annuel en recherche robotique serait de 100 millions de dollars américains pour l'Europe, du même montant pour la Corée et le Japon. D'après le même article [10], les États-unis investiraient 500 millions annuellement pour la recherche en robotique.

Toutefois, selon Dutkiewicz et coll.[11], les chercheurs qui travaillent avec des robots industriels ont à faire face à de nombreux problèmes dus au fait que les contrôleurs de robots industriels sont fermés et non standardisés. Les auteurs [11] soutiennent que pour être en mesure d'avoir un robot plus adapté pour leurs expérimentations, ils doivent soit construire un nouveau manipulateur ainsi qu'un contrôleur ou encore reprogrammer un contrôleur existant au complet.

Développer ou reprogrammer un contrôleur de robot demande beaucoup d'efforts car les systèmes mécaniques, électriques, électroniques et informatiques doivent être agencés harmonieusement. Ces systèmes sont brièvement présentés dans la section qui suit.

1.4 Anatomie d'un système robotique industriel

Tout système robotique, bien qu'il en existe une grande variété, possède certaines composantes standard pouvant être catégorisées. Une catégorisation de ces composants est tentée dans cette section de façon à mettre en relief la place qu'occupe le contrôleur de robot dans ce type de système.

La configuration générale d'un système robotique générique est donc la suivante [12]:

1. Le mécanisme du robot;
2. Les capteurs externes;
3. Le contrôleur de robot;
4. Le système de simulation et de programmation hors-ligne.

Il est pertinent de décrire les composantes de ce système, dont l'interaction est illustrée à la Figure 2. Le mécanisme du robot est caractérisé par ses composantes physiques (membrures, actionneurs, capteurs, etc.) qui accomplissent concrètement les tâches. Les capteurs externes permettent d'effectuer des mesures de l'environnement pour permettre au programme de prendre des décisions quant aux actions à exécuter.

Le contrôleur de robot orchestre le fonctionnement du mécanisme d'après les informations des capteurs et de sa programmation. Ce dernier fait donc le lien entre les utilisateurs, les capteurs et les actionneurs. Le système de simulation permet à l'utilisateur de voir le comportement programmé du robot avant d'exécuter le programme sur le système physique, ce qui évite d'éventuels bris. Le système de programmation hors-ligne permet quant à lui de programmer une tâche avant de l'exécuter sur le robot. Il est important de préciser qu'il est maintenant possible que le système de programmation hors-ligne et le simulateur soit à l'intérieur du contrôleur.

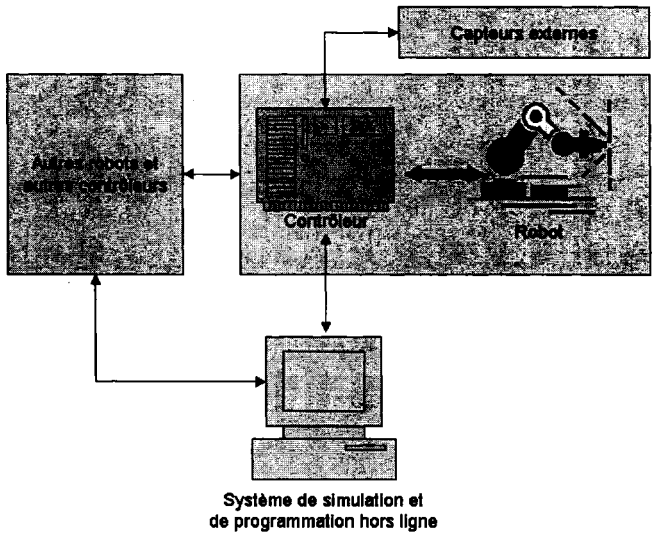

Figure 2 Illustration de la configuration générale d'un système robotique [12]

Le cœur du système est le contrôleur de robot car c'est par lui que transite l'ensemble des informations. Le développement de ce dernier nécessite de nombreuses heures de développement car il s'agit d'une tâche critique et complexe. Selon Loffler, Dawson et coll. [13], la difficulté de ce travail réside particulièrement dans le fait qu'il nécessite une expertise dans plusieurs domaines, notamment en robotique, en programmation temps réel, en intégration de matériel et en gestion d'accès simultanés.

Pour ajouter à la complexité, plusieurs types de programmation sont possibles à l'intérieur même du contrôleur. La distinction entre ces différents types de programmation est présentée dans la sous-section suivante.

1.5 L'organisation logicielle d'un contrôleur

D'après Kapoor et Tesar [14], un contrôleur peut être divisé en trois couches logicielles distinctes (voir Figure 3), soit la couche d'interface matérielle, la couche opérationnelle et la couche de programmation robotique.

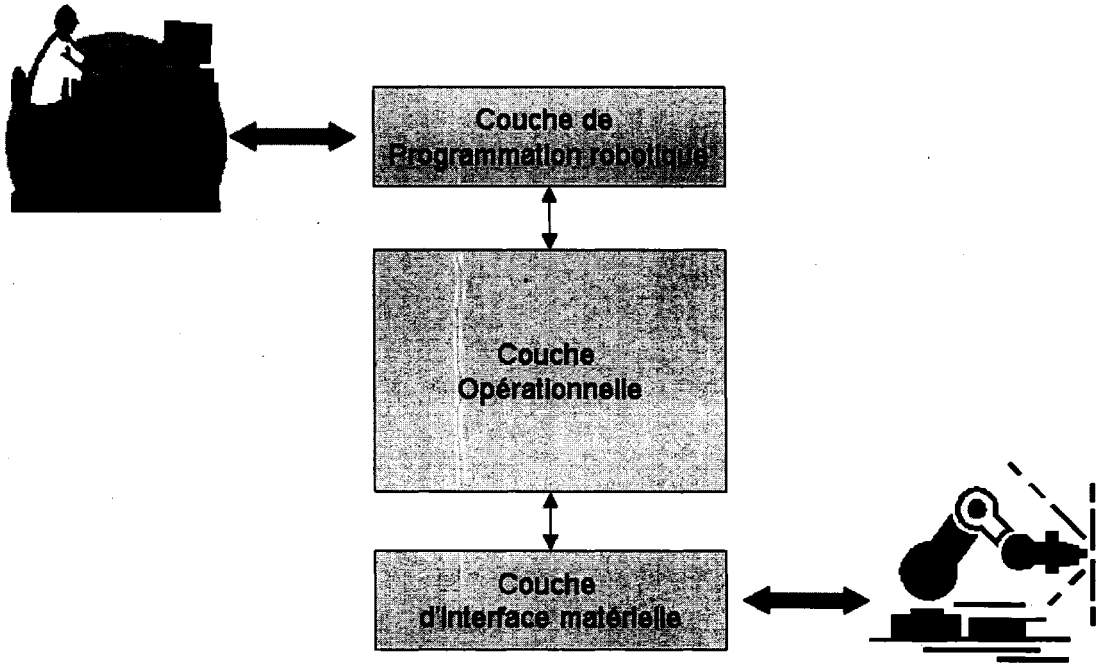

Figure 3 Illustration de l'organisation logicielle d'un contrôleur de robot

La couche d'interface matérielle a pour but de servir d'interface entre les logiciels de servocommande, les périphériques et les bus de communication [14]. Du point de vue pratique, il s'agit du système d'exploitation et des pilotes de périphériques.

La couche de programmation robotique est l'interface homme-machine du système robotique [14]. Deux types de programmation sont possibles dans cette couche, soit la programmation manuelle et la programmation assistée. Ils se définissent comme suit [15]:

- Programmation manuelle: programmation, typiquement sans le robot, qui se fait hors ligne. Elle peut s'effectuer graphiquement avec un schéma, un diagramme de fluence (*flowchart*) ou un graphe. Cette programmation peut également se faire textuellement dans un langage soit:

  - Propre au manufacturier du contrôleur, par exemple avec Karel, Rapid, etc.;
  - Générique, par exemple avec C++, Java, etc.;
  - Comportemental, par exemple avec Yampa [16], dérivé du langage Haskell [17].

- Programmation assistée: programmation typiquement effectuée lorsque le robot est en ligne (*on-line*), c'est-à-dire que sa programmation nécessite que le robot soit sous tension. Trois méthodes catégorisent ce type de programmation:
  - Par l'exemple: programmation assistée la plus courante, utilisant généralement un boîtier de commande (*teach pendant*) et plus rarement la voix ou le geste. L'usager fait bouger le manipulateur, le contrôleur enregistre les actions effectuées (mouvement, préhension, etc.) dans le but de les reproduire sur demande.
  - Par l'information: programmation basée sur les informations que le robot reçoit via un de ses capteurs, qui a pour effet de déclencher l'exécution d'une tâche connue a priori ou d'un modèle d'action généralement prédéfini par programmation.
  - Par l'apprentissage: programmation rarement utilisée se basant sur le renforcement ou la répression de certains comportements (réseau de neurones, etc.).

La couche opérationnelle fait le lien entre les demandes des clients, les informations reçues des capteurs, des contraintes de sécurité et de tout ce qui n'est pas directement lié à l'usager ou au matériel. Cette couche assure la communication entre les couches d'interface matérielle et de programmation, supportant donc le mode de programmation manuelle ou assistée ou les deux, en plus de gérer l'accès au matériel, de gérer les communications, etc.: c'est le point central du contrôleur de robot.

Cette catégorisation des couches logicielles montre que la couche de programmation, comprenant la programmation manuelle et assistée, est davantage liée à l'exécution d'une tâche précise du robot tandis que la couche opérationnelle permet de faire fonctionner le système: sans elle, rien n'est possible.

La partie logicielle qui sera abordée dans ce mémoire concernera uniquement le développement de la couche opérationnelle. Avant d'aborder le développement de cette couche, il convient de présenter au sein de quels types de contrôleur elle s'effectue.

1.6 Types de contrôleurs robotiques

La couche opérationnelle diffère selon le type de contrôleur utilisé. Cette section présente les différents types de contrôleurs documentés dans la littérature. Selon Dixon, Moses et coll.[18], il existe quatre types de contrôleur robotique:

1. contrôleur propriétaire;
2. contrôleur développé avec un langage de programmation traditionnel;
3. contrôleur développé avec un cadre d'application robotique;
4. contrôleur développé grâce à un environnement de simulation graphique avec un générateur de code.

Ces différents types seront décrits dans les quatre sous-sections qui suivent. Le but de cet exercice est de bien comprendre quels sont les contrôleurs qui sont le plus appropriés pour la recherche appliquée en robotique.

1.6.1 Le contrôleur propriétaire

La première alternative, le contrôleur fourni par un fabricant, permet de programmer facilement les tâches à effectuer par le robot: c'est un système clé en main. Dans ce cas,

le fabricant fournit un contrôleur de robot complet et fonctionnel avec un interpréteur de commande qui est spécifique au langage propriétaire.

Les commandes disponibles sont celles qui sont liées à la définition et à l'exécution des tâches pour laquelle le robot a été conçu. Elles incluent généralement le mouvement entre deux points, l'approche normale vers une surface, la préhension ou le dépôt d'un objet, etc. Ce langage textuel contient les principales commandes robotiques et ressemble souvent à du basic, comme le montre la portion de code Karel suivante:

```
$MOTYPE = JOINT          -- activer le mode articulaire
MOVE TO depart           -- aller à la position départ
OPEN HAND 1              -- ouvrir le préhenseur 1
MOVE TO distributeur     -- aller chercher la pièce
CLOSE HAND 1             -- agripper la pièce
MOVE TO boite            -- aller à la position de la boîte
OPEN HAND 1              -- déposer la pièce
```

Il existe trois types de contrôleurs propriétaires [19]:

1. Fermé: système où il est très difficile, voire impossible, de modifier le matériel et la structure logicielle pour quelqu'un d'autre que le manufacturier original. Ce sont les contrôleurs les plus fréquemment rencontrés dans l'industrie. Ils possèdent comme avantage d'être économiques et éprouvés. C'est la solution la plus intéressante lorsque le contrôleur répond aux exigences de l'application désirée;
2. Ouvert: systèmes qui permettent de modifier tous les aspects, qu'ils soient logiciels ou matériels. Comme ces contrôleurs sont généralement des produits uniques, ils sont plus onéreux et demandent plus de validation que les contrôleurs propriétaires;
3. Hybride: système semi-ouvert qui permet d'accéder de l'extérieur aux fonctions de bas niveau. Des compagnies ont mis en marché dans les années 1990 ces contrôleurs qui tentent d'allier les avantages des contrôleurs fermés et ouverts. Par exemple, cette approche permet de connecter un ordinateur standard sur un

contrôleur hybride qui peut aller jusqu'à servir uniquement de boîte d'amplification pour les moteurs. L'auteur souligne que cette approche a réduit les coûts de systèmes robotiques car elle permet la réutilisation tant de matériel éprouvés que de logiciels fiables. La compagnie allemande Kuka produit un tel type de contrôleur [20].

Presque chaque compagnie robotique d'envergure possède son propre langage textuel dont quelques exemples sont donnés au Tableau I.

Tableau I

Les fabricants de robots et leurs langages associés

Compagnie	**Langage**
Fanuc / GMF	Karel
CRS	Rapl
KUKA	KRL
ABB	Rapid
Adept	V+

Un langage de programmation propriétaire doit être utilisé en conjonction avec un contrôleur compatible, habituellement fourni par le fabricant à l'achat du manipulateur. Ce langage est habituellement facile à utiliser mais a comme principal désavantage d'être peu adaptable à une nouvelle tâche pour laquelle le système robotique n'a pas été conçu. Cette spécialisation pour la robotique, bien qu'elle apporte de la simplicité au utilisateurs des systèmes robotiques, peut parfois être limitative: il peut être difficile de modifier la boucle de contrôle ou encore de créer un protocole de communication sur mesure. De plus, ce type de langage ne peut être utilisé que lorsqu'il existe un contrôleur compatible avec le robot à utiliser, ce qui limite encore davantage les possibilités.

1.6.2 Le contrôleur basé sur un langage de programmation traditionnel

La seconde alternative consiste à développer un contrôleur au complet avec un langage traditionnel, tel le C++, le Pascal, le Java, etc. Cette approche est extrêmement flexible mais demande énormément de temps de développement ainsi qu'une connaissance très poussée des contrôleurs de robots en général, ce qui n'est pas à la portée de tous. De plus, chaque nouveau contrôleur est reprogrammé à partir de zéro, ce qui génère une perte de temps considérable.

1.6.3 Le contrôleur développé avec un cadre d'application robotique

La troisième alternative est le contrôleur développé à l'aide d'un Cadre d'Application Robotique (CAR). C'est en fait un cadre d'application de type « Infrastructure système » adapté aux contraintes et réalités des systèmes robotiques.

Le Grand dictionnaire terminologique de l'Office de la langue française [21] définit un cadre d'applications (appelé en anglais *application framework* ou encore *object-oriented application framework*) de la façon suivante:

« Infrastructure logicielle qui facilite la conception des applications par l'utilisation de bibliothèques de classes ou de générateurs de programmes en programmation orientée objet. »

Il existe plusieurs cadres orientés objet, dont la *Microsoft Foundation Class* (MFC), les *Distributed Component Object Model* (DCOM) et les applets Java. Selon Fayad et Schmidt [22], un cadre d'application orienté objet a les avantages suivants:

- Modularité: encapsule les détails d'implémentation pour faciliter la compréhension et la maintenance du logiciel;

- Réutilisation: définit des composants génériques pouvant être réemployés pour créer de nouvelles applications;
- Extensibilité: permet d'étendre une interface stable et son comportement à une autre interface qui requiert un comportement similaire;
- Inversion du contrôle: appel du code de l'usager par le cadre d'application plutôt que l'inverse.

Par contre, ces mêmes auteurs [22] mentionnent que ce type d'approche technologique comporte également ses désavantages:

- Les connaissances requises pour développer un cadre d'applications appartiennent souvent à un nombre limité d'experts;
- Apprendre à utiliser un cadre d'applications orienté objet de façon efficace requiert des efforts considérables;
- Il peut être difficile d'intégrer plusieurs cadres d'application car ils n'ont pas nécessairement été conçus pour cela;
- Effectuer la maintenance d'un cadre d'application requiert une compréhension profonde de ce dernier;
- Le déverminage et la validation d'un programme développé à l'aide d'un cadre d'application peuvent être difficiles pour plusieurs raisons:
    - Les bogues peuvent venir de plusieurs sources, notamment du cadre d'application, de l'utilisation faite de ce dernier ou encore du code du programme développé à l'aide du cadre d'application;
    - L'inversion de contrôle peut rendre plus difficile la localisation d'un bogue;
    - Le code du cadre d'application peut ne pas être disponible;
    - Le code du cadre d'application peut être mal compris des usagers.

Malgré la liste des désavantages, les auteurs [22] soutiennent que les cadres d'applications orientés objet seront le noyau de l'avant-garde technologique en matière

de développement logiciel au cours du 21ième siècle. Il est intéressant de constater que cet avis se reflète dans les publications du IEEE [23]: une recherche du mot clé *framework* par année, toutes disciplines confondues, montre qu'il y a une augmentation constante du nombre de documents publiés depuis les 15 dernières années (voir Figure 4).

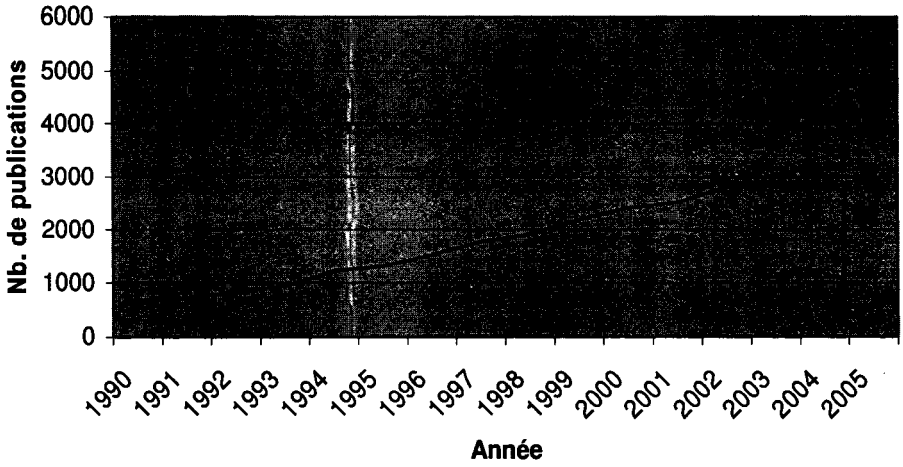

Figure 4 Évolution du nombre de publications par année de l'IEEE traitant de « framework » [23]

Un Cadre d'Application Robotique (CAR) est simplement l'application à la robotique des cadres d'application exposés précédemment. Un CAR est une bibliothèque modulaire de fonction couplée à une structure d'exécution appliquée à la robotique et développée dans un langage traditionnel. Il permet d'avoir la grande flexibilité du développement à l'aide d'un langage traditionnel tout en diminuant considérablement le temps de développement.

Plusieurs CAR existent, dont Orocos[24], Oscar [25] et Microb [26]. Ces CAR ont été développés dans différents contextes mais toujours en réponse au manque de contrôleurs de robots ouverts et flexibles disponibles sur le marché.

Orocos est l'acronyme de « Open RObot COntrol Software ». Ce projet à code source ouvert subventionné par la Communauté Économique Européenne (CEE) était à l'origine composé de quatre institutions de recherche: l'Université Catholique de Leuven [27], le KTH en Suède [28], le LAAS en France [29] et l'ULM en Allemagne [30]. Son développement a été coordonné de la Belgique par le docteur Herman Bruyninckx. L'objectif fondateur de ce CAR est de fournir des outils de développement de contrôleurs standardisés et adaptés à la recherche académique ainsi qu'à l'éducation [31].

Toutefois, l'union de plusieurs centres de recherche ne s'est pas avérée fructueuse car l'intégration logicielle des segments développés dans les différentes universités n'a jamais eu lieu [32]. La portion française, le « LAAS Open Software for Autonomous Systems » [29], ne semble plus disponible publiquement. La contribution allemande, le « OROCOS::SmartSoft » [33], est distribué séparément tout comme la contribution suédoise « ORCA » [32]. Ces deux contributions sont davantage des boîtes à outils modulaires que de véritables CAR. Un seul cadre d'application distinct a finalement émergé de cette union: OROCOS. OROCOS est encore activement développé, maintenu, utilisé et distribué sous licence LGPL [34] à ce jour par la portion belge du projet [24].

Oscar est l'acronyme de « Operational Software Components for Advanced Robotics » et a été développé à l'Université du Texas à Austin par le Robotic Research Group [35]. Ce projet, fondé sur la thèse du docteur Kapoor [36], a deux principales orientations: prodiguer des outils de commande optimale et favoriser la reconfiguration pour les robots redondants [14]. Malgré que l'auteur parle de l'évolution du CAR via la contribution de ses utilisateurs [14], le code source ainsi que les binaires ne sont pas disponibles et les détails sur la licence sont introuvables sur le site officiel . Mis à part la présence des publications Robotic Research Group [35], rien n'indique que le CAR est encore en évolution.

Microb est l'acronyme de « Module Intégré de Contrôle ROBotique » et a été développé par l'Institut de Recherche d'Hydro-Québec (IREQ). Le démarrage de ce projet a principalement été motivé par les objectifs suivants (extraits de [37], p.6):

- Concevoir une bibliothèque commerciale de manière à garantir la robustesse des différents robots utilisés « ... »;
- Garantir l'évolution continue de la bibliothèque pour éviter d'avoir à reconstruire tous les contrôleurs « ... »;
- Faciliter la gestion des ressources « ... » en s'assurant que Microb est un produit indépendant de ses concepteurs;
- Créer un outil de pointe permettant à Hydro-Québec de conserver son leadership mondial en matière de contrôle robotique;
- Créer de nouveaux projets d'innovation « ... ».

Microb est utilisé dans le cadre de plusieurs projets robotiques (voir Figure 5) à Hydro-Québec, notamment pour un manipulateur [38], un robot sous-marin [39] ou encore comme une boîte à outil pour un logiciel de planification de meulage robotisé [26]. Deux projets étudiants utilisent actuellement Microb pour développer leurs contrôleurs: le club Walking Machine de l'ÉTS [40] (robot mobile) et Groupe Robofoot ÉPM [41] (multi-robots). Auparavant, d'autres projets universitaires ont déjà utilisé cette technologie (Hélicoptère [26], SARCOS [26], SONIA-ÉTS [42] et SAE Robotique [43]) mais rien n'indique qu'ils utilisent encore Microb.

Le projet a été entrepris peu avant 2000 et sa distribution sous licence *Lesser General Public License* (LGPL) [34] a débuté en 2002. L'utilisation de cette licence avait pour but de permettre d'élargir le nombre d'utilisateurs pour dynamiser le développement et d'assurer la pérennité du CAR [37]. Malheureusement, malgré le fait que Microb soit encore distribué à ce jour, il semble qu'il soit davantage utilisé à l'IREQ qu'ailleurs, ce qui oblige l'IREQ à supporter les coûts de maintenance. De plus, comme l'évolution de

Microb dépend de ses utilisateurs et que ces derniers sont peu nombreux, elle est relativement lente.

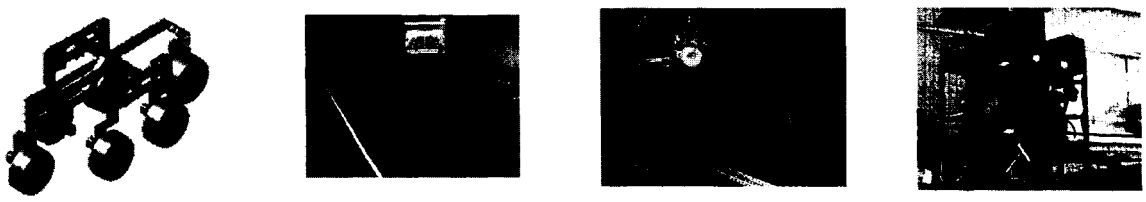

Figure 5 Plateformes robotisées utilisant Microb
(source: [3])

Après analyse de ces différents CAR, il apparaît que les caractéristiques communes sont généralement les suivantes:

- Polyvalent: conçu pour tous types de robots;
- Flexible: s'adapte aux besoins spéciaux en évoluant;
- Modulaire: permet de sélectionner un ou plusieurs modules, selon les besoins;
- Portable: généralement utilisable sous plusieurs systèmes d'exploitation;
- Indépendant: non lié spécifiquement à un manufacturier de robot ou de matériel;
- Standard: structure toujours les contrôleurs de la même façon;
- Réutilisable: diminue le temps de développement via la réutilisation du code;
- Adaptable: exécuter le même code en simulation que lors de l'exécution temps réel;
- Convenable: contient des outils de base liés à la robotique (cinématique, dynamique, génération de trajectoire, etc.).

Toutefois, un CAR demande à l'usager d'avoir des connaissances avancées en programmation dans un langage orienté objet et des systèmes temps réels. De plus, cette approche a tous les désavantages liés aux cadres d'application (voir ci-dessus).

1.6.4 Le contrôleur développé en prototypage rapide

Le quatrième choix, la génération de code via un outil de simulation graphique tel Matlab/Simulink ou Scilab/Scicos, est associée à la notion de prototypage rapide de contrôleurs (PRC), appelée en anglais *rapid controller prototyping* (RCP). Cette approche modifie le processus traditionnel de conception de contrôleurs en automatisant les étapes de développement logiciel.

La conception est vue comme un processus itératif, qui nécessite plusieurs itérations avant d'aboutir au produit répondant aux spécifications. Budgen [44] présente la conception comme étant un problème malicieux (*wicked*). Il donne quelques caractéristiques d'un problème malicieux:

- Il n'y a pas de bonnes ou de mauvaises solutions au problème, seulement quelque chose entre les deux;
- Il n'y a pas de formulation définitive ni de fin au problème;
- Il n'y a pas de test immédiat ou ultime pour évaluer la solution;
- Il n'y a pas un nombre de solution défini au problème;
- Il s'agit d'un problème unique.

La conception n'est pas un processus linéaire comme celui de la résolution d'un problème scientifique [44]. Elle est souvent perçue comme un cheminement suivant une spirale dont le point le plus près du centre représente l'obtention d'une solution jugée acceptable, comme le montre la Figure 6.

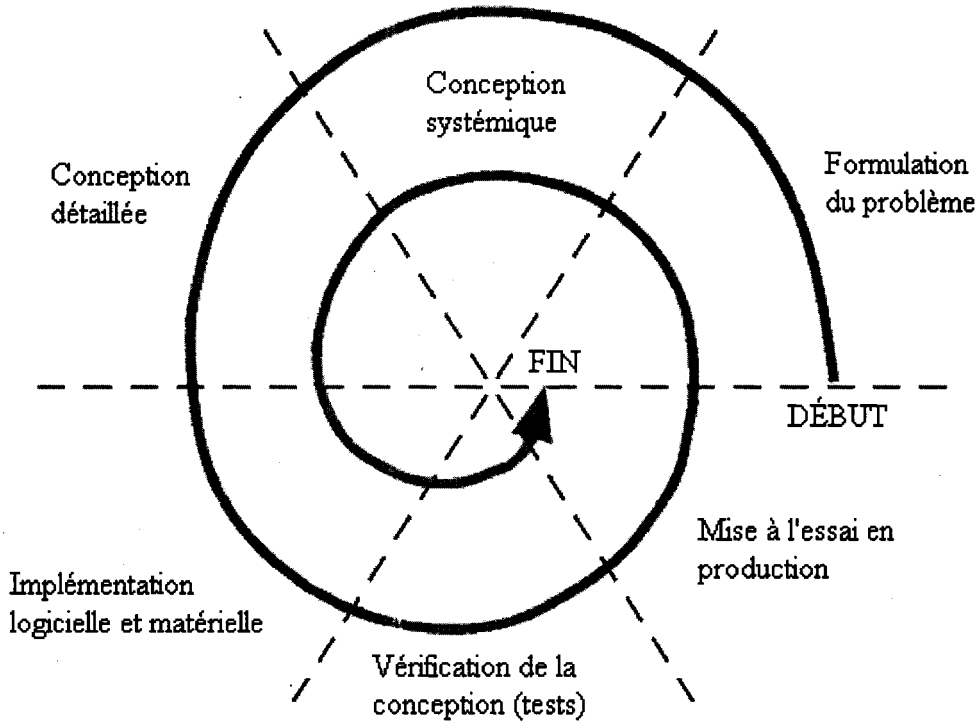

Figure 6 Illustration du processus de design [45]

Comme présenté à la Figure 7, deux principaux facteurs différencient le processus de prototypage rapide:

1. L'itération est brève et automatique dans le cas du prototypage rapide au lieu d'être manuelle. La brièveté de l'itération est due au fait qu'elle est gérée par un seul environnement de développement conçu à cette fin. Cette approche permet de se rapprocher plus rapidement du centre de la spirale de conception;
2. La partie logicielle et matérielle est déjà prise en charge par le logiciel, ce qui retire une étape du processus. Cela permet au concepteur de mettre l'accent sur les algorithmes plutôt que de mettre du temps sur le design informatique du contrôleur.

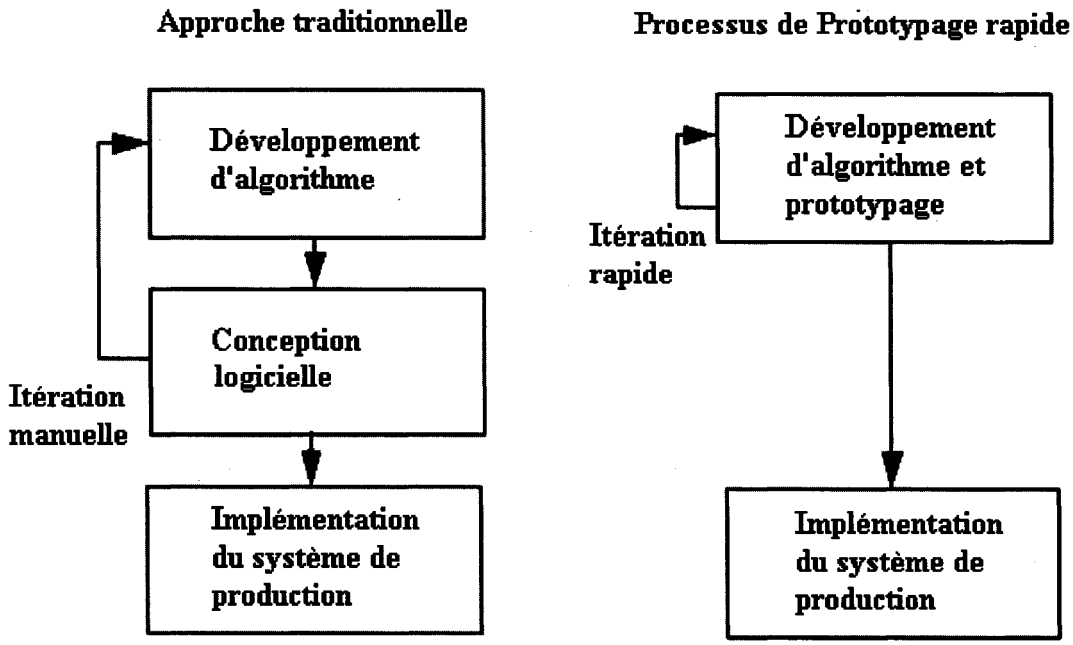

Figure 7 Comparaison graphique entre l'approche traditionnelle et le prototypage rapide [45]

Le PRC permet donc de rapidement simuler et expérimenter des algorithmes de contrôle sans avoir à consacrer beaucoup d'efforts pour développer la plateforme logicielle. Ces environnements de simulation graphique possèdent de puissants outils de calcul et de modélisation particulièrement adaptés au contrôle ainsi qu'à la simulation de système réels.

Trois environnements de simulation munis de générateur de code existent dans la littérature, soit la suite Matlab / Simulink / RTW [2], la suite Scilab /Scicos [46] et la suite NI MATRIXx / SystemBuild / Autocode [47]. En se basant sur le nombre d'articles disponibles sur l'engin de recherche d'IEEE [23], il est possible de déduire que les produits de Mathworks sont davantage utilisés que la suite Scilab / Scicos. En effet, une recherche sur cet engin des termes « Scilab et Scicos » donne seulement 2 résultats tandis qu'une recherche de « Matlab et Simulink » retourne 601 documents différents. Il est à noter que « Matrixx et SystemBuild » ne retourne aucun document.

La première suite, l'environnement Matlab / Simulink est développé par la compagnie Mathworks et possède un générateur de code nommé Real-Time Workshop (RTW)[45]. Les produits de cette compagnie sont une référence dans le domaine de l'ingénierie: le logiciel Matlab est disponible depuis 1984 et le générateur de code existe depuis 1994 [45]. Le générateur de code possède de nombreuses plateformes cibles sur lesquelles le code peut être exécuté, que ce soit sur des systèmes d'exploitation temps réels que sur des microcontrôleurs. Ce générateur peut être personnalisé via la modification de fichiers en format texte responsables de la génération du code ou de sa compilation.

La suite Matlab / Simulink / RTW est un outil très utilisé en recherche, en éducation et en industrie. Les applications robotiques utilisant le PRC sont très présentes dans la littérature: certains l'utilisent pour concevoir un contrôleur pour un robot parallèle [48], d'autre pour un robot à 6 degrés de liberté avec vision artificielle [49] ou encore pour remettre à jour un contrôleur de robot désuet, en l'occurrence le Puma 560, pour l'utiliser à des fins de recherche [18] ou d'éducation [50]. De façon générale, le PRC est appliqué à des systèmes extrêmement diversifiés (automobile[51], moteurs [52], convoyeur [53], etc.). La compagnie Mathworks présente sur son site Internet [2] plus de 54 différentes organisations (compagnies, centres de recherches, etc.) qui ont utilisés leur suite logicielle avec succès.

La seconde suite, Scilab / Scicos, est un logiciel à code source libre plus récent que la suite de Mathworks: son développement a débuté en 1990 et aucune mention de la génération de code n'a été fait avant 2001 dans la littérature [54]. Cet environnement s'intègre harmonieusement avec RTAI-Lab [55], module qui génère du code destiné à être exécuté sur la plateforme temps réel Linux RTAI. L'acronyme RTAI vient de l'anglais *Real-Time Application Interface*. Il est à noter que le générateur de code est également compatible avec les fichiers Simulink mais que la seule cible disponible est la plateforme temps réel Linux RTAI.

La suite Scilab / Scicos est utilisée comme outil de PRC principalement en recherche, plus particulièrement en Suisse et en Italie. Les applications présentées dans la littérature ne sont pas appliquées à la robotique mais plutôt à des applications simples, par exemple le contrôle d'un moteur [56] ou d'un pendule [57]. Quelques écoles américaines utilisent Scilab dans un but éducatif [58; 59] mais uniquement en simulation logicielle.

La troisième suite, soit Matrix-x / System build, est développé par la compagnie National Instruments et possède un générateur de code nommé Autocode [60] qui peut soit produire du code C ou Ada. Le matériel informatique compatible est limité au matériel supporté par LabVIEW Real-Time [61] ou conforme avec le standard PXI [62].

Par contre, le PRC n'est pas complètement flexible car il ne supporte pas tous les systèmes d'exploitation, ni tout le matériel informatique. Il a été nécessaire dans plusieurs cas de développer des pilotes compatibles avec la plateforme visée [49; 53].

De façon générale, il est difficile d'implanter des mécanismes de contrôle de haut niveau liés à la sécurité des opérations de systèmes réels tels les superviseurs ou chiens de garde (*watchdog*). Le problème avec les outils de PRC réside dans le fait que ces derniers sont davantage conçus pour effectuer du traitement de signal que pour faire de la gestion événementielle.

1.7 Les contrôleurs de robots dans un cadre de recherche appliquée

Les chercheurs ont besoin de plateformes robotisées flexibles pour permettre d'expérimenter de nouvelles avenues. De plus, il arrive fréquemment que ces plateformes soient conçues sur mesure tant au niveau mécanique, électrique et électronique. Sachant cela, il convient d'évaluer les alternatives afin de déterminer lesquelles sont les plus appropriées pour la recherche appliquée.

La première option, l'utilisation du contrôleur propriétaire (voir 1.6.1), ne peut être envisagée que lorsqu'un fabricant est en mesure de fournir un contrôleur compatible avec le système robotique à développer. De plus, comme les chercheurs ont besoin d'une plateforme possédant une grande flexibilité et qu'il n'y a que peu de contrôleurs commerciaux qui permettent d'avoir un contrôle logiciel total, cette solution est à proscrire dans la majorité des cas. Toutefois, s'il arrive qu'un contrôleur de type ouvert soit totalement compatible avec le système robotique et qu'il soit en mesure de répondre au cahier de charges sans aucune modification dans sa conception, cette solution peut être retenue.

La seconde option, qui consiste à développer un contrôleur à partir de zéro à l'aide d'un langage traditionnel (voir 1.6.2), est contre-indiquée pour la recherche car elle entraîne une perte de temps trop importante par rapport aux avantages qu'elle peut apporter en terme de flexibilité.

La troisième option, soit le développement de contrôleur en utilisant un cadre d'application robotique (voir 1.6.3), est très intéressante pour la recherche. Elle permet d'allier la flexibilité de la seconde méthode tout en réduisant drastiquement le temps de développement total. Par contre, en plus des difficultés inhérentes à l'utilisation d'un cadre d'application standard, l'utilisation du CAR demande des connaissances avancées en informatique. Malgré le fait que cette solution soit très viable, elle peut ne pas être la solution idéale pour un spécialiste en contrôle dont la compétence première n'est pas la programmation.

La quatrième option, le contrôleur développé à l'aide d'un environnement de simulation graphique avec un générateur de code (voir 1.6.4), est très attirante pour le chercheur qui ne désire pas se lancer dans une longue et coûteuse activité de programmation car il peut obtenir rapidement des résultats concrets avec le matériel. Cette approche a, par contre, le désavantage d'être moins flexible que la précédente: le matériel supporté peut s'avérer

limité et les mécanismes de supervision, pour la sécurité par exemple, peuvent s'avérer plus difficiles à implanter. De plus, comme le code est généré automatiquement, il peut être difficile de le modifier manuellement, ce qui est parfois requis avec un système robotique utilisé en recherche appliquée.

1.8 Conclusion

À la lumière de ces informations, il est dégagé que la première et la seconde option sont peu appropriées pour la recherche appliquée dans la plupart des cas. Le problème de la première option est son manque de flexibilité tandis que la seconde option demande beaucoup trop de ressources. Les deux dernières options sont de bonnes avenues pour la recherche appliquée et sont utilisées tant par les instituts de recherche que par les universités.

À ce jour, il ne semble pas exister d'approche pour profiter à la fois des avantages du PRC et du CAR. Une approche combinant les deux options pourrait potentiellement accélérer le développement des contrôleurs tout en garantissant à ces derniers un maximum de flexibilité.

Dans le chapitre 2, les deux approches qui semblent appropriées pour la recherche appliquée seront mises en contexte au sein de deux organismes dans lesquels elles sont employées (section 2.2). Cette mise en contexte aboutira à la présentation de la problématique (section 2.3), puis à la proposition d'une solution (sections 2.4 à 2.7) alliant deux approches de développement de contrôleur.

CHAPITRE 2

PROBLÉMATIQUE

2.1 Introduction

Le premier chapitre a permis de mettre en relief les différences entre les types de contrôleurs de robots et de les mettre en contexte avec le domaine de la recherche appliquée en robotique industrielle.

Ce chapitre présente deux orientations empruntées en recherche par deux organismes existants. Suite à cela, les incompatibilités entre ces deux organismes sont soulignées dans le but de trouver une façon de concilier les deux façons de faire. Finalement, une solution est proposée ainsi que des objectifs liés à sa réalisation.

2.2 Description de deux approches

Dans cette section sont présentés deux contextes de recherche appliquée, soit un dans l'univers académique et un autre dans une entreprise de recherche industrielle issue d'une société d'État.

2.2.1 Description d'une approche académique

À l'École de technologie supérieure (ÉTS), plusieurs chercheurs utilisent le prototypage rapide de contrôleur (PRC), c'est-à-dire qu'ils emploient un environnement de simulation graphique avec un générateur de code pour la programmation de leurs contrôleurs robotiques.

Les contrôleurs développés servent à des fins éducatives et pour la recherche appliquée. L'utilisation du PRC facilite le développement de contrôleurs performants, ce qui a pour effet d'agrandir le bassin d'utilisateurs potentiels ainsi que de faciliter l'évolution et la validation des outils.

Cette approche n'est pas uniquement utilisée par l'ÉTS: d'autres institutions d'enseignement utilisent cette approche de développement de contrôleur de robots [48-50].

2.2.2 Description d'une approche industrielle

À l'Institut de recherche d'Hydro-Québec (IREQ), un cadre d'application robotique (CAR) est utilisé pour concevoir des contrôleurs fiables. Le CAR développé par l'IREQ se nomme Microb (voir section 1.6.3).

Ce CAR est utilisé depuis plusieurs années et a permis de standardiser l'aspect logiciel de nombreux projet robotiques, dont la plupart ont été faits sur mesure pour des applications spéciales. Comme les applications pour lesquelles les contrôleurs ont été développés font souvent appel à du matériel très spécialisé, ce type de contrôleur est tout à fait indiqué. De plus, ce milieu est particulièrement propice à l'utilisation d'un CAR puisque de nombreux chercheurs connaissent à la fois la robotique et la programmation orientée objet.

L'IREQ a besoin de quelques chercheurs qui ont développé des contrôleurs avec un langage de programmation traditionnel, car elle veut être leader mondial pour son marché qui est constitué de petites niches où sont utilisés des robots très spécialisés. Pour avoir le maximum de flexibilité, elle a besoin des concepteurs du CAR sur place. Par ailleurs, elle doit posséder un CAR pour développer rapidement de nouvelles

applications. Finalement, l'IREQ se doit d'être en lien avec les chercheurs universitaires pour demeurer un leader mondial.

2.3 Problèmes observés

Actuellement, les méthodes de conception de contrôleurs de l'IREQ et de l'ÉTS ne sont pas compatibles. De plus, l'usage exclusif de l'une de ces deux approches comporte des désavantages importants, exposés dans les lignes qui suivent.

L'utilisation exclusive de Microb est peu avantageuse, particulièrement à cause du niveau des connaissances requis pour l'utiliser, du temps d'apprentissage nécessaire pour le maîtriser et par le fait que son utilisation est peu répandu. Pour utiliser Microb, il faut dans un premier temps avoir des connaissances suffisantes en programmation orientée objet ainsi qu'en systèmes temps réels. De plus, son apprentissage nécessite beaucoup de temps et souvent une formation venant d'un expert Microb. Finalement, comme l'utilisation de Microb est limitée à une poignée de chercheurs, cela limite les possibilités d'amélioration ainsi que les outils disponibles.

Les chercheurs externes à l'IREQ ne sont donc pas enclins à apprendre Microb car en plus d'être difficile à apprendre, c'est un logiciel peu utilisé. Ce problème est actuellement contourné en adjoignant un expert Microb avec un chercheur venant de l'extérieur. Toutefois, cette solution s'avère coûteuse en ressources spécialisées. De plus, comme le chercheur est dépendant, il peut être retardé à cause d'un manque de disponibilité de l'expert Microb, ce qui retarde l'ensemble d'un projet.

L'utilisation exclusive des outils de PRC est peu intéressante dans cette situation. En plus d'une flexibilité réduite, un problème majeur réside dans le fait que de nombreux contrôleurs, intégrant plusieurs aspects de haut niveau tels que la sécurité, sont déjà

développés avec le CAR Microb. Selon l'IREQ, l'option de refaire ces contrôleurs n'est pas techniquement justifiable ni économiquement viable.

De plus, les outils de PRC sont conçus principalement pour faire du traitement de signal plutôt que de la gestion événementielle, ce qui en fait un outil peu adapté au développement de contrôleur de robot. En effet, les différents modes dans lequel peut se trouver un contrôleur de robot dépendent des événements qui se sont produits ou de signaux d'interruption particuliers. De prime abord, il semble que la gestion des différents états du contrôleur à l'aide d'un logiciel de traitement de signal requiert l'utilisation de subterfuges divers tandis que faire la même gestion avec un CAR apparaît plus aisé.

Finalement, de façon générale, les outils de PRC ont une moins grande compatibilité avec le matériel informatique que le CAR. Cela rends donc nécessaire le développement des pilotes logiciels, opération pouvant s'avérer coûteuse en temps ainsi qu'en retard qu'elle peut causer au projet de recherche dans son ensemble.

2.4 Solution proposée

La solution proposée est de combiner les deux méthodes de conception de contrôleur, soit celle utilisant le PRC et l'autre utilisant le CAR, de façon à maximiser leurs avantages respectifs.

Pour éviter qu'il y ait confusion dans les outils utilisés, il est primordial de séparer les mandats de ces deux outils. Pour contourner le problème de compatibilité matérielle, il est préférable que les outils de PRC ne soient pas liés directement au matériel. C'est pourquoi il est proposé que le contrôleur global, développé à l'aide du CAR, soit le centre névralgique du système, responsable de gérer l'exécution des lois de contrôles et d'interagir avec le matériel ainsi qu'avec les usagers, tel qu'illustré à la Figure 8. Des

lois de contrôle générées selon l'approche de PRC pourront s'ajouter à ce contrôleur global.

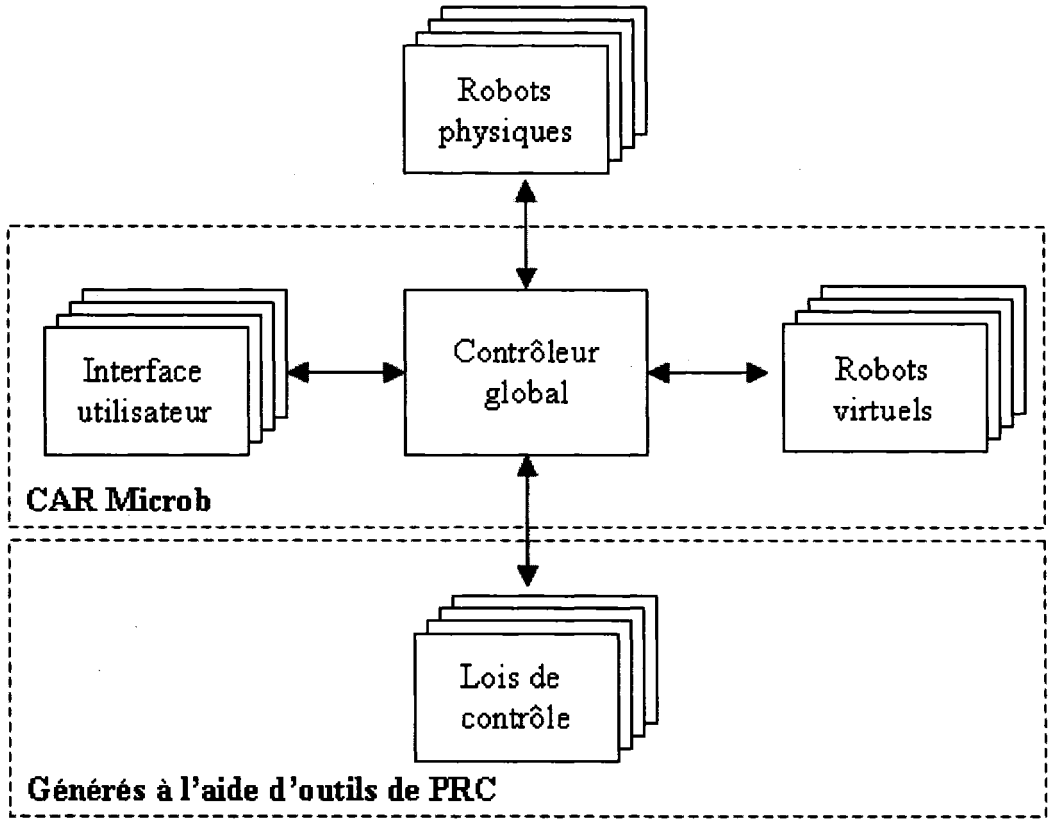

Figure 8 Illustration de la structure de contrôleur proposée

Cette solution maximise l'utilisation des experts de Microb pour les tâches critiques: le contrôleur global doit être configuré pour accueillir les lois de contrôle. Une fois cette opération complétée, les experts en contrôle peuvent contribuer aux contrôleurs de la façon la plus autonome possible sans connaître Microb.

Cette solution peut toutefois nuire à l'évolution de Microb à long terme en limitant son développement. Malgré tout, elle permet d'accroître l'utilisation du CAR à court terme et potentiellement permettre au CAR de se spécialiser.

2.5 Objectif général

Afin de combiner les méthodes de développement de contrôleurs de l'IREQ et de l'ÉTS, une approche logicielle permettant la migration de lois de commande à partir d'un environnement de simulation graphique (PRC) vers un CAR est développée. Cette approche doit autant que possible permettre de bénéficier à la fois des avantages du CAR et de ceux de l'environnement de simulation.

2.6 Objectifs spécifiques

Le but précis est d'élaborer une procédure et les outils informatiques requis permettant l'implantation rapide d'un contrôleur développé sous l'environnement Matlab/Simulink vers un contrôleur temps réel développé avec le CAR Microb conçu par Hydro-Québec.

Un contrôleur à couple précalculé pour un robot virtuel à deux degrés de liberté sera développé sous l'environnement Matlab/Simulink puis implanté selon la procédure élaborée. Le contrôleur devra fonctionner sous le système d'exploitation Windows XP ainsi que sur le système d'exploitation temps réel QNX tout en satisfaisant les besoins des utilisateurs.

2.7 Avantages escomptés

En plus de permettre un rapprochement entre les deux méthodes de conception, il est pressenti que la combinaison de ces deux approches apporte les avantages suivants:

- Permettre d'utiliser les puissants outils de calcul et de modélisation provenant de l'environnement de simulation graphique;
- Apporter une plus grande compatibilité au niveau matériel informatique ;
- Assurer une gestion simple et robuste des superviseurs grâce au CAR;
- Augmenter l'utilisation du CAR.

2.8 Conclusion

Ce chapitre a permis de constater que deux organismes en recherche appliquée avaient des approches différentes et peu compatibles. Une solution a été avancée afin de concilier la conception de contrôleur à l'aide d'un CAR et du PRC. La solution proposée consiste à allier deux méthodes de conception de contrôleur en permettant qu'une loi de contrôle réalisée à l'aide d'un outil de PRC s'adapte à un contrôleur global développé à l'aide d'un CAR.

CHAPITRE 3

MÉTHODOLOGIE DE RECHERCHE

3.1 Introduction

Cette section présente le plan de recherche et le plan de la suite du mémoire. Elle fait état des étapes à franchir pour atteindre les objectifs spécifiques fixés dans le chapitre précédent.

3.2 Étapes de réalisation du projet de recherche

Avant d'élaborer la procédure de migration, il est primordial de connaître quels sont les besoins des futurs usagers pour la solution qui leur est proposée. Au CHAPITRE 4, un sondage (voir ANNEXE 1) est développé à partir de la norme ISO 9126-1 [63]. Ce sondage est soumis à un groupe d'usagers potentiels provenant tant de l'IREQ que de l'ÉTS.

La compilation de ce sondage permet de connaître les attentes des usagers et également de savoir quels sont les éléments à prioriser lors d'une décision de conception. De plus, chaque critère est pondéré de façon à pouvoir faire une évaluation quantitative au cours du projet.

Un produit de migration peut prendre la forme d'une bibliothèque ou d'une application externe. Le CHAPITRE 5 détermine sous quelle forme le produit de migration répond le plus aux besoins des utilisateurs. Ce choix conditionne l'ensemble du développement du protocole de migration.

Pour prendre une décision éclairée, les principaux produits utilisables sont testés. Les tests sont effectués en simulation avec un modèle simple sans utiliser le CAR Microb afin de raccourcir le temps de développement requis. La performance de ces produits est mesurée à partir des critères fixés au CHAPITRE 4. Grâce à ces mesures de performance, il est possible d'évaluer les différents produits afin de déterminer lequel est le plus pertinent à utiliser.

Le choix du produit de migration fait, il est alors possible de développer le protocole de migration au CHAPITRE 6. Dans un premier temps, la séquence de migration est décrite afin de comprendre quelles sont les étapes de développement d'un contrôleur migré. Dans un deuxième temps, l'environnement Matlab est modifié pour permettre la génération du produit désiré. Dans un dernier temps, l'environnement du CAR Microb est adapté afin qu'il puisse utiliser adéquatement le produit de migration.

Le protocole de migration étant développé, son utilisation est documentée au CHAPITRE 7. La documentation comprend un exemple de migration, puis la description de l'installation du protocole et enfin l'utilisation du protocole.

Le CHAPITRE 8 permet d'effectuer l'évaluation du protocole de migration. Cette évaluation est basée sur la migration d'une loi de contrôle d'un environnement de PRC vers un contrôleur conçu à l'aide du CAR. Ce contrôleur, développé avec Microb, fonctionne sous QNX et sous Windows XP.

L'évaluation de la solution finale est effectuée de façon quantitative et qualitative. La partie quantitative est basée sur des mesures de performance fondées sur les besoins des utilisateurs décris au CHAPITRE 4. La portion qualitative consiste à consulter un groupe d'usager qui évalue le protocole en le testant puis en répondant à un questionnaire (voir ANNEXE 2) également basé sur les besoins des usagers décrits au CHAPITRE 4.

3.3 Conclusion

Un plan de réalisation du projet de recherche a été développé et se divise en cinq étapes distinctes, soit:

1. Fixer les objectifs du protocole de migration (CHAPITRE 4);
2. Choisir le produit de migration (CHAPITRE 5);
3. Développer le protocole de migration (CHAPITRE 6);
4. Décrire l'utilisation du protocole de migration (CHAPITRE 7);
5. Évaluer le protocole de migration (CHAPITRE 8).

Après l'exposition de la méthodologie, il convient d'entrer dans le vif du sujet en analysant les besoins des utilisateurs afin d'être en mesure de prendre les meilleures décisions lors de la conception logicielle du protocole de migration.

CHAPITRE 4

ANALYSE DES BESOINS DES UTILISATEURS

4.1 Introduction

Afin de fonder la conception logicielle sur des bases solides, un sondage a été développé pour consulter un groupe d'utilisateurs. Suite à cette consultation, des spécifications logicielles ont été dégagées afin d'orienter le développement de la solution.

4.2 Création d'un sondage

Un sondage a été conçu afin de connaître quels sont les aspects prioritaires à considérer lors de la conception, tant dans celle du processus de migration que dans celle du produit de la migration. Le processus de migration est caractérisé par tous les éléments qui conduisent à la préparation du code migré tandis que le produit de migration est ce qui sera exécuté dans le contrôleur.

Le sondage se divise en deux volets, soit un premier volet plus global concernant les qualités logicielles requises puis un second volet portant sur les attributs quantifiables.

4.3 Qualités et attributs logiciels

Les qualités logicielles, tirées de la norme ISO-9126-1 [63], ont été présentées aux utilisateurs dans le but d'être priorisées. Toutes les qualités de la norme ont été utilisées de façon à ne rien omettre. Cette priorisation des qualités est destinée à faire des choix techniques lors de la conception.

Les attributs associés à ces qualités sont également décrits afin de rendre la priorisation des qualités plus concrète. Les qualités et les attributs contenus dans cette norme sont décrits dans cette section. Cette norme n'est pas disponible en version française. Les définitions sont traduites et adaptées du texte de la norme ISO-9126-1 [63] sauf lorsqu'une autre source est spécifiée.

4.3.1 Fonctionnalité

La fonctionnalité est la qualité logicielle caractérisant un bien ou un service qui répond à sa finalité, et qui est donc fonctionnel et pratique [21]. En bref, c'est la capacité du logiciel à remplir son mandat selon les besoins qui ont été énoncés au départ. La qualité de fonctionnalité est associée aux attributs d'aptitude, de précision, d'interopérabilité et de sécurité.

- Aptitude: capacité d'un logiciel à fournir un ensemble de fonctions afin d'atteindre les objectifs spécifiques des utilisateurs.

- Précision: qualité, état ou degré de conformité par rapport à un standard reconnu ou par rapport à une spécification [21].

- Interopérabilité: capacité que possèdent des systèmes informatiques hétérogènes à fonctionner conjointement, grâce à l'utilisation de langages et de protocoles communs, et à donner accès à leurs ressources de façon réciproque [21].

- Sécurité: fonction qui permet de s'assurer que les services sont employés de façon adéquate par les clients appropriés [21].

La fonctionnalité s'illustre dans notre cas comme étant la capacité du protocole soit d'effectuer une migration réussie ou de générer un produit de migration effectuant les calculs de façon exacte et d'être contrôlé de l'extérieur par le contrôleur global.

4.3.2 Fiabilité

La qualité de fiabilité est la propriété d'un système informatique désignant sa capacité d'assurer ses fonctions sans défaillance, dans des conditions préalablement définies et sur une période déterminée [21]. La qualité de fiabilité est associée aux attributs de maturité, tolérance aux fautes et récupération.

- Maturité: capacité d'un logiciel à éviter les pannes lorsqu'il y a des erreurs dans ce dernier.

- Tolérances aux fautes: capacité d'un logiciel à fonctionner conformément à ses spécifications en cas de défaillance d'un ou de plusieurs de ses constituants [adapté de 21].

- Récupération: capacité d'un logiciel de rétablir un niveau de performance spécifique et de récupérer les données affectées suite à une panne.

La fiabilité s'illustre dans notre cas comme étant la génération automatique d'une loi de contrôle qui ne tombe pas en panne lors de son exécution ou qui permet au contrôleur global de détecter une panne qui survient dans le produit de migration.

4.3.3 Convivialité

La convivialité est la qualité d'un matériel ou d'un logiciel à être facile et agréable à utiliser et à comprendre, même par quelqu'un qui a peu de connaissances en

informatique [21]. La qualité de convivialité est associée aux attributs de facilité de compréhension, de facilité d'apprentissage, de facilité d'opération et de pouvoir d'attraction.

- Facilité de compréhension: capacité d'un logiciel à permettre à l'usager de saisir à quoi il sert et quelles sont ses conditions d'utilisation.

- Facilité d'apprentissage: qualifie un matériel ou un logiciel dont l'apprentissage est agréable et facile, même pour quelqu'un qui a peu de connaissances en informatique [21].

- Facilité d'opération: capacité d'un logiciel à permettre à l'usager de l'opérer et d'en avoir le contrôle.

- Pouvoir d'attraction: capacité d'un logiciel d'être attirant pour l'utilisateur.

Par exemple, la convivialité dans notre contexte est l'intégration du protocole à l'environnement de développement, le temps d'apprentissage requis pour utiliser le protocole de migration à pleine capacité, le temps requis pour effectuer une migration et le nombre d'itérations nécessaires pour y parvenir.

4.3.4 Efficacité

La qualité d'efficacité représente la capacité du logiciel à performer adéquatement relativement aux ressources disponibles selon certaines conditions [21]. La qualité d'efficacité est associée aux attributs de temps de réponse par cycle et d'utilisation des ressources.

- Temps de réponse: capacité d'un logiciel à fournir une réponse appropriée en un temps voulu lorsqu'il remplit sa fonction dans des conditions données.

- Utilisation des ressources: capacité du logiciel à utiliser une quantité et un type de ressources appropriées lorsque ce dernier est exécuté dans des conditions données.

L'efficacité est dans notre cas le temps réponse pour un cycle de calcul, ou encore la mémoire vive requise durant l'exécution et l'espace disque qu'occupe le produit de migration.

4.3.5 Maintenabilité

La qualité de maintenabilité est la capacité du logiciel à être modifié. Les modifications peuvent inclure les corrections, les améliorations, ou l'adaptation d'un logiciel aux changements dans son environnement ou dans ses spécifications. La qualité de maintenabilité est associée aux attributs de facilité d'analyse, de facilité de modification, de stabilité et de facilité de test.

- Facilité d'analyse: capacité d'un logiciel à être diagnostiqué pour des défauts ou des pannes.

- Facilité de modification: capacité d'un logiciel à permettre une modification donnée.

- Stabilité: capacité d'un logiciel à éviter les effets inattendus suite à une modification au logiciel.

- Facilité de test: capacité d'un logiciel à être valider lorsque des modifications sont effectuées.

La maintenabilité s'illustre dans notre cas comme étant la capacité à effectuer des modifications dans le produit de migration ainsi que de permettre de tester les fonctionnalités et la stabilité d'un produit de migration avant son utilisation.

4.3.6 Portabilité

La qualité de portabilité est la capacité d'un logiciel à être transporté d'un environnement à un autre. La qualité de Portabilité est associée aux attributs de facilité d'adaptation, de facilité d'installation, de coexistance et d'interchangeabilité.

- facilité d'adaptation: capacité d'un logiciel à être adapté pour différents environnements sans appliquer des actions ou des moyens autres que ceux fournis à cette fin pour le logiciel.

- facilité d'installation: capacité d'un logiciel à être installé dans un environnement spécifique.

- Coexistence: capacité d'un logiciel à coexister avec un autre logiciel indépendant dans un environnement commun partageant les mêmes ressources.

- Interchangeabilité: capacité d'un logiciel à être utilisé à la place d'un autre pour la même application dans le même environnement.

La portabilité dans notre cas est la capacité d'un produit de migration à être exécuté et installé sur plusieurs systèmes d'exploitation (Windows, QNX, etc.) tout en s'intégrant au contrôleur global développé avec le CAR Microb.

4.4 Objectifs quantifiables

Les qualités et attributs logiciels décrits précédemment permettent de définir l'orientation globale du logiciel à concevoir. Toutefois, elles restent peu précises et leur évaluation reste floue. L'utilité de fixer des objectifs quantifiables permet de préciser le résultat plus aisément. La ligne directrice de ces objectifs a été initialement proposée par l'auteur lors de l'élaboration d'un sondage réalisé sur un groupe d'utilisateurs potentiels du logiciel. Ce sondage a permis d'ajouter de nouveaux objectifs quantifiables (voir ANNEXE 1).

Il est prévisible que l'utilisation du protocole de migration conduira à une perte de performance mesurable, telle l'augmentation du temps de cycle, de la mémoire vive requise ou de l'espace disque utilisée. Une perte de performance aux alentours de 10% est considérée comme acceptable dans la littérature [51]. Par exemple, une augmentation du temps de cycle de 10% pour le contrôleur global à cause de l'utilisation du protocole de migration est admissible.

Le sondage réalisé, dont les résultats sont présentés plus bas, a également permis de prioriser certains objectifs:

- Minimiser le temps de cycle lors de l'exécution du code généré;
- Minimiser la mémoire vive utilisée lors de l'exécution du code généré (RAM);
- Minimiser l'espace disque utilisé pour le code généré (ROM);
- Minimiser l'utilisation du processeur pour la génération du code (CPU);
- Minimiser le temps de migration total.

En plus de la priorisation des objectifs quantifiables, quelques questions ouvertes ont été soulevées lors de ce sondage. Ces questions sont décrites à la section 4.4.6 tandis que les objectifs priorisés sont décrits dans les paragraphes suivants.

4.4.1 Minimiser le temps de cycle

Temps d'exécution moyen pour un cycle du système après un grand nombre de cycles d'exécution. Un court temps d'exécution est souhaitable car certains calculs doivent être faits à très haute fréquence.

4.4.2 Minimiser la mémoire vive utilisée (RAM)

La mémoire vive occupée par le code généré doit être considérée car il peut s'agir d'une ressource critique au sein d'un contrôleur embarqué.

4.4.3 Minimiser l'espace disque utilisé (ROM)

L'espace disque requis par le code généré doit être considéré car il peut s'agir d'une denrée rare au sein d'un contrôleur embarqué.

4.4.4 Minimiser l'utilisation du processeur (CPU)

Minimiser l'utilisation du processeur lors de la génération du code pour permettre l'utilisation du protocole de migration sur une machine peu performante.

4.4.5 Minimiser le temps de migration total

L'opération de migration, c'est-à-dire du moment de la génération du code jusqu'à son exécution, doit être d'une durée la plus courte possible. Le facteur qui permet de réduire le temps de migration est directement lié au fait de devoir ou non recompiler le contrôleur global. Cette action peut demander plusieurs minutes, tout dépendant de la dimension de ce dernier. Il serait souhaitable de ne pas avoir à recompiler à chaque modification du code pour que l'utilisation du protocole soit la plus rapide possible.

4.4.6 Autres objectifs quantifiables

Deux questions ouvertes sont posées afin d'aller chercher des balises plus précises pour le protocole de migration. Le but de la première questions est de chiffrer le temps de migration total maximal, c'est-à-dire de Simulink jusqu'à l'exécution du code sur le contrôleur physique. La seconde question avait pour objectif de connaître le temps d'apprentissage maximal acceptable pour le protocole, c'est-à-dire le temps requis pour installer, comprendre et utiliser pleinement ce dernier. De plus, une rubrique supplémentaire appelée « autres considérations » a été ajoutée de façon à ce que les gens sondés puissent ajouter certains éléments qui auraient pu être omis (voir ANNEXE 1).

4.5 Consultation d'un groupe d'usagers

La consultation des usagers potentiels du protocole de migration a été réalisée à l'IREQ le 25 octobre 2005. La session a été animée par l'auteur du présent mémoire. Huit personnes étaient présentes lors de la consultation et provenaient soit de l'ÉTS ou de l'IREQ. Les personnes choisies pour effectuer la consultation avaient des connaissances pratiques en programmation de contrôleur de robot, en contrôle de robots et avaient déjà utilisé soit un outil de PRC ou un CAR ou les deux.

Une présentation globale du projet a été faite afin d'introduire les participants au contexte du projet de recherche. Un questionnaire (voir annexe 1) a été distribué à chaque participant pour leur permettre d'inscrire leurs réponses et également de leur rappeler la définition des qualités et attributs logiciels. Le questionnaire a été complété dans un premier temps sur une base individuelle puis les réponses obtenues ont été discutées en groupe. Le groupe tentait dans un premier temps de décider de façon consensuelle la priorisation finale. Lorsqu'un consensus n'était pas possible, l'assemblée passait au vote et, en cas d'égalité, une personne désignée tranchait sur la décision finale.

4.6 Résultats de la consultation

Suite à cette consultation, la priorisation des qualités logicielles a été effectuée et est présentée dans cette section.

4.6.1 Priorités des qualités du protocole de migration

Les qualités logicielles pour le protocole de migration ont été priorisées de la façon suivante, du plus important au moins important:

1. fonctionnalité;
2. convivialité;
3. fiabilité;
4. maintenabilité;
5. efficacité;
6. portabilité.

4.6.2 Priorités des qualités du produit de migration

Les spécifications logicielles du produit de migration ont été priorisées de la façon suivante, du plus important au moins important:

1. fiabilité;
2. fonctionnalité;
3. efficacité;
4. portabilité;
5. convivialité;
6. maintenabilité.

4.6.3 Priorité des objectifs quantifiables

Les objectifs ont été priorisées de la façon suivante, du plus important au moins important:

1. Minimiser le temps de cycle lors de l'exécution du code généré;
2. Minimiser le temps de migration total;
3. Minimiser la mémoire vive utilisée lors de l'exécution du code généré (RAM);
4. Minimiser l'espace disque utilisé pour le code généré (ROM);
5. Minimiser l'utilisation du processeur pour la génération du code (CPU).

Lors du sondage, il a été également mentionné que le cinquième objectif, soit l'utilisation du processeur lors de la génération du code, était peu important parce que les plateformes de PRC sont généralement pourvues d'une grande puissance de calcul.

Les questions ouvertes ont permis de déterminer que la migration devait durer environ une minute et que l'apprentissage du protocole devrait s'étendre de 2h à une journée. De plus, les usagers ont demandé que le code généré soit une boîte noire et qu'il ne soit pas nécessaire d'aller le modifier.

4.7 Conclusion

Ce chapitre a permis d'exposer la démarche qui a mené à l'obtention des priorités en terme de qualités logicielles pour le produit et le protocole de migration. Une étude réalisée auprès d'utilisateurs potentiels a permis de fixer ces priorités. De plus, des objectifs quantifiables ont été priorisés lors de cette étude de façon à effectuer des choix technologiques sans ambiguïté pour satisfaire les besoins des utilisateurs. Les réponses obtenues dans ce chapitre permettront d'effectuer les choix du produit de migration, qui fait l'objet du prochain chapitre.

CHAPITRE 5

PRODUIT DE MIGRATION

5.1 Introduction

Avant de développer le protocole de migration du PRC Simulink / RTW au CAR Microb, il est primordial de déterminer quelle forme aura le produit de migration. Les objectifs de conception ont été fixés au dernier chapitre et seront utilisés pour faire les choix technologiques pour le produit de migration. Suite au présent chapitre, il sera possible de développer le protocole de migration, c'est-à-dire la génération automatique de code et son intégration dans le contrôleur global préalablement développé à l'aide d'un CAR.

5.2 Méthodologie pour choisir le produit de migration

Le but de ce chapitre est de déterminer de façon méthodique quel est le moyen technologique le plus approprié pour interfacer le code généré, c'est-à-dire le produit de migration, avec le contrôleur global. La démarche pour choisir la technologie du produit de migration la plus adéquate est la suivante:

1. Choisir le modèle à simuler;
2. Déterminer les produits de migration disponibles;
3. Coder le modèle et les produits de migration;
4. Mesurer la performance de chaque produit de migration;
5. Analyser les performances obtenues;
6. Choisir le produit de migration d'après les critères quantifiables fixés (voir section 4.6.3).

5.3 Produits de migration disponibles

Les alternatives étudiées dans ce document sont décrites plus bas avec les avantages et inconvénients escomptés a priori.

5.3.1 Codage manuel

Ajouter le code dans le contrôleur global manuellement. Temps d'exécution optimal mais plus laborieux pour extraire le code car ce dernier possède une structure complexe. Recompilation du contrôleur global obligatoire car le code ajouté en fait partie.

5.3.2 Bibliothèque statique

Encapsuler le code généré par le PRC dans une bibliothèque statique (.lib). Ralentissement négligeable escompté. Compilation obligatoire de la bibliothèque statique et du contrôleur global pour tout changement.

5.3.3 Bibliothèque dynamique

Encapsuler le code généré par le PRC dans une bibliothèque dynamique (.dll). Ralentissement négligeable et pas de recompilation du contrôleur global requis.

5.3.4 Mémoire partagée

Programme esclave (généré par le PRC) qui utilise la mémoire partagée et des signaux pour communiquer avec le programme maître (développé à l'aide du CAR). Des performances moyennes sont attendues à cause de la grande interaction avec les fonctions du système d'exploitation.

5.3.5 Tuyau (pipe)

Programme esclave qui utilise un tuyau pour communiquer avec le programme maître. Des performances moyennes sont prévues à cause de la grande interaction avec les fonctions du système d'exploitation.

5.3.6 Lien TCP/IP

Programme esclave qui utilise un lien TCP/IP pour communiquer avec le programme maître. De piètres performances sont attendues à cause de la latence du protocole réseau.

5.3.7 Redirection

Programme esclave exécuté à chaque appel de fonction. D'exécrables performances sont prévisibles à cause du temps de chargement de l'application esclave par le système d'exploitation.

5.4 Mesure de performance des produits de migration

Les objectifs ont été choisis et pondérés suite à une consultation effectuée auprès d'usagers potentiels qui travaillent en robotique à l'IREQ et à l'ÉTS (voir CHAPITRE 4). La liste initiale des objectifs quantifiables priorisés est présentée à la section 4.6.3.

Ces objectifs ont tous été considérés dans le cadre de cette étude sauf le dernier intitulé «Minimiser l'utilisation du processeur pour la génération du code (CPU) », car il n'est pas possible de prévoir l'utilisation du processeur pour la génération du code étant donné qu'il n'y a pas de génération automatique à ce moment. De plus, le groupe d'usagers s'accorde pour dire que cet objectif est peu pertinent.

5.5 Pondération des mesures de performance

La pondération des objectifs a été basée sur leur priorité. Les pondérations données aux objectifs choisis sont présentées dans le Tableau II.

Tableau II

Priorités et pondération des objectifs choisis

Objectif	Priorité	Pondération
Minimiser le temps de cycle	1	40%
Minimiser le temps de migration	2	30%
Minimiser la mémoire vive utilisée	3	20%
Minimiser l'espace disque utilisé	4	10%
Total	*10*	*100%*

La formule qui a été utilisée pour calculer le pourcentage de pondération pour chaque objectif est la suivante:

$$pondération = \frac{\left((nb.\,objectifs + 1) - indice\ de\ priorité\right)}{\sum indice\ de\ priorité}$$

$$ex.:\ pondération_{temps\ de\ cycle} = \frac{\left((4+1)-1\right)}{10} = 40\%$$

Les indices de priorité et la pondération sont présentés dans le Tableau II. Les objectifs concernant le temps de cycle et la mémoire utilisée (RAM et ROM) sont évalués en fonction du pointage maximal obtenu.

Voici la formule choisie pour calculer le pourcentage de ces trois objectifs:

$$Pointage = \left(1-\left(\frac{x-1}{\max-1}\right)\right)\%$$

Ex.: *Pour l'approche avec tuyau, l'évaluation du temps de cycle est:*

$x = 51$ *fois le temps de cycle de l'approche manuelle*

$\max = 181$ *fois le temps de cycle de l'approche manuelle*

$$Pointage_{\text{temps de cycle tuyau}} = 1-\left(\frac{51-1}{181-1}\right)=72{,}2\%$$

*Le temps de cycle a un poids de 40% sur le pointage final de l'approche (voir Tableau II). Cela signifie donc que cet objectif aura un poids résultant de 28.88 points (72,2% * 40%) sur le pointage final de l'approche du tuyau.*

5.6 Procédure d'évaluation des performances

Pour déterminer l'approche la plus appropriée, un simulateur de piston a été utilisé. Ce simulateur a été choisi car c'est un système simple à modéliser ainsi qu'à comprendre et c'est un exemple existant dans la documentation de la bibliothèque du CAR Microb [64, p. 30-41].

Le temps de cycle, l'utilisation de la mémoire vive et de l'espace disque sont évaluées par rapport à l'approche manuelle. Si par exemple la bibliothèque dynamique affiche un résultat de 1.2 pour l'utilisation de mémoire vive, cela signifie que cette technologie utilise 20% plus de mémoire vive par rapport à l'approche manuelle.

L'objectif de minimisation de temps de migration, étant directement lié à la recompilation du contrôleur global, est binaire: si aucune recompilation du contrôleur global est nécessaire, l'approche obtiendra 100%. Dans le cas contraire, 0% sera accordé pour cet objectif.

5.7 Analyse des performances des produits de migration

Voici un tableau résumant les observations qui ont été faites lors de l'exécution des programmes. Les mesures ont été prises après 100 millions de cycles.

Tableau III

Résultats des essais en vue de choisir le produit de migration

Approche	*Perfo. Temps cycle*	*pts*	*Perfo. Mémoire vive*	*pts*	*Perfo. Espace disque*	*pts*	*Perfo. Compi-lation*	*pts*	*Total*
Manuel	**1**	40	**1**	20	**1**	10	oui	0	70%
Bibliothèque Statique	**1,2**	40	**1**	20	**1**	10	oui	0	70%
Bibliothèque dynamique	**1,3**	40	**1,2**	19	2,2	0	**non**	30	**89%**
Mémoire partagée	21	40	2,5	11.8	2	1.7	**non**	30	84%
Tuyau	51	40	2,5	11.8	2	1.7	**non**	30	83%
Lien TCP/IP	181	39.6	4,7	0	2	1.7	**non**	30	71%
Redirection	14 377	0	2,4	12.4	2	1.7	**non**	30	44%

(note: les résultats en **gras** *sont les plus intéressants)*

La redirection a un très mauvais temps par cycle à cause du délai de chargement de l'application. Le temps obtenu est disproportionné par rapport aux autres produits. La

redirection a donc été exclue afin de ne pas fausser les résultats. Le tableau suivant est alors obtenu:

Tableau IV

Résultats des essais en vue de choisir le produit de migration (corrigé)

Approche	*Perfo. Temps cycle*	*pts*	*Perfo. Mémoire vive*	*pts*	*Perfo. Espace disque*	*pts*	*Perfo. Compi-lation*	*pts*	*Total*
Manuel	**1**	40	**1**	20	**1**	10	oui	0	70%
Bibliothèque Statique	**1,2**	40	**1**	20	**1**	10	oui	0	70%
Bibliothèque dynamique	**1,3**	40	**1,2**	19	2,2	0	**non**	30	**89%**
Mémoire partagée	21	35.6	2,5	11.8	2	1.7	**non**	30	79%
Tuyau	51	28.8	2,5	11.8	2	1.7	**non**	30	72%
Lien TCP/IP	181	0	4,7	0	2	1.7	**non**	30	32%

(note: les résultats en **gras** *sont les plus intéressants)*

Après un bref regard sur les temps de cycle (voir Tableau IV), il est aisé de constater qu'une approche par bibliothèque est beaucoup plus rapide qu'une application externe car elle ne nécessite pas de communication entre les procédés (IPC). Le délai supplémentaire entre la communication par mémoire partagée et celle par tuyau s'explique par le fait que les demandes d'accès sont gérées par un serveur du système d'exploitation. Cela n'est pas le cas pour la mémoire partagée car son accès n'est pas contrôlé. Quant au lien TCP/IP, il offre un temps par cycle assez long, vraisemblablement à cause de la lenteur du protocole.

L'utilisation de l'espace disque (voir Tableau IV) dépend de l'utilisation de une ou deux portions de code précompilé. Lorsqu'il y a deux programmes, cette quantité est multipliée environ par deux. Il est à noter que l'utilisation d'une bibliothèque dynamique

(.dll) augmente légèrement le facteur multiplicateur, probablement à cause de l'ajout de code permettant d'interfacer les fonctions de cette dernière.

L'utilisation de la mémoire vive (voir Tableau IV) augmente également avec le nombre de programmes. Lorsqu'un seul programme est en exécution, entre 652 Ko et 804 Ko de mémoire vive sont occupés, tandis que l'utilisation de deux programmes provoque une utilisation beaucoup plus grande: le double (mémoire partagée, tuyau, redirection) ou le quadruple (Lien TCP/IP).

La recompilation du contrôleur global est nécessaire uniquement lorsque l'on utilise l'approche manuelle ou celle par bibliothèque statique (voir Tableau IV). Dans les autres cas, il est possible de remplacer les anciens fichiers de la loi de contrôle générée par les nouveaux, qu'ils soient exécutables (.exe) ou non (.dll).

Toutefois, il est à noter que lorsque le lien avec le contrôleur global change (entrées, sorties, appel de fonction, etc.), il devra être recompilé au complet, et ce, pour toutes les approches.

Les approches ayant un plus court temps de migration sont celles qui ne nécessitent pas de recompiler le contrôleur global. Les autres étapes, pouvant être automatisées, ne devraient pas augmenter le temps total de migration.

De plus, comme une loi de contrôle est souvent un élément critique d'un procédé, une éventuelle défaillance de cette dernière doit être détectée à l'aide d'une horloge de surveillance (*watchdog*). Si les lois de contrôle ajoutées ne font pas partie du contrôleur global, une horloge de surveillance doit être ajoutée pour chacune d'elles. Cela résulte en une plus grande utilisation du processeur et augmente le risque d'erreur de programmation à cause de la complexité additionnelle requise. Dans le cas d'un exécutable, il devient nécessaire de le gérer en fonction des états du système et de

détecter une éventuelle panne. Dans le cas des bibliothèques et de l'intégration manuelle, aucun problème de ce type n'est possible: la même horloge de surveillance pourra être employée tant pour le contrôleur global que pour les lois de contrôle ajoutées.

5.8 Choix technologique du produit de migration

Voici les résultats finaux suite à l'expérimentation des différentes approches. L'ajout du code manuel peut être exclu *a priori* car cela ne permet pas d'atteindre un temps de migration raisonnable, dû au fait qu'il faut recompiler le contrôleur global. Les deux approches les plus prometteuses sont donc celles utilisant les bibliothèques (voir Tableau IV).

La bibliothèque dynamique peut être utilisée lors de la phase de développement et de test car son temps de migration est court. La bibliothèque statique pourrait être utilisée lors de l'implémentation de la version finale étant donné qu'elle occupe moins d'espace disque et de mémoire vive tout en étant légèrement plus rapide.

Comme la différence est relativement négligeable entre les deux bibliothèques et que les avantages pour l'utilisation d'une bibliothèque dynamique (*dll*) sont indéniables, il est recommandé de développer un protocole pour cette approche en premier lieu. Dans le futur, il pourrait être intéressant d'adapter le protocole pour la bibliothèque statique (*lib*) si certaines applications requièrent des temps de cycle encore plus courts, mais cela ne fera pas l'objet du présent mémoire.

5.9 Conclusion

L'évaluation des principaux produits de migration disponibles a été réalisée dans ce chapitre. Après avoir pondéré ces priorités et pris des mesures quant à la performance de

chaque produit, l'utilisation d'une bibliothèque partagée a été sélectionnée comme étant la plus avantageuse. C'est donc ce produit que devra utiliser le processus de migration, tant dans la génération de code que dans l'intégration au contrôleur global développé avec le CAR Microb. Le développement du processus permettant la génération et l'utilisation de la bibliothèque dynamique fait l'objet du prochain chapitre.

CHAPITRE 6

DÉVELOPPEMENT DU PROCESSUS DE MIGRATION

6.1 Introduction

Ce chapitre décrit le fonctionnement du processus de migration. Il tient compte des chapitres précédents qui ont permis de fixer les qualités logicielles prioritaires et de choisir la bibliothèque dynamique pour encapsuler la loi de commande élaborée à l'aide du PRC. Dans cette section, le choix des logiciels, la séquence de migration ainsi que la stratégie d'implémentation du processus sont présentés.

6.2 Logiciels utilisés

Le choix des logiciels est requis pour débuter le développement du protocole de migration. Le processus de migration doit être développé dans un environnement de simulation donné et doit permettre de s'adapter à un CAR particulier. Le choix de cet environnement de simulation et du CAR utilisé est expliqué dans les lignes qui suivent.

Quelques environnements de simulation existent et les trois principaux qui possèdent un générateur de code sont Matlab / Simulink, SCILAB / SCICOS et NI MATRIXx / SystemBuild (voir section 1.6.4). L'environnement PRC sélectionné pour développer le processus de migration est Matlab / Simulink muni du générateur de code RTW. Quelques raisons motivent ce choix. Premièrement, l'IREQ utilise déjà les outils de Mathworks dans plusieurs projets. Deuxièmement, les plateformes cibles supportées sont plus nombreuses. Troisièmement, cette suite d'outil semble avoir une crédibilité plus grande car elle existe depuis plus longtemps et qu'elle est supportée par une compagnie reconnue. Il est à noter que les outils Mathworks sont également abondamment utilisés dans les milieux académiques tel que l'ÉTS.

Le CAR choisi est celui présentement utilisés par Hydro-Québec, soit Microb (Module intégré de Contrôle Robotique). Il n'a pas été comparé en profondeur aux autres cadres d'application liés au domaine robotique (voir section 1.6.3) car le partenaire industriel désire utiliser le CAR qu'il a développé et qu'il utilise depuis plusieurs années sur différents projets robotiques.

Les deux outils logiciels choisis sont donc le CAR Microb d'Hydro-Québec et la suite Matlab / Simulink / RTW développée par Mathworks. À partir de ce choix technologique, il est maintenant possible de décrire le déroulement de la séquence de migration.

6.3 Description de la séquence de migration

La migration consiste à utiliser la plateforme de simulation Simulink pour générer le code exécutable qui sera appelé par le contrôleur global sur la plateforme d'exécution.

L'activité de migration est unidirectionnelle et se divise en trois étapes (voir Figure 9). Les deux premières étapes sont exécutées sur la plateforme de simulation tandis que la dernière étape est destinée à être réalisée sur la plateforme d'exécution, c'est-à-dire celle sur laquelle est exécuté le contrôleur. Il est à noter que la plateforme de simulation et d'exécution peuvent être sur le même ordinateur.

La première partie consiste à générer le code C du modèle. L'exécution de cette action est assurée par la chaîne d'outils Matlab / Simulink / RTW. La seconde partie, l'encapsulation en une bibliothèque dynamique, est effectuée avec la même chaîne d'outil mais avec une configuration personnalisée. La troisième partie consiste à compiler le code destiné à être utilisé par le contrôleur global.

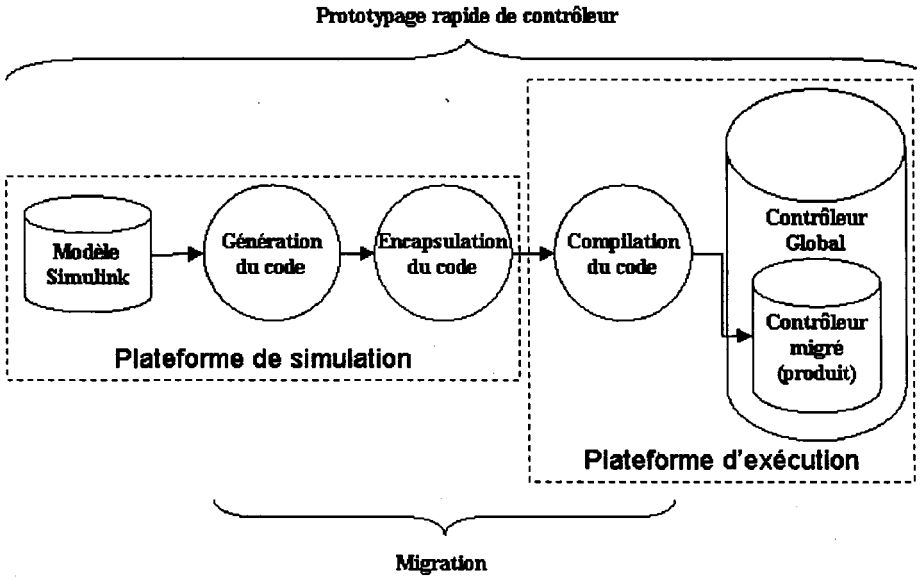

Figure 9 Illustration du processus de migration

La génération de code à partir d'un modèle Simulink s'effectue grâce à RTW. Ce mécanisme est illustré dans la Figure 10. La boîte à outils, en se basant sur les informations du modèle, génère un fichier contenant l'ensemble des boîtes à générer. Ce fichier a pour extension *rtw* (ex: *model.rtw*). À partir de ce fichier, plusieurs fichiers contenant du code C (ex: *model.c, model.h, etc.*) sont générés en utilisant les fichiers *Target Language Compiler* (TLC) associé à chacun des blocs utilisés dans le modèle. Ces fichiers définissent ce qui doit être fait à l'intérieur du bloc et RTW s'assure que les signaux sont bien acheminés à chaque bloc pour être traités. Finalement, un fichier de compilation (makefile) est généré (ex: *model.mk*) à partir d'un fichier *Template Make File* (TMF) propre à la cible choisie. L'exécution du fichier de compilation par la commande *make* crée un exécutable indépendant de Matlab.

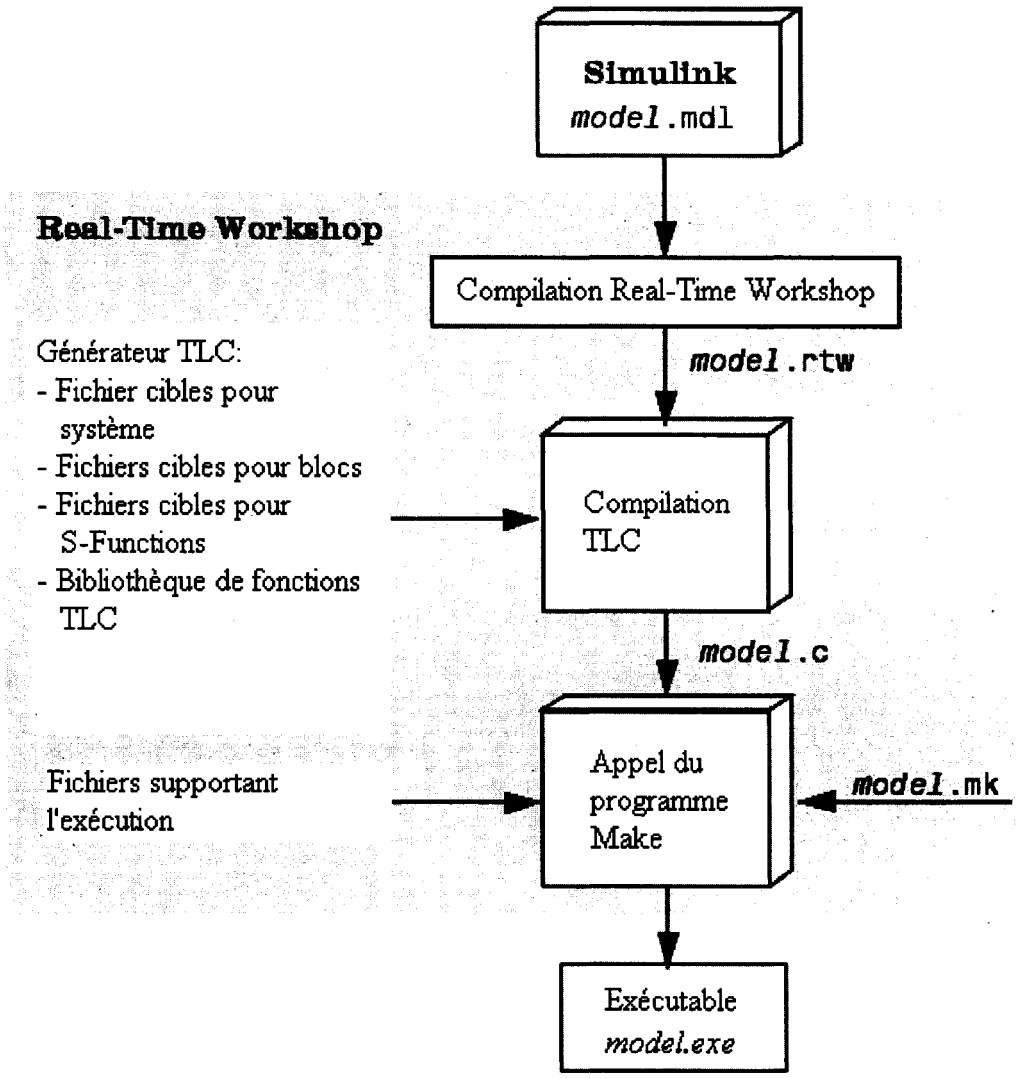

Figure 10 Mécanisme de génération de RTW

Pour la compilation du code [45], il existe deux cas de figure: soit que le contrôleur est destiné à fonctionner sur l'ordinateur de simulation ou qu'il doit être exécuté sur un autre ordinateur.

Dans le premier cas, lorsque le tout est exécuté sur un seul ordinateur, la chaîne d'outils Matlab / Simulink / RTW se charge de compiler la bibliothèque dynamique. Lorsque le

code est compilé, l'usager n'a qu'à copier la bibliothèque dans le répertoire où elle doit être utilisée.

Dans le second cas où les plateformes d'exécution et de simulation sont distinctes, il y a quatre possibilités pour accomplir la dernière étape:

1. Manuel: copier le code généré de la plateforme de simulation vers la plateforme d'exécution et lancer la compilation manuellement. Cette solution est la plus simple à développer en plus de limiter les risques d'erreurs mais c'est également celle qui demande le plus de temps à l'usager;
2. Programme résident: un processus s'exécutant en permanence sur la machine cible s'assure que la bibliothèque est toujours à jour en vérifiant la version du code. Simple à l'usage mais obligation de développer et d'installer un exécutable supplémentaire, qui peut potentiellement être une source d'erreurs;
3. Exécution distante: utiliser un protocole de transfert et d'exécution pouvant lancer la compilation (ssh, telnet, rlogin, etc.). Solution intéressante mais risquée au niveau de la sécurité selon le protocole utilisé (intrusion possible);
4. Compilation croisée: compiler sur la plateforme de simulation le code destiné à la plateforme d'exécution à l'aide d'un compilateur compatible avec la plateforme d'exécution. C'est une solution moins intéressante car ces compilateurs ont plus de risques de générer du code instable qu'un compilateur natif.

Comme ce projet de maîtrise a principalement pour but de prouver la fonctionnalité du concept et qu'il se peut que l'usager préfère utiliser sa propre méthode, le choix de la méthode manuelle a été fait. De plus, le temps supplémentaire requis pour la méthode manuelle reste somme toute raisonnable.

6.4 Modification à l'environnement de simulation

Pour fonctionner dans l'environnement Matlab, certaines modifications dans les règles de génération et dans la compilation du code doivent être effectuées. Cette section présente un aperçu de ce qui doit être fait sur la plateforme de simulation; le détail des opérations à effectuer sur la plateforme de simulation est présenté au CHAPITRE 7.

6.4.1 Modification à la génération du code

Le code généré par RTW est initialement destiné à produire un exécutable dont les fonctions d'interactions avec d'autres logiciels sont limités voire même inexistantes. Il doit donc être converti afin de permettre de contrôler son exécution à partir d'un programme principal. Cette conversion est effectuée en modifiant le code généré à l'aide d'une procédure suggérée par Roland Pfeiffer [65].

Cette procédure se résume à ajouter trois fonctions: la première pour commander l'initialisation, la seconde pour procéder aux calculs et la troisième pour terminer l'exécution de la bibliothèque dynamique. Le code résultant est recompilé sous forme de bibliothèque dynamique, prêt à être utilisé par le contrôleur global.

Les étapes suivantes sont effectuées par le contrôleur global selon la phase dans lequel il se trouve:

- Phase initiale
  - Chargement en mémoire de la bibliothèque dynamique[1]
  - Initialisation des pointeurs sur les 3 fonctions disponibles[1]
  - Exécution de la fonction d'initialisation

[1] Étape nécessaire sous Windows seulement, uniquement dans le cas où la bibliothèque est chargée durant l'exécution.

- Phase exécution
    - Exécution de la fonction de calcul (*n* fois)

- Phase finale
    - Exécution de la fonction de terminaison

La procédure proposée par monsieur Pfeiffer [65] est manuelle et requiert plusieurs manipulations pouvant induire des erreurs. Ces erreurs peuvent empêcher la compilation ou encore une exécution fiable. Pour éviter cet effet indésirable, la mécanique de génération de code a été modifiée pour rendre la procédure entièrement automatique.

L'automatisation de cette procédure passe par l'utilisation du langage TLC de Mathworks qui permet de modifier la génération du code. Les fichiers TLC permettent de générer le code propre au modèle en se basant sur les informations contenues dans le fichier RTW (voir Figure 10). En modifiant ou en ajoutant des fichiers TLC, il est possible de générer un fichier qui permettra de créer une bibliothèque dynamique dont l'exécution sera dirigée par une application externe. Le détail des changements effectués est disponible à l'ANNEXE 3 pour Windows et à l'ANNEXE 5 pour QNX.

6.4.2 Modification à la compilation du code

La compilation du code, initialement destinée à créer un exécutable indépendant, doit être modifiée pour indiquer au compilateur de créer une bibliothèque dynamique. Le fichier TMF associé à la cible est responsable de générer le makefile propre au modèle. En modifiant ce fichier, il est possible de changer les règles de compilation pour obtenir une bibliothèque dynamique. Le détail des changements effectués est disponible à l'ANNEXE 4 pour Windows et à l'ANNEXE 6 pour QNX.

6.4.3 Encapsulation des changements

Pour encapsuler les changements dans les fichiers TLC, une nouvelle cible (*target*) a été créée selon le système d'exploitation visé (Windows, QNX, etc.) pour tenir compte des différences entre ces plateformes. Cette cible doit être simplement ajoutée à l'arborescence de la suite de Mathworks pour être aussitôt accessible via l'interface de Matlab.

Sous Windows, les fichiers *Template Makefile* (*grt.tmf*) et *Target Language Compiler* (*grt.tlc*) de la cible *Generic Real-Time* (*grt*) ont été modifiés. Les deux fichiers (voir ANNEXE 3 et ANNEXE 4) ont été encapsulés dans une nouvelle cible RTW pour générer une bibliothèque dynamique (*.dll*) sous Windows et cette cible a été appelée *win_dll.*

Sous QNX, les deux mêmes fichiers ont été modifiés mais au lieu d'être basée sur la cible *Generic Real-Time* (*grt*), la cible a été basée sur une cible développée expressément pour QNX par Robert F.K. Martin [66]. Ce dernier a développé cette cible afin de faire fonctionner un robot Puma 560 à l'aide d'un contrôleur fonctionnant sous QNX. Les deux fichiers (voir ANNEXE 5 et ANNEXE 6) ont été encapsulés dans une nouvelle cible RTW pour générer une bibliothèque partagée (*.so*) sous QNX et cette cible a été appelée *qnx_so*.

6.5 Modification à l'environnement cible

Pour utiliser la bibliothèque dynamique dans Microb, il faut pouvoir compiler cette dernière sur la plateforme cible et permettre son appel à partir du contrôleur global. Cette section présente un survol de ce qui doit être fait sur la plateforme cible; le détail des opérations à effectuer est présenté au CHAPITRE 7.

Pour permettre la compilation de la bibliothèque générée par la suite Mathworks, il faut tout d'abord posséder le code source de Matlab. Si Matlab, Simulink et RTW sont installés sur la plateforme cible, le code est déjà présent. Sinon, il faut copier les fichiers source pour permettre de compiler la bibliothèque (voir section 7.2.2). De plus, il faut posséder le compilateur requis associé à la plateforme, soit *lcc* pour Windows ou *qcc* pour QNX.

L'environnement QNX doit être configuré pour lui permettre de localiser la bibliothèque tant à la compilation qu'à l'exécution (voir section 7.2.2). Pour qu'il localise la bibliothèque lors de la compilation, le nom de la bibliothèque doit être ajouté dans les fichiers de compilation de Microb. Pour permettre la localisation de la bibliothèque par le système d'exploitation, une variable d'environnement doit être modifiée.

Pour appeler la bibliothèque dans le contrôleur global développé avec Microb, il faut ajouter une portion de code qui diffère selon la plateforme. Le code à ajouter sous Windows est disponible à l'ANNEXE 7 et le code requis pour QNX est disponible à l'ANNEXE 8. Il est à noter que la cible QNX génère automatiquement un programme de test qui constitue un excellent exemple d'utilisation de la bibliothèque partagée (voir section 7.2.3).

6.6 Conclusion

Ce chapitre a permis de présenter quelles modifications ont été réalisées autant dans l'environnement de simulation que dans l'environnement cible pour parvenir à utiliser une bibliothèque partagée, sous Windows et sous QNX, avec le CAR Microb. Le prochain chapitre explique comment s'appliquent ces changements, c'est-à-dire comment s'installe et s'utilise en pratique le protocole.

CHAPITRE 7

UTILISATION DU PROTOCOLE DE MIGRATION

7.1 Introduction

La section précédente a permis de comprendre le fonctionnement global du protocole de migration. Ce chapitre présente le manuel d'utilisation qui détaille chacune des étapes à franchir pour effectuer la migration d'une loi de commande, du PRC Simulink vers le CAR Microb. L'utilisation pratique du protocole de migration est présentée par la suite d'un exemple de migration.

7.2 Manuel d'utilisation

La migration s'effectue lorsque l'utilisateur possède un modèle Simulink fonctionnel, avec un certain nombre d'entrées et de sorties. Deux phases devront être franchies afin d'utiliser le protocole de migration, soit l'installation et l'initialisation. Dans un premier temps, les outils nécessaires au protocole devront être installés sur la machine de développement ainsi que sur la machine ciblée pour exécuter le contrôleur. Dans un deuxième temps, le programme du contrôleur Microb doit être initialisé afin que ce dernier puisse utiliser le code encapsulé dans la bibliothèque dynamique.

7.2.1 Préalables à l'installation

La bibliothèque Microb (version mai 2006) doit être installée sur les plateformes de simulation et d'exécution. L'ordinateur de simulation doit fonctionner sous Windows XP. Il doit avoir la suite logicielle Matlab (version 7.0.0.19920 (R14)) avec Real-Time Workshop (RTW 6.0) installée de façon fonctionnelle. Il semble que le protocole

fonctionne avec d'autres versions de Matlab mais aucun test approfondi n'a été fait pour valider cette hypothèse et rien ne laisse présager des problèmes de compatibilité.

7.2.2 Installation des fichiers requis par le protocole

L'installation des fichiers requis par le protocole permet de générer le code sur la plateforme de développement, i.e. celle sur lequel le modèle est développé sous Matlab/Simulink. Cette installation permet également de compiler le code sur la plateforme d'exécution, i.e. celle sur lequel la bibliothèque dynamique sera exécutée. Il est possible que ces deux ordinateurs ne fassent qu'un.

L'Ajout des cibles permettant de générer des bibliothèques dynamiques sous QNX et sous Windows se fait en copiant deux répertoires dans l'arborescence de Matlab. Les deux répertoires contenant les cibles sont présents sur le disque dans le dossier *Protocole_de_Migration\Installation*. Les deux cibles (*qnx_so* et *win_dll*, voir ANNEXE 3 à ANNEXE 6) doivent être ajoutées dans le répertoire *$Matlab$/rtw/c*[2].

Si la machine d'exécution n'a pas Real-Time Workshop, il faut copier les fichiers source fournis par Real-Time Workshop de manière à permettre la compilation des programmes générés. Le répertoire Matlab pour QNX est présent sur le disque dans le dossier *Protocole_de_Migration\Installation\QNX* et doit être copié à la racine (*/*) du système de disque.

7.2.3 Initialisation du protocole

Pour ajouter une nouvelle loi de contrôle au contrôleur global, il faut insérer du code permettant au contrôleur global d'utiliser la bibliothèque partagée. Il est à noter que dans

---

[2] Notez que $Matlab$ correspond au répertoire racine de Matlab, par exemple « C:\Matlab7\ ».

tous les cas, le contrôleur global doit être recompilé après l'initialisation. Un modèle de code est présenté pour un appel sous Windows à l'ANNEXE 7 et pour QNX à l'ANNEXE 8. Un exemple pour le robot Bart est présenté à l'ANNEXE 11. Notez que la cible QNX génère automatiquement un programme de test dont un exemple est disponible à ANNEXE 13. Le code de ce programme de test généré automatiquement peut être inséré directement dans le contrôleur global.

Sous QNX, il faut modifier le fichier de compilation Microb (*makefile*) en ajoutant le nom de la bibliothèque et également effectuer la compilation initiale du contrôleur global. Par exemple, dans le cas du robot Bart, la bibliothèque doit être ajoutée dans le fichier *makefile* du répertoire racine de Bart à la suite de la liste des autres bibliothèques.

7.2.4 Utilisation du protocole

L'utilisation du protocole s'effectue en six étapes distinctes (voir Figure 11), soit:

1. La modélisation dans l'environnement Simulink;
2. La génération du code;
3. La copie du code de l'environnement de simulation à celui d'exécution;
4. La compilation du code;
5. La copie de la bibliothèque dans l'arborescence du CAR;
6. Le démarrage et la validation du modèle dans le contrôleur global.

Les étapes énumérées ci-dessus sont à répéter jusqu'à l'obtention d'une implantation satisfaisante de la loi de contrôle.

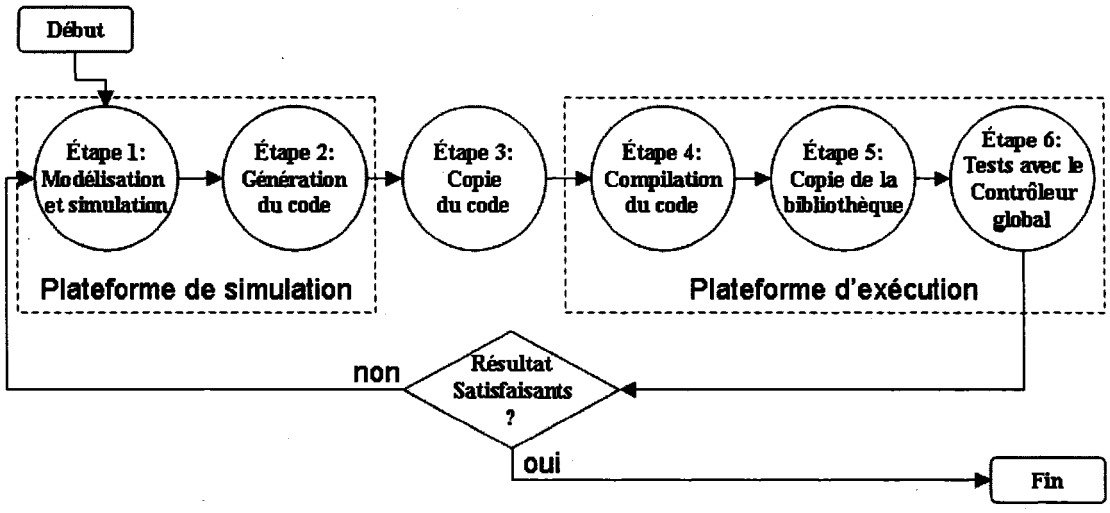

Figure 11 Étapes d'utilisation du protocole de migration

Lors de la modélisation de la loi de contrôle (étape 1), les dimensions de chaque vecteur d'entrées et de sorties doivent être spécifiées à l'intérieur des blocs *inport* et *outport*. De plus, les signaux doivent être identifiés comme étant des données de type double (*Data type = double)* et étant réel (*Signal type = real*) via le menu propriété du bloc (voir Figure 17).

La transmission des signaux de la bibliothèque partagée vers le contrôleur global se fait via un tableau de type *double* et le nombre identifiant le bloc *inport* ou *outport* détermine l'ordre numérique dans ce tableau.

Le modèle doit être configuré pour fonctionner à l'infini avec un pas fixe d'une durée déterminée. Le menu *Simulation / Configuration parameters / Solver* (voir Figure 12) permet de spécifier le temps d'arrêt (*stop time = inf*), le type d'algorithme (*Solver type = Fixed step*) et la période d'échantillonnage (*Fixed step size* = 0.01 secondes ou autre). L'algorithme de résolution (*Solver*) utilisé est à la discrétion de l'utilisateur.

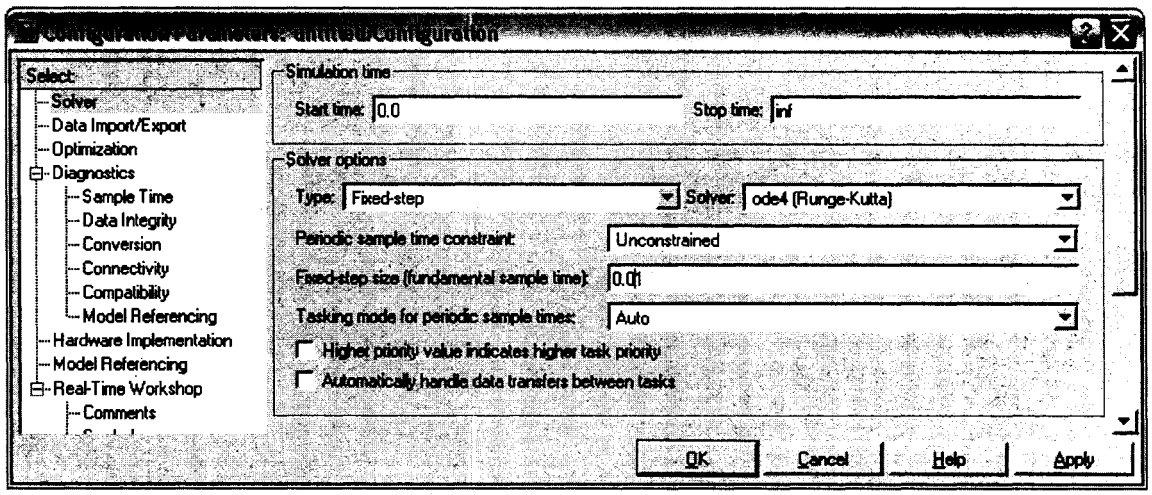

Figure 12 Menu des paramètres de résolution

Migration sous Windows

Pour générer le code et le compiler sous Windows (étapes 2 à 4), il suffit d'aller dans l'interface du modèle Simulink via le menu *Tools / Real-Time Workshop / Options*. Dans l'encadré *Target selection*, il faut appuyer sur le bouton *Browse*, puis choisir dans la liste la cible *win_dll.tlc / Migration Windows (dll)*. Après avoir fermé le menu, la compilation du modèle peut être lancée via le menu *Tools / Real-Time Workshop / Build*. La compilation du modèle devrait se terminer avec succès en affichant dans l'interface Matlab le texte suivant:

```
*** Created dynamic library: $modèle$.dll[3]
### Successful completion of Real-Time Workshop build procedure for model: <$modèle$>[3]
```

La bibliothèque dynamique générée sous Windows doit être copiée (étape 5) dans l'arborescence du CAR. Le répertoire par défaut est généralement le suivant:

[3] Notez que $modèle$ correspond au nom du modèle Simulink, par exemple « CTorque5 ».

$robot$\bin\WindowsNT\PentiumPro[4]. À ce moment, la loi de contrôle est prête à être testée (étape 6).

Migration de Windows vers QNX

Pour générer le code pour QNX à partir d'une plateforme de simulation Windows (étape 2), il suffit d'aller dans l'interface du modèle Simulink via le menu *Tools / Real-Time Workshop / Options*. Dans l'encadré Target selection, il faut appuyer sur le bouton Browse, puis choisir dans la liste la cible *qnx_so.tlc / QNX Neutrino Real-Time Target (Shared library)*. Après avoir fermé le menu, la génération du code du modèle peut être lancée via le menu *Tools / Real-Time Workshop / Build*. Il est à noter que le code n'est pas compilé et l'interface Matlab affiche le texte suivant:

```
### Make will not be invoked - template makefile is for a different host
### Successful completion of Real-Time Workshop build procedure for model: $modèle$
```
[3]

Le répertoire contenant le code source généré, nommé *$modèle$_qnx_so_rtw*[3], doit être copié sur l'ordinateur QNX (étape 3) puis ensuite être compilé (étape 4) en exécutant la commande *make –f $modèle$.mk*[3] sous QNX. La compilation du modèle devrait se terminer avec succès en affichant dans le terminal QNX le texte suivant:

```
*** Created executable: $modèle$
```
[3]

La bibliothèque créée nommée *lib$modèle$.so*[3] doit être copiée (étape 5) dans l'arborescence du CAR, généralement dans le répertoire *$robot$/lib/QNX6/Pentium*[4]. À ce moment, la loi de contrôle est prête à être testée (étape 6).

---

[4] Notez que $robot$ correspond à la variable d'environnement utilisée par Microb pointant sur le répertoire du contrôleur global courant, par exemple « /Bart» sous QNX ou « C:\Bart » sous Windows.

7.3 Exemple de migration

Un scénario typique de migration est présenté dans cette section pour donner un aperçu des étapes à effectuer pour parvenir à implanter, dans un contrôleur global développé avec le CAR Microb, une loi de contrôle élaborée dans Simulink. Le robot présenté dans cet exemple est également utilisé dans l'évaluation du protocole au CHAPITRE 8.

Dans cet exemple, la plateforme de simulation est différente de la plateforme d'exécution afin d'illustrer le cas le plus complexe. Certains logiciels doivent être présents sur les deux plateformes (voir section 7.2.1). La plateforme de simulation tourne sous Windows XP et la suite complète de Mathworks y est installée, incluant Real-Time Workshop. Sur la plateforme d'exécution tournant sous QNX, le CAR Microb est installé ainsi que le contrôleur global du robot Bart.

Avant d'utiliser le protocole sur un nouvel ordinateur, tant sur la plateforme de simulation que sur la plateforme d'exécution, il est nécessaire de procéder à son installation (voir section 7.2.2). L'installation sur l'ordinateur de simulation requiert que les nouvelles cibles de Real-Time Workshop (*win_dll* et *qnx_so*) soient copiées dans l'arborescence de Matlab. L'installation sur l'ordinateur d'exécution nécessite de copier les répertoires contenant le code source utilisé pour compiler les programmes générés par Real-Time Workshop. Il est également requis d'initialiser des variables d'environnement lorsque le protocole est utilisé sous le système d'exploitation QNX.

Le protocole doit être initialisé (voir section 7.2.3) lorsqu'une nouvelle loi de contrôle est intégrée au contrôleur global, ou encore lorsque les entrées et les sorties de la loi de contrôle sont modifiées. Pour initialiser le protocole, il faut ajouter du code dans le contrôleur global développé à l'aide du CAR Microb (voir ANNEXE 12). De plus, lorsque la plateforme d'exécution fonctionne avec le système d'exploitation QNX, il

faut modifier le fichier de compilation (voir ANNEXE 10) afin qu'il soit en mesure de localiser la bibliothèque lors de la compilation du contrôleur global.

L'utilisation du protocole (voir section 7.2.4), permet d'effectuer les itérations nécessaires pour implémenter la loi de contrôle avec le contrôleur global. Elle rend possible la modification et la validation du modèle de simulation jusqu'à l'obtention de résultats satisfaisants. Cette activité consiste à générer les fichiers en langage C, à les copier, les compiler puis les tester sur la plateforme d'exécution (voir Figure 13). Ces étapes sont répétées jusqu'à l'obtention d'un contrôleur jugé satisfaisant.

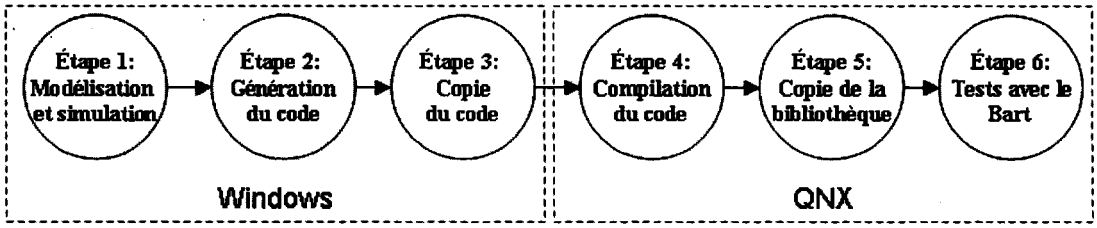

Figure 13 Déroulement de l'implantation d'une loi de contrôle pour Bart

Dans l'exemple présent, le contrôleur à couple précalculé nommé CTorque4, dont le schéma est présenté à la Figure 15, est modélisé dans Simulink (voir Figure 16). Les vecteurs d'entrées sont représentés par les blocs Simulink *inport* et sont au nombre de sept (X, Xp, Xd, Xpd, Xppd, kp, kd, ki). L'unique vecteur de sortie est représenté par le bloc Simulink *outport* nommé Tau. Les paramètres physiques du robot Bart (voir ANNEXE 9) sont encapsulés dans le modèle mais auraient également pu être passée en entrée sous forme de matrice.

Étape 1: Modélisation et simulation

L'objectif poursuivi est de faire migrer une loi de contrôle de Simulink pour l'implanter dans un contrôleur de robot développé sous Microb. Le robot, nommé Bart, est un

manipulateur à deux degrés de liberté dont les deux articulations rotoïdes tel qu'illustré à la Figure 14. Il est couramment utilisé à titre d'exemple dans la littérature [64; 67; 68] pour sa simplicité et sa dynamique bien connue. Les spécifications détaillées de ce robot sont disponibles à l'ANNEXE 9.

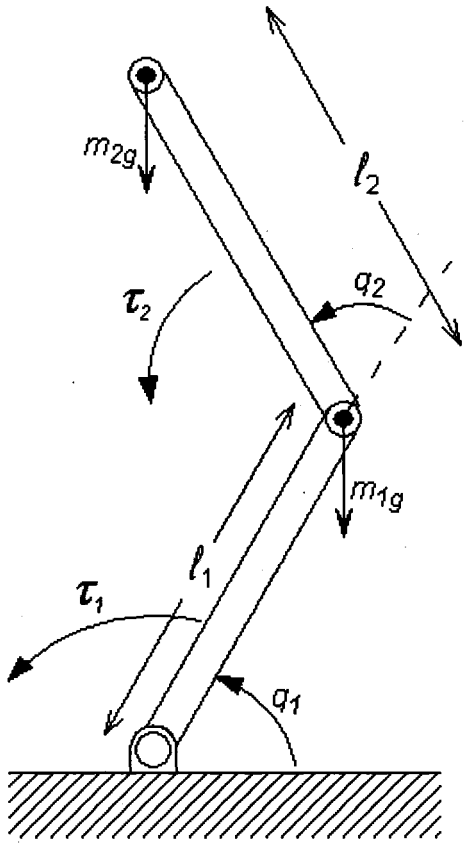

Figure 14 Illustration du robot Bart

La loi de contrôle est un contrôleur à couple précalculé [67]. La loi de contrôle non linéaire a été modifiée afin de lui ajouter un intégrateur. L'usage d'un intégrateur permet de prouver que les états continus et l'intégration temporelle sont bien supportés par la bibliothèque générée.

La loi de contrôle est la suivante:

$$\tau = \mathbf{M}(\mathbf{q}) \cdot \mathbf{u} + \mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q},\dot{\mathbf{q}})$$

Où:

$\mathbf{M}(\mathbf{q}), \mathbf{V}(\mathbf{q},\dot{\mathbf{q}}), \mathbf{G}(\mathbf{q})$ et $\mathbf{F}(\mathbf{q},\dot{\mathbf{q}})$ sont respectivement la matrice de masse ainsi que les vecteurs des forces centrifuges et de Coriolis, de gravité et de frottement. La partie linéaire de la loi de commande (proportionnelle, intégrale et dérivée, avec un terme d'anticipation) est donnée par

$$\mathbf{u} = \ddot{\mathbf{q}}_d + \mathbf{K}_v \dot{\mathbf{e}} + \mathbf{K}_p \mathbf{e} + \mathbf{K}_i \int \mathbf{e}$$

$\mathbf{q}_d$ est le vecteur de position désirée

$\dot{\mathbf{q}}_d$ est le vecteur de vitesse désirée

$\ddot{\mathbf{q}}_d$ est le vecteur d'accélération désirée

$\mathbf{e}$ est le vecteur d'erreur sur la position articulaire ($\mathbf{q}_d - \mathbf{q}$)

$\dot{\mathbf{e}}$ est la dérivée du vecteur d'erreur par rapport au temps

$\int \mathbf{e}$ est l'intégrale du vecteur d'erreur par rapport au temps

$\mathbf{K}_v$, $\mathbf{K}_p$ $\mathbf{K}_i$ sont les matrices de gains proportionnels, dérivés et intégraux. La valeur de ces gains est respectivement (fixée d'après [68]):

$\mathbf{K}_v$ = [37.62 37.62]

$\mathbf{K}_p$ = [471.75 471.75]

$\mathbf{K}_i$ = [1971.9 1971.9]

Le schéma de la Figure 15, à l'exception du bloc robot, représente le contrôleur que l'on veut encapsuler dans la bibliothèque dynamique afin qu'il retourne le couple τ à partir des signaux d'entrée $\ddot{\mathbf{q}}_d, \dot{\mathbf{q}}_d, \mathbf{q}_d, \mathbf{q}, \dot{\mathbf{q}}$ ainsi que les gains $\mathbf{K}_v$, $\mathbf{K}_p$, $\mathbf{K}_i$. Le contrôleur global développé avec le CAR Microb a donc la responsabilité de fournir les matrices de gains

(K_v, K_p, K_i), les signaux de consigne ($\ddot{q}_d, \dot{q}_d, q_d$) ainsi que la lecture des encodeurs ($q, \dot{q}$).

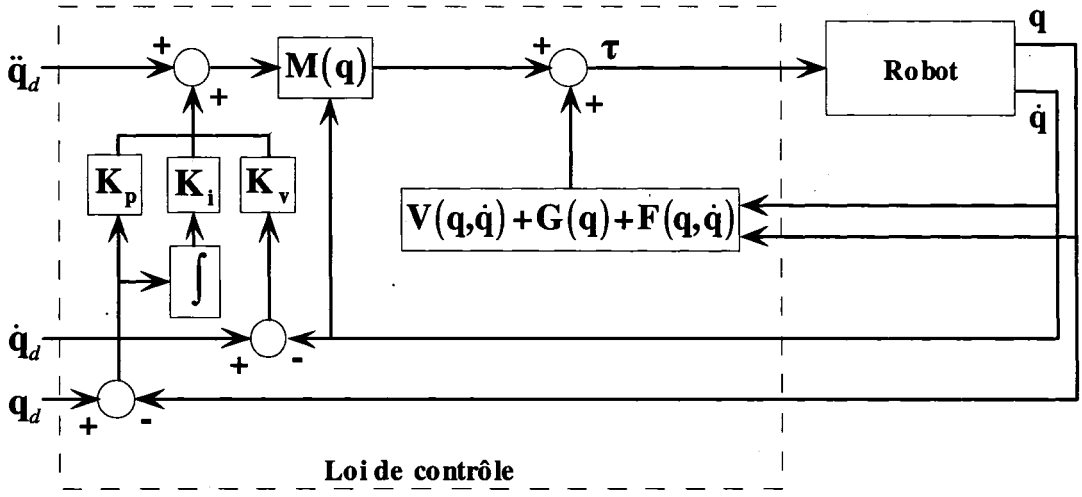

Figure 15 Schéma de la loi de contrôle à couple précalculé (avec PID)

La dimension de chaque vecteur a été spécifiée à l'intérieur des blocs *inport* et *outport* (voir Figure 17). Il est à noter que les signaux ont également été identifiés comme étant des données de type double (*Data type = double*) et qu'il s'agit d'un signal réel (*Signal type = real*).

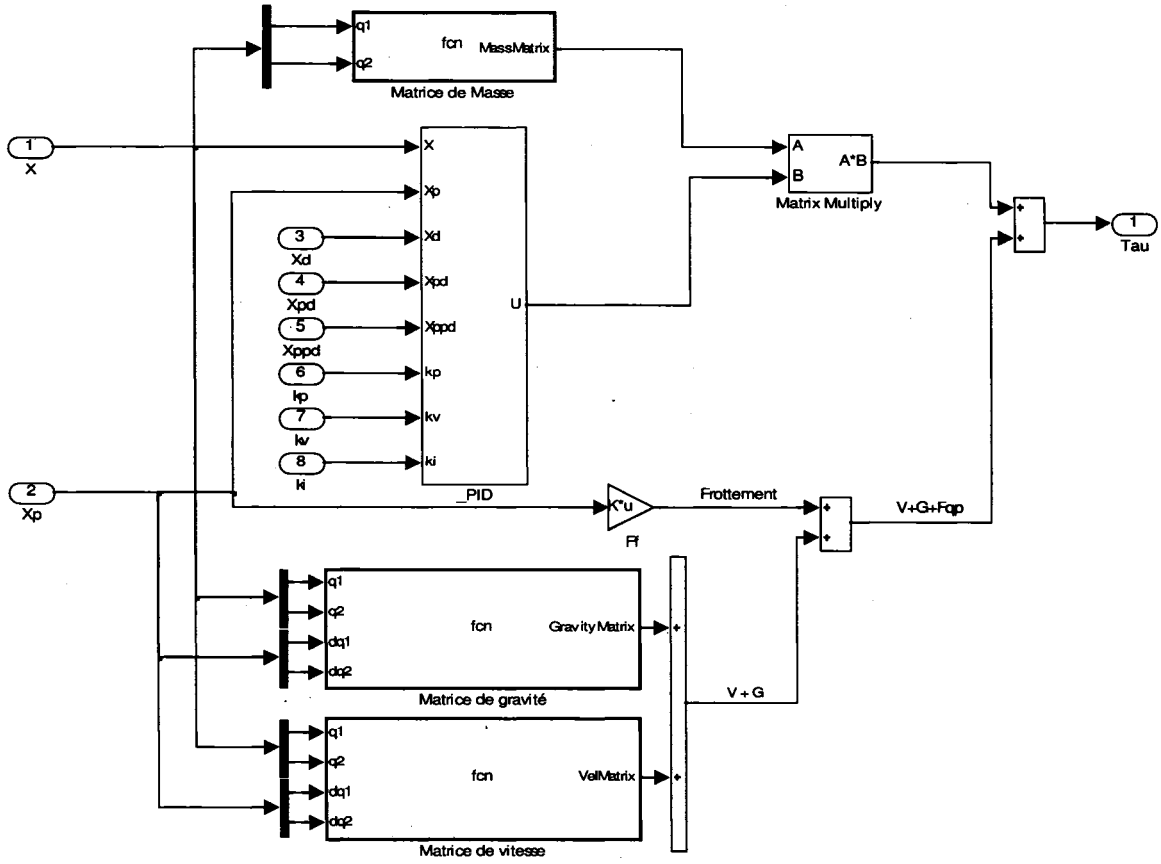

Figure 16 Implémentation sous Simulink de la loi de contrôle

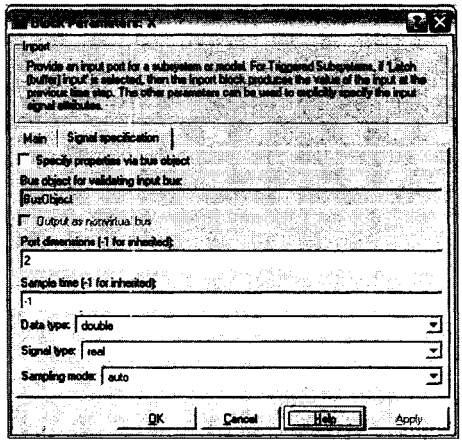

Figure 17 Spécification de la dimension des vecteurs d'entrée et de sortie

Étape 2: Génération du code

Lorsque toutes les dimensions et les types des signaux sont spécifiés, le choix de la cible QNX est fait dans l'interface Simulink via le menu *Tools / Real-Time Workshop/ Options* (voir Figure 18). Ensuite le code est généré via la commande Build (voir Figure 19). Il est à noter qu'un programme de test constituant un exemple d'utilisation de code est généré automatiquement pour la plateforme QNX à ce moment (voir ANNEXE 13). Ce code peut également être utilisé pour initialiser le protocole (voir section 7.2.3).

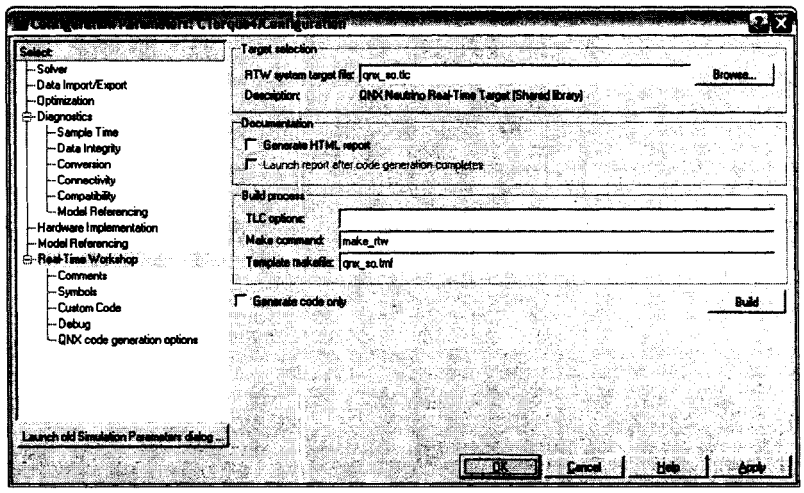

Figure 18 Choix de la cible QNX

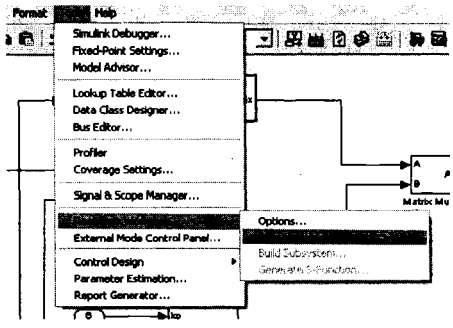

Figure 19 Démarrage de la génération de code

Étape 3: copie du code

Le code ainsi généré contenu dans le répertoire *CTorque4_qnx_so_rtw* doit être copié sur la plateforme QNX.

Étape 4: Compilation du code

La compilation du modèle est lancée dans un terminal dans le répertoire copié à l'aide de la commande *make –f CTorque4.mk*. Cette commande retourne le message « *** *Created executable: CTorque4* » lorsqu'elle s'exécute avec succès.

Étape 5: Copie de la bibliothèque

Le fichier généré nommé libCTorque4.so doit être copié dans le répertoire du contrôleur Bart (*/Bart/lib/QNX6/Pentium/*).

Étape 6: Tests avec Bart

Le contrôleur global peut être démarré pour tester la nouvelle loi de contrôle ainsi migrée. Si le contrôleur se comporte de façon satisfaisante, le processus est terminé. Dans le cas contraire, l'utilisateur retourne à l'étape 1 pour effectuer la modélisation et la simulation de la loi de contrôle.

En résumé, lorsque les logiciels préalables sont présents sur l'ordinateur effectuant la migration (voir section 7.2.1), il est indispensable d'installer les fichiers requis par le protocole (voir section 7.2.2). Puis lorsqu'une nouvelle loi de contrôle doit être implantée, il est nécessaire de réinitialiser le contrôleur global (voir section 7.2.3). Lorsque ces deux étapes sont franchies, il est possible d'utiliser le protocole pour effectuer l'activité de conception proprement dite (voir section 7.2.4).

7.4 Conclusion

La présentation d'un scénario typique de migration a permis de mettre en contexte les étapes d'installation, d'initialisation et d'utilisation du protocole présentées dans ce chapitre. Le présent chapitre constitue donc le manuel de l'usager du protocole. L'évaluation du protocole est effectuée au chapitre suivant.

CHAPITRE 8

ÉVALUATION DU PROTOCOLE DE MIGRATION

8.1 Introduction

Le protocole de migration est développé et documenté. Cette section permet d'évaluer qualitativement et quantitativement le protocole de migration afin de connaître son niveau de succès.

8.2 Démarche d'évaluation

L'évaluation du protocole a deux facettes, l'une quantitative et l'autre qualitative.

L'évaluation quantitative du protocole a pour but de déterminer quelles sont les performances résultantes d'un produit de migration utilisé avec un mécanisme plus complexe que le vérin utilisé au CHAPITRE 5, qui avait pour but de faire le choix technologique du produit de migration.

L'évaluation qualitative du protocole a pour but de connaître l'appréciation des utilisateurs face à l'emploi du protocole de migration. Cette évaluation est effectuée en faisant expérimenter le protocole aux utilisateurs, puis en leur soumettant un questionnaire d'évaluation.

8.3 Évaluation quantitative

L'évaluation quantitative est décrite dans cette section. Les critères évalués sont présentés, suivi de la description des éléments à évaluer pour se terminer avec l'analyse des résultats obtenus.

8.3.1 Critères évalués

Tous les critères évalués sont mesurés dans deux situations. La première situation est caractérisée par le fait que le contrôleur global et la loi de contrôle sont développés exclusivement avec le CAR Microb. C'est normalement cette situation qui donne les meilleures performances. Les performances obtenues servent de base de comparaison. La seconde situation est caractérisée par l'utilisation du protocole de migration pour intégrer la loi de contrôle au même contrôleur global.

Les couples calculés pour les deux situations par la loi de contrôle sont comparés pour valider l'exactitude des calculs effectués. Trois des quatre critères présentés au CHAPITRE 5 sont évalués, soit le temps de cycle, la mémoire vive utilisée et l'espace disque utilisé. Toutefois, comme le temps de migration ne peut être comparé significativement, il n'est pas mesuré.

8.3.2 Description des éléments évalués

Le contrôleur global Bart, décrit à la section 7.3 et dont les paramètres détaillés sont à l'ANNEXE 9, est employé pour l'évaluation. Toutes les mesures sont prises sur les deux plateformes supportées par le protocole de migration, soit sous Windows XP et QNX. Les performances du contrôleur à couple précalculé muni d'un PID, présenté à la Figure 15, sont mesurées.

Chaque articulation du robot doit suivre un profil de vitesse trapézoïdal calculé par le contrôleur global tel qu'illustré à la Figure 20. Par simplicité, chaque articulation suit la même trajectoire. Les informations sont enregistrées durant 7 secondes.

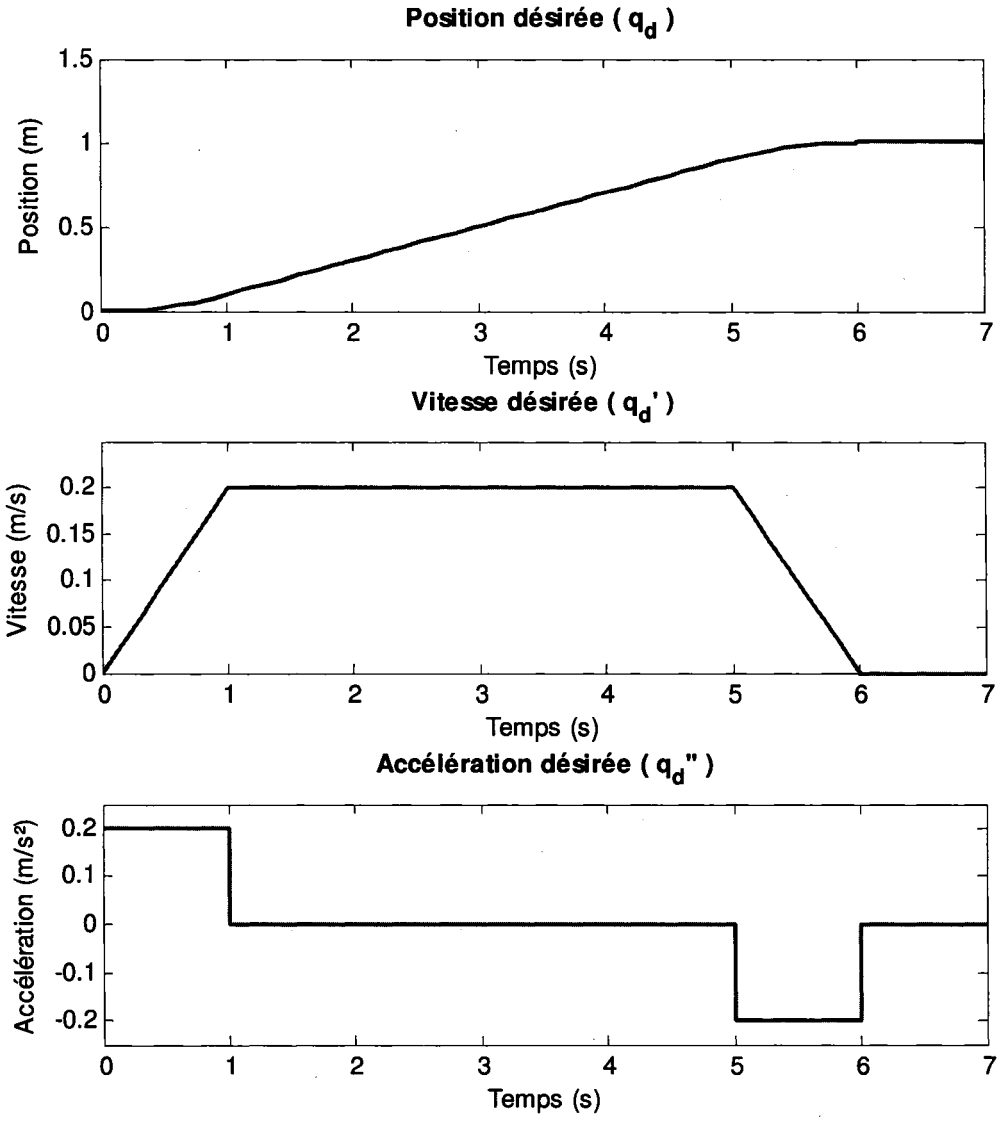

Figure 20 Illustration de la trajectoire désirée

8.3.3 Analyse des résultats

Les couples calculés sont dans un premier temps analysés afin de valider les calculs effectués. Par la suite, les critères (temps de calcul, mémoire vive utilisée, espace disque occupé) sont mesurés puis analysés pour connaître les performances réelles du protocole. Le sommaire des résultats est présenté dans le Tableau V. Il est à noter que le

contrôleur codé manuellement a été optimisé de façon à diminuer le temps de calcul présentés dans le Tableau VI.

Tableau V

Sommaire de l'évaluation quantitative

Critères	Contrôleur codé manuellement	Contrôleur Migré	Augmentation
Temps de calcul moyen sous Windows XP (ms)	0,0100	0,0099	-1,3%
Temps de calcul moyen sous QNX (ms)	0,0251	0,0385	53,3%
Mémoire vive utilisée sous Windows XP (Ko)	2788	3012	8,0%
Mémoire vive utilisée sous QNX (Ko)	1748	1876	7,3%
Espace disque occupé sous Windows XP (Ko)	396	432	9,1%
Espace disque occupé sous QNX (Ko)	1736	1886	8,6%

Tableau VI

Comparaison des temps de calculs après optimisation du code manuel

Critères	Contrôleur codé manuellement (optimisé)	Contrôleur Migré	Augmentation
Temps de calcul moyen sous Windows XP (ms)	0,0078	0,0099	26,4%
Temps de calcul moyen sous QNX (ms)	0,0172	0,0385	123,3%

8.3.3.1 Couples calculés

Les couples calculés, tant sous Windows XP (voir Figure 21) que sous QNX (Figure 22), diffèrent légèrement lorsqu'ils sont calculés avec Microb ou avec la bibliothèque partagée. Il est suspecté que la méthode d'intégration soit en cause. Bien que les deux calculs utilisent l'algorithme d'intégration de Runge-Kutta d'ordre 4, des

différences dans l'implémentation de cet algorithme peuvent conduire à ces écarts dans les résultats obtenus. Toutefois, l'écart maximal relatif obtenu est en deçà de 0,055 %, ce qui peut être considéré comme étant négligeable.

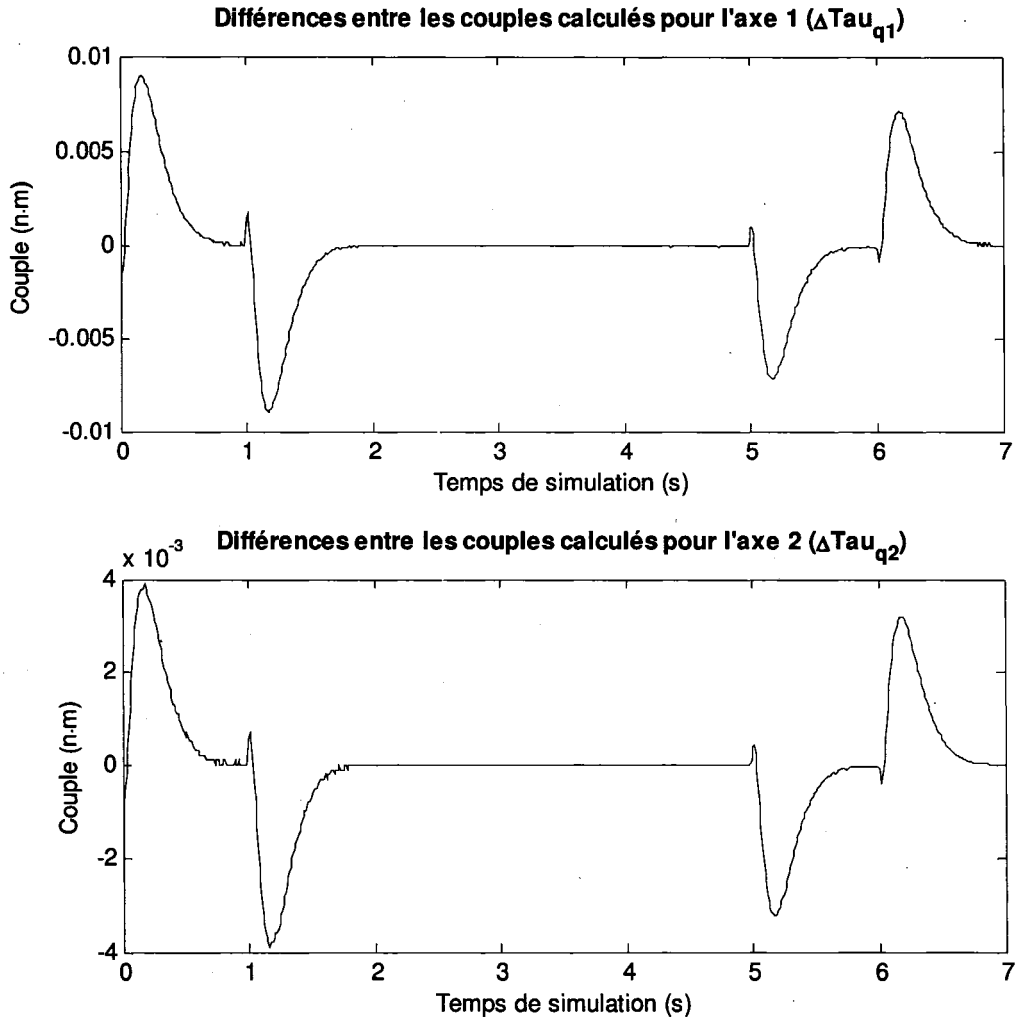

Figure 21 Différence entre les couples calculés par Microb et par la bibliothèque partagée sous Windows XP

Afin de confirmer que c'est l'implémentation de l'intégrateur qui cause cette différence de calcul, la simulation est relancée en posant un gain intégral nul. Cela a pour effet d'annuler complètement l'effet de l'intégrateur. L'erreur obtenue suite à cette

modification est nulle. Ce résultat confirme donc que les calculs sans intégration sont exacts. Cela met également en relief le fait qu'il peut y avoir des différences de calculs d'intégration entre Microb et les bibliothèques partagées.

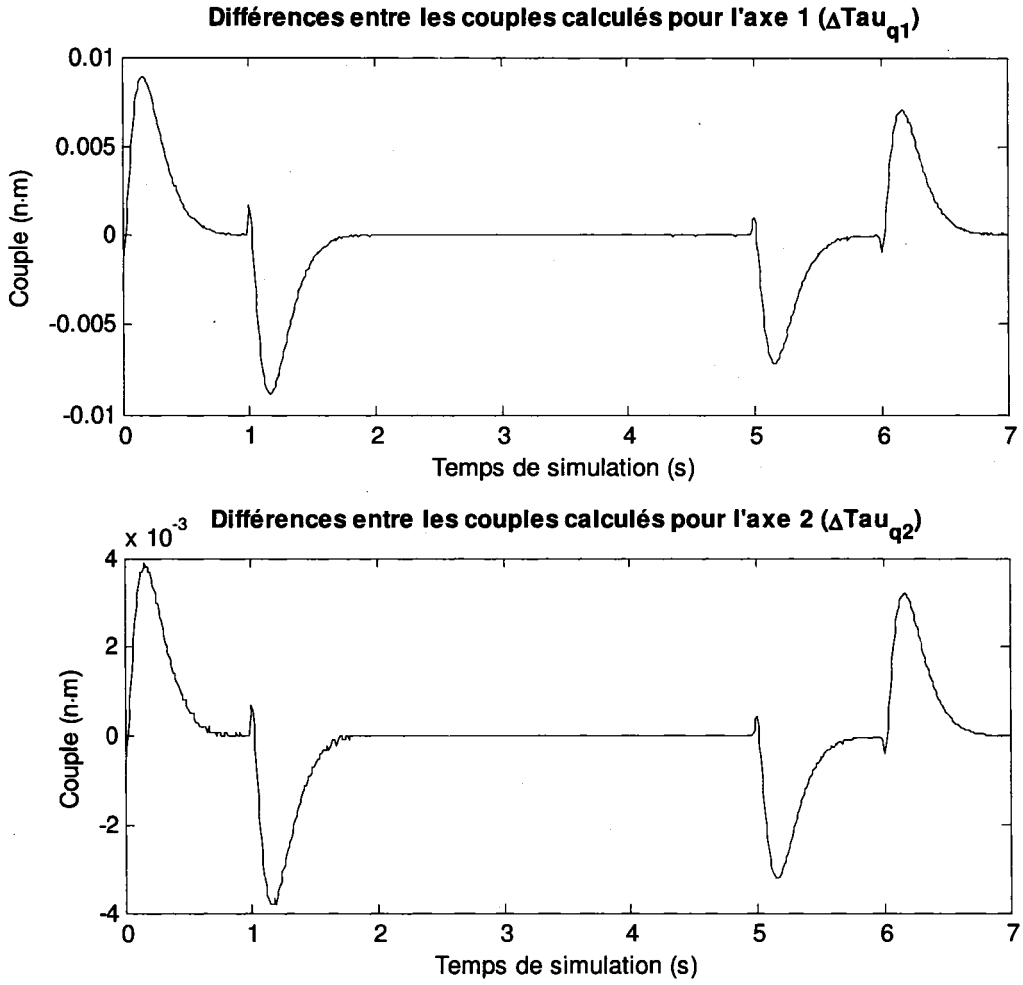

Figure 22 Différence entre les couples calculés par Microb et par la bibliothèque partagée sous QNX

Il est difficile d'affirmer quelle est la méthode d'implémentation de l'algorithme d'intégration qui est supérieure à l'autre. D'un côté, le logiciel Matlab possède une renommée qui le précède. D'un autre côté, une expérience réalisée avec Simulink (voir

Figure 23) montre que le logiciel a recours à l'extrapolation des points pour se rapprocher de la réalité.

Dans cette expérience, deux signaux rampes sont intégrés. Le premier signal vient du bloc rampe natif de Simulink (pente = 1) et le second est généré à partir d'une variable initialisé manuellement (x = [0:.1:5; 0: .1:5]'). Il est à noter que les deux signaux utilisés sont numériquement identiques, i.e. que la différence est nulle entre les deux signaux illustrés dans le bloc « Erreur signal ».

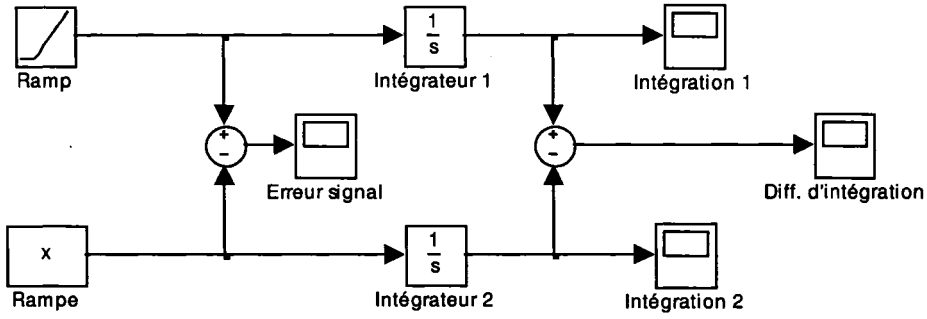

Figure 23 Schéma expérimental sous Simulink

Les différences entre les intégrations suite à une simulation de 5 secondes sont illustrées à la Figure 24. La différence entre les deux méthodes est constante (0,00333 par itération), donc cela signifie qu'il s'agit d'une erreur linéaire.

Dans le cas de l'intégration d'un signal provenant d'un bloc rampe, Simulink n'intègre pas à l'aide d'un bloqueur d'ordre zéro, mais en considérant la pente de la rampe et en extrapolant le résultat. Cependant, lorsque le type de signal est camouflé à Simulink à l'aide d'une variable issue de l'environnement de Matlab, l'intégration est réalisée avec un bloqueur d'ordre zéro et aucune extrapolation n'est réalisée.

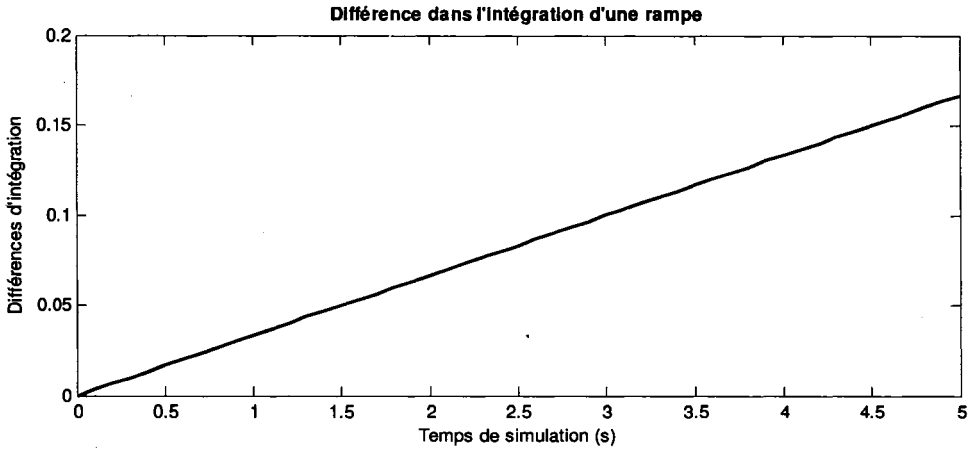

Figure 24 Différence d'intégration d'une rampe

Bien que cette expérience ne prouve pas que Matlab soit dans l'erreur, elle montre que le logiciel ne se comporte pas toujours de la même façon selon le signal reçu en entrée. Il n'est donc pas possible de conclure que l'approche de Matlab ou celle Microb donne des résultats plus exacts. Des tests plus poussés doivent être faits avant de statuer sur la question et cela ne fera pas l'objet de ce mémoire. Toutefois, comme les deux méthodes arrivent à des résultats similaires, elles peuvent être considérées comme équivalentes.

8.3.3.2 Temps de calculs

Les temps de calculs sont illustrés dans les graphiques qui suivent (voir Figure 25 à Figure 28). Il est bon de souligner que les données ont été lissées à l'aide de la fonction *smooth* de Matlab pour permettre une meilleure lisibilité des graphiques. Les pointes dans les graphiques représentent un calcul interrompu par le processeur. Sous Windows XP, les calculs de la bibliothèque dynamique sont 1.3 % plus rapide que ceux entièrement effectués uniquement par le contrôleur global. En simplifiant l'équation, il est possible d'optimiser le code manuel pour minimiser le temps de calcul. Cette optimisation permet d'avoir un contrôleur 23% plus rapide que le contrôleur non optimisé sous Microb et 26.4% plus rapide que le contrôleur migré de Simulink.

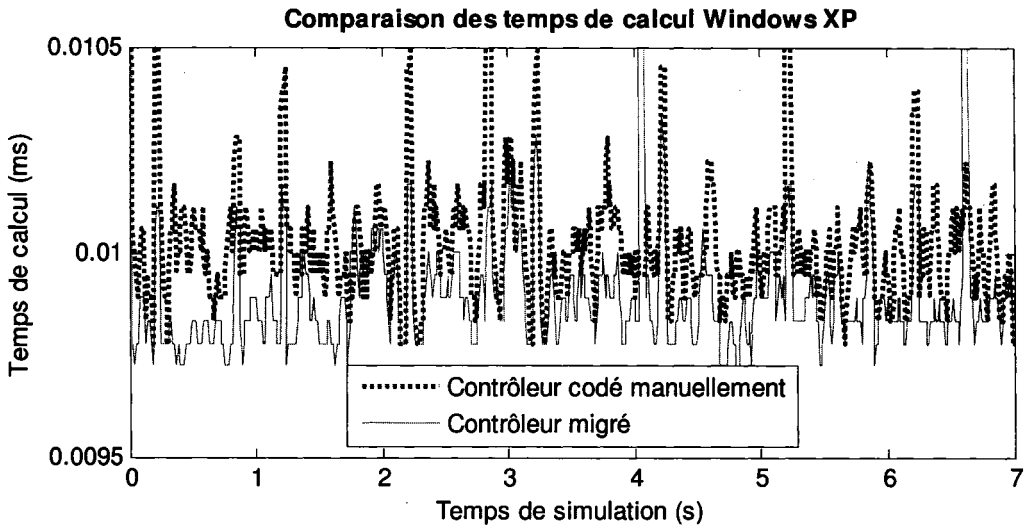

Figure 25 Temps de calcul sous Windows XP

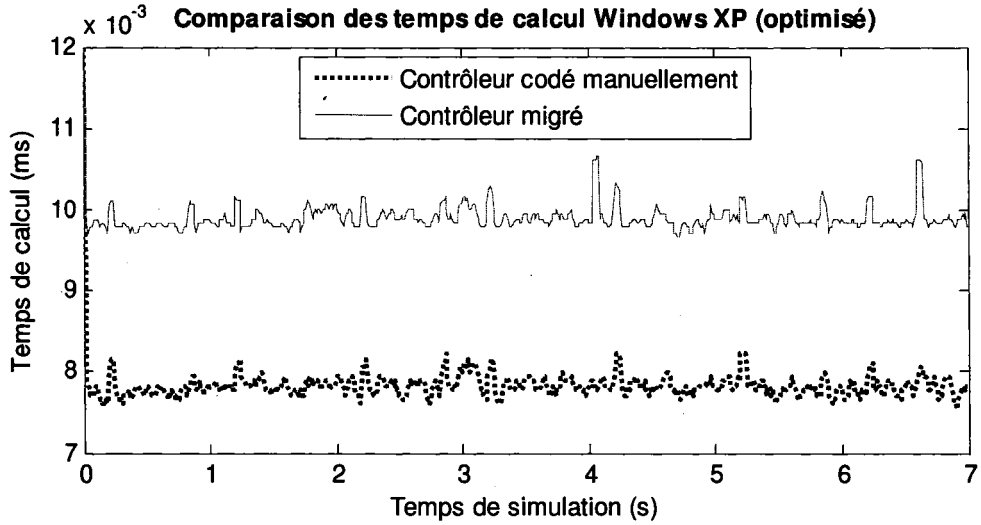

Figure 26 Temps de calcul sous Windows XP après optimisation

Les calculs de la bibliothèque partagée sous QNX sont 53 % plus lents que ceux entièrement effectués uniquement par le contrôleur global. En simplifiant l'équation, il est possible d'optimiser les calculs. Cette optimisation permet d'avoir un contrôleur 31,4 % plus rapide que le contrôleur sous Microb et 123 % plus rapide que le contrôleur

migré. Cette moins bonne performance sous QNX est potentiellement due au fait que les mesures ont été prises sous une machine virtuelle ou encore à la façon dont QNX gère ses bibliothèques partagées.

La comparaison avec les contrôleurs optimisés, tel que présenté dans la Figure 26 et la Figure 28, met en relief le fait que la bibliothèque générée par Matlab optimise les calculs, ce qui peut résulter en un gain de performance net lorsque la loi de contrôle est suffisamment complexe.

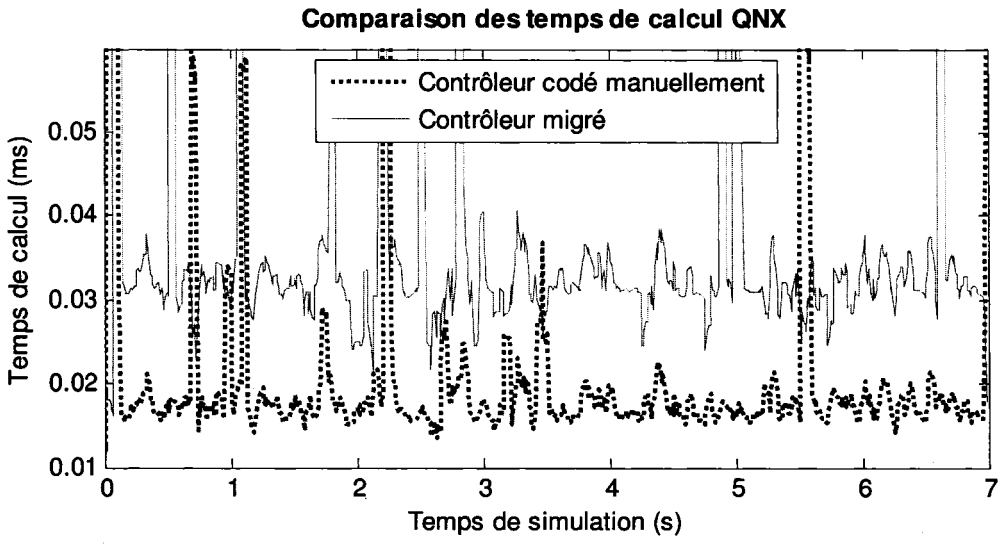

Figure 27 Temps de calcul sous QNX

Le temps supplémentaire requis par les calculs avec la bibliothèque partagée, allant jusqu'à 123 %, peut sembler important mais représente en fait une fraction négligeable du temps de cycle (~0,1 %). Malgré que cette hausse soit supérieure à 10%, elle est considérée comme admissible car elle est peu élevée par rapport à la période d'échantillonnage (T = 0.01 s, 100 Hz). Comme cette hausse est inférieure à 10% de la période d'échantillonnage, elle est considérée comme acceptable [51]. Dans le pire des cas, la perte d'efficacité attendrait 10% du temps de cycle pour un contrôleur fonctionnant à une fréquence 100 fois plus élevée (T = 0.0001 s, 10 000 Hz).

Néanmoins, d'autres tests pourraient être faits avec des lois de commande plus complexes.

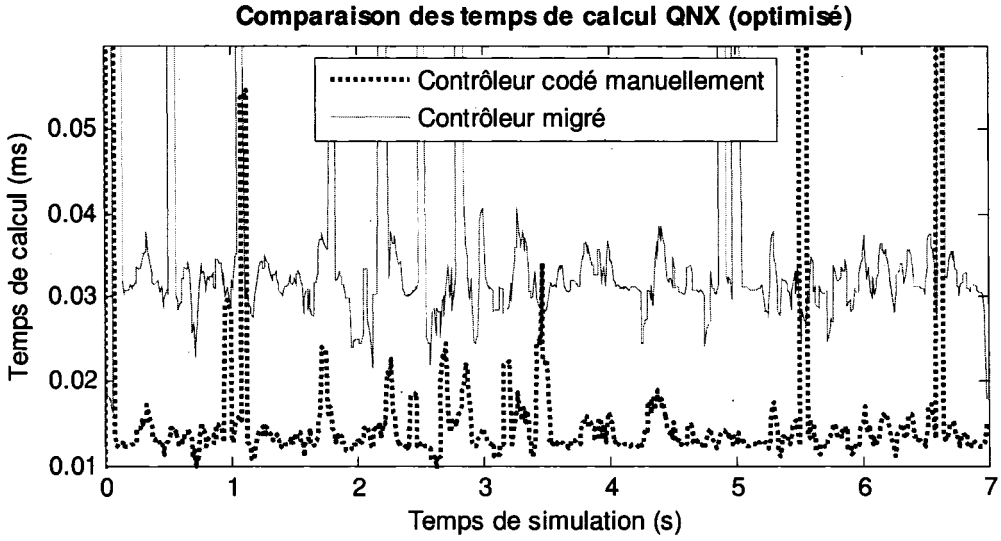

Figure 28 Temps de calcul sous QNX après optimisation

8.3.3.3 Mémoire vive utilisée

La mesure de mémoire vive est prise en faisant la différence entre la mémoire vive totale utilisée par le système avant et celle à la fin de l'exécution du contrôleur global. La mémoire vive utilisée sous Windows XP par le contrôleur codé manuellement est de 2788 Ko tandis que la mémoire vive utilisée par le contrôleur migré sous forme de bibliothèque dynamique est de 3012 Ko. Le contrôleur global utilisant une loi de migration migrée augmente la mémoire vive utilisée de 8 % (3012 / 2788). Sous QNX, une augmentation de 7.3 % ((1748 Ko +128 Ko) / 1748 Ko) est observée. Comme cette hausse est sous la barre du 10 %, elle est considérée comme acceptable [51].

8.3.3.4 Espace disque utilisé

L'espace disque utilisée sous Windows XP augmente de 9 % ((396ko + 36ko)/396 Ko) en utilisant le produit de migration. Sous QNX, une hausse similaire de 8.6 % ((1736 Ko + 150 Ko) / 1736 Ko) de l'utilisation de l'espace disque est observée. Comme cette hausse est sous la barre du 10 %, elle est considérée comme admissible [51].

8.4 Évaluation qualitative

L'évaluation consiste à consulter un groupe d'usager. Ce groupe évalue le protocole en le testant puis en répondant à un questionnaire (voir ANNEXE 2). Ce questionnaire est basé sur les besoins des usagers décrits au CHAPITRE 4.

L'expérimentation du protocole pour l'évaluation est similaire à l'exemple d'utilisation présenté à la section 7.3. La principale différence entre l'exemple et l'évaluation tient dans le fait que la migration est effectuée uniquement sous Windows XP au lieu d'être de Windows XP vers QNX comme dans l'exemple.

Les six évaluateurs sont invités à répondre à un questionnaire comportant douze questions ouvertes et une section de commentaires. Les mêmes personnes qui ont effectués la consultation au CHAPITRE 4 répondent à ce sondage, à l'exception de moi-même et d'une autre personne qui n'était malheureusement pas disponible.

Il est à noter que deux erreurs se sont glissées aux questions 6 et 8 au niveau du pourcentage d'augmentation du temps de calcul requis et de la mémoire utilisé: on aurait dû lire respectivement 26,4 % (ou -1.3 %) et 10 % au lieu de 10 % et 40 %. Les réponses à ces deux questions ont donc été ignorées.

Les résultats obtenus suite à cette consultation sont les suivants:

- 83 % des répondants jugent que le protocole de migration remplit son mandat. La personne insatisfaite mentionne que le protocole répond partiellement au mandat car une partie du code doit être programmé par un usager Microb expérimenté;
- 100 % des répondants saisissent bien ce que le protocole de migration fait;
- 100 % des répondants trouvent que l'apprentissage du protocole de migration est aisé;
- 100 % des répondants trouvent la migration facile à effectuer. Un répondant mentionne que le protocole demande quand même un travail d'apprentissage pour le programmeur Microb;
- 67 % des répondants considèrent Real-Time Workshop comme étant un générateur de code suffisamment fiable pour être utilisé dans le développement de contrôleurs de robot. Les autres répondants s'abstiennent;
- 100 % des répondants considèrent le temps de migration requis comme satisfaisant;
- 100 % des répondants considèrent comme acceptable une augmentation d'environ 10 % de l'espace disque (ROM) dû à l'utilisation d'un contrôleur migré;
- 83 % des répondants trouvent l'installation du protocole suffisamment simple et claire. Le répondant insatisfait mentionne que le texte pourrait être retravaillé, ce qui a été fait dans le CHAPITRE 7;
- 100 % des répondants trouvent que le protocole de migration s'intègre harmonieusement à l'environnement Matlab \ Simulink.

Certains ont émis des suggestions pour améliorer le protocole: clarifier la procédure d'installation et d'utilisation du protocole, automatiser la copie des fichiers directement dans Matlab et fournir un exemple complet d'utilisation, incluant ce qu'il faut faire du côté Microb, ce qui a été fait dans le CHAPITRE 7. Le commentaire positif suivant a été émis: « Bon travail, très prometteur pour diverses applications». En bref, les résultats de

l'évaluation montrent que sur l'ensemble des questions, la vaste majorité des utilisateurs potentiels semblent satisfaits du protocole de migration développé.

8.5 Problèmes observés

Lors des tests sous Windows XP, l'utilisation de la bibliothèque dynamique a pour effet de provoquer une panne dans le contrôleur global lorsqu'il s'approche de 65 000 itérations. Ce problème est probablement dû au mécanisme de chargement de bibliothèque dynamique sous Windows car il n'est pas observé sous QNX. La voie de solution envisageable à emprunter consiste à modifier le chargement de la bibliothèque pour qu'il se fasse au chargement de l'application plutôt que durant son le début de son exécution, comme c'est fait sous QNX.

8.6 Conclusion

L'évaluation du protocole de migration, tant quantitative que qualitative, s'est avérée positive. Voici les points saillants de cette évaluation:

- Les algorithmes d'intégration du produit de migration n'arrivent pas tout à fait aux mêmes résultats que ceux du CAR Microb mais donnent des résultats très semblables (différence relative maximale de 0.05 %);
- Bien que l'augmentation du temps de calcul sous QNX soit plus importante que prévue, les temps de calculs demeurent négligeables par rapport à la période d'échantillonnage (< 0.1 %);
- L'augmentation de la mémoire vive utilisée ainsi que de l'espace disque requis provoquée par l'utilisation du produit de migration est raisonnable car inférieure à 10 % [51];
- La majorité des utilisateurs consultés sont satisfaits du protocole de migration développé;

- Plusieurs utilisateurs apprécieraient que la copie des fichiers soit automatisée.

En conclusion, le protocole atteint les objectifs, tant qualitatifs que quantitatifs, fixés au CHAPITRE 4.

CONCLUSION

Un protocole de migration est proposé dans ce mémoire. La migration désirée est celle d'une loi de contrôle développée par un outil de PRC vers un contrôleur global basé sur un CAR. Cette migration est effectuée à partir d'un environnement de PRC composé de la chaîne d'outils Matlab / Simulink / RTW. La loi de contrôle générée est encapsulée dans une bibliothèque partagée puis intégrée au CAR Microb.

Le protocole proposé est développé sur la base des objectifs fixés par les utilisateurs (CHAPITRE 4). Ces objectifs ont déterminé le choix du produit de migration (CHAPITRE 5), puis ont permis de développer le protocole de migration (CHAPITRE 6), d'en décrire l'utilisation (CHAPITRE 7) et finalement d'en évaluer les performances (CHAPITRE 8).

Les performances obtenues montrent que le processus de migration est viable et prometteur. L'évaluation quantitative montre que le produit de migration performe bien. Les tests effectués présentent que l'utilisation du protocole de migration n'affecte pas significativement les performances du contrôleur global. L'évaluation qualitative met en relief le fait que le protocole de migration répond aux besoins des utilisateurs et que ces derniers en sont satisfaits.

Ce protocole de migration est développé spécifiquement pour faire migrer des lois de contrôle robotiques mais peut être utilisé à d'autres fins. Par exemple, il est possible d'employer le protocole pour faire migrer des modèles de robot ou de n'importe quel autre mécanisme à des fins de simulation.

RECOMMANDATIONS

D'après les tests effectués au CHAPITRE 8, le produit de migration est prêt à être utilisé en simulation, tant pour des lois de contrôle que pour des modèles de procédés.

Par contre, avant d'employer le protocole sur des mécanismes physiques, la fiabilité du produit de migration doit être assurée afin d'éviter d'éventuels accidents ou bris de matériel. Cela peut être fait en effectuant divers tests sur la stabilité du logiciel, tant en simulation qu'en exécution en temps réel. Cette activité de validation de fiabilité est suffisamment importante pour faire l'objet d'un travail distinct.

Dans le futur, quelques améliorations pourraient être faite pour élargir l'utilisation du protocole de migration. Pour l'instant, la génération de code peut être faite uniquement sous Windows XP et le produit de migration peut être exécuté tant sous Windows XP que sous QNX. La génération de code pourrait être facilement étendue à des plateformes Linux ou Sun. D'autres cibles pourraient exister (Linux, Vx Works, etc.) afin que les produits de migration puissent être aussi portables que le CAR Microb.

De plus, il serait pratique d'être en mesure de modifier la période d'échantillonnage de la bibliothèque partagée sans avoir à retourner dans l'environnement Simulink. Il serait également souhaitable de générer un programme de test pour Windows pour faciliter l'intégration des nouvelles bibliothèques. Il pourrait être intéressant de générer une bibliothèque statique au lieu d'une bibliothèque dynamique dans certains cas, par exemple lorsque le temps de cycle requis est plus court.

Finalement, pour maximiser l'utilisation des experts Microb, il serait intéressant d'améliorer la génération automatique de code du CAR Microb pour le contrôleur global. Présentement, cette génération de code Microb est un mécanisme complexe et nécessite la participation active d'un expert. Un logiciel de génération automatique d'un

contrôleur global pour Microb permettant l'ajout de bibliothèques dynamique permettrait de libérer davantage les experts Microb.

ANNEXE 1

SONDAGE AUX USAGERS

Objectifs pour le protocole de migration

Selon vous, quelle importance devrait-on accorder aux qualités logicielles suivantes[1] pour le logiciel qui effectue la migration (a) et pour celui dont le code a été généré (b)? Classez-les par ordre d'importance.

(**note:** 1 = très importante; 6 = peu importante; 0 = non requise)

a:___ b:___ **Fonctionnalité**: Capacité du logiciel à remplir son mandat selon les besoins qui ont été énoncés au départ, autrement dit, de « Faire la job » [5] (ex: effectuer une migration).

a:___ b:___ **Fiabilité**: Propriété d'un système informatique capable d'assurer ses fonctions sans défaillance, dans des conditions préalablement définies et sur une période déterminée[6] (ex: éviter les pannes).

a:___ b:___ **Convivialité**: Qualité d'un matériel ou d'un logiciel qui est facile et agréable à utiliser et à comprendre, même par quelqu'un qui a peu de connaissances en informatique6 (ex: temps d'apprentissage).

a:___ b:___ **Efficacité**: Capacité du logiciel à performer adéquatement relativement aux ressources disponibles selon certaines conditions[6] (ex: temps réponse par cycle).

a:___ b:___ **Maintenabilité**: Capacité du logiciel à être modifié[5] (ex: être corrigé, amélioré, etc.).

a:___ b:___ **Portabilité**: Capacité d'un logiciel à être utilisé sur des systèmes informatiques de types différents[6] (ex: QNX, Windows, Linux, etc.).

---

[5] Définition inspirée de la norme ISO 9126 (2001), voir extrait au verso.
[6] Définition tirée du Grand dictionnaire terminologique (http://w3.granddictionnaire.com).

2. Afin de sélectionner la meilleure approche de migration, classez les objectifs suivants par ordre d'importance. Ajoutez des objectifs quantifiables s'il y a lieu.

(**note:** 1 = très important; 8 = peu important; 0 = non requis)

___ Minimiser le temps de cycle

___ Minimiser la mémoire vive utilisée

___ Minimiser le stockage utilisé

___ Minimiser l'utilisation du processeur

____ Minimiser le temps de migration

____ _____

____ _____

____ _____

3. Selon vous, quel serait le temps de migration total maximal, c'est-à-dire de Simulink jusqu'à l'exécution du code sur le contrôleur physique? Expliquez s'il y a lieu.

_____

4. Selon vous, quel serait le temps d'apprentissage maximal du protocole, c'est-à-dire le temps requis pour installer, comprendre et utiliser pleinement ce dernier? Expliquez s'il y a lieu.

_____

ANNEXE 2

QUESTIONNAIRE D'ÉVALUATION

Évaluation du protocole de migration

Instructions

Le document suivant a pour but d'évaluer le protocole de migration. Cette évaluation sera effectuée sur un ordinateur qui vous sera fourni et dont les logiciels prérequis sont tous fonctionnels. Afin que votre évaluation soit significative, assurez-vous de respecter les étapes suivantes:

1. Effectuez l'installation de la cible RTW et des fichiers Simulink. Cette opération doit être faite à chaque fois que le protocole est utilisé sur une nouvelle machine.

2. Suivez les étapes indiquées dans la section utilisation. Cette section reproduit une itération de conception typique;

3. Suivez les étapes indiquées dans la section vérification. Cette section sert à valider que votre migration a fonctionnée;

4. Répondez au questionnaire.

Merci!

Vincent Poiré

Procédure d'évaluation

Mise en contexte

Ce document a pour but d'effectuer une évaluation qualitative du protocole de migration par ses utilisateurs potentiels. L'évaluation se divise en deux étapes: tout d'abord, l'usager expérimente lui-même le protocole de migration puis ensuite répond à un questionnaire.

Rappelons que le protocole de migration a pour objectif d'implanter une loi de contrôle développée sous l'environnement Matlab/Simulink sur un contrôleur temps réel, appelé contrôleur global, développé avec la bibliothèque de développement en temps réel Microb.

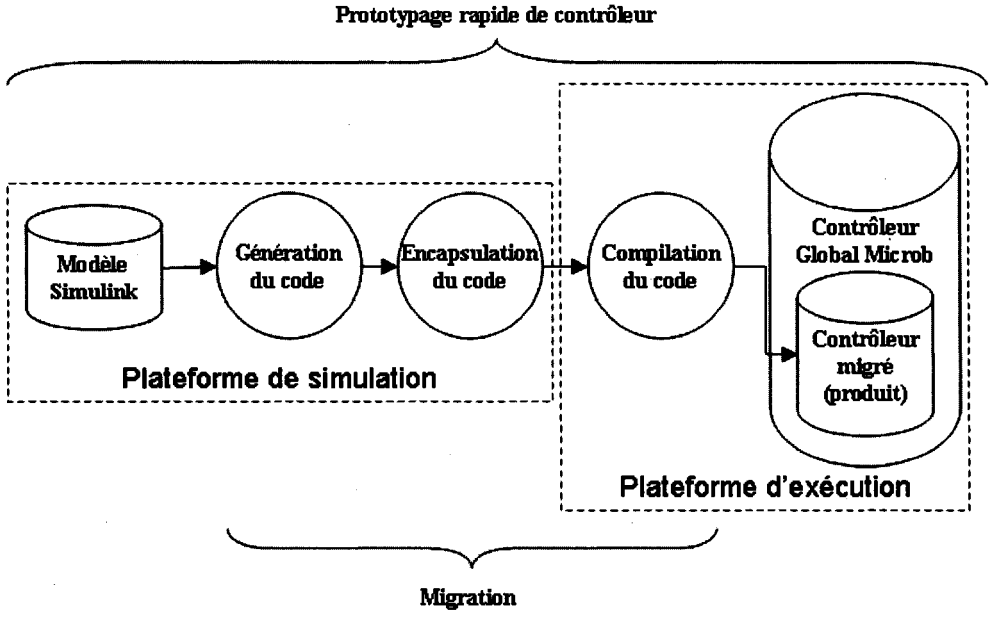

Illustration 1: Processus de migration

L'expérimentation du protocole par l'usager consiste à faire migrer un contrôleur à couple précalculé afin qu'il soit utilisé par le contrôleur global Bart. Le robot Bart, illustré à droite, est un robot à deux axes rotatifs présenté dans la documentation de Microb à titre d'exemple.

L'ordinateur d'évaluation sera le même pour tous les évaluateurs afin de standardiser l'expérience. Dans cette évaluation, la simulation et l'exécution s'effectuent sur le même ordinateur tournant sous Windows XP, ce qui permet d'effectuer la génération, l'encapsulation et la compilation du code en une seule étape dans l'environnement Simulink (voir Illustration 1). Le protocole de migration a déjà été initialisé par un expert Microb, c'est-à-dire que le code du contrôleur global a été modifié pour interagir avec une nouvelle loi de contrôle encapsulée dans une bibliothèque dynamique.

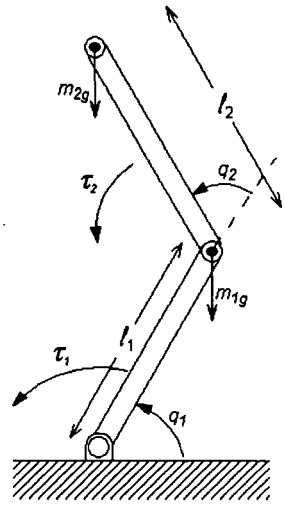

Illustration 2: Robot Bart

Le schéma de contrôle représentant le contrôleur à couple précalculé (voir Illustration 3) a préalablement été converti en un modèle Simulink prêt pour la migration (voir Illustration 4). L'évaluateur a donc la tâche de faire migrer le modèle Simulink (voir Illustration 4) vers le contrôleur global de Bart développé à l'aide de Microb.

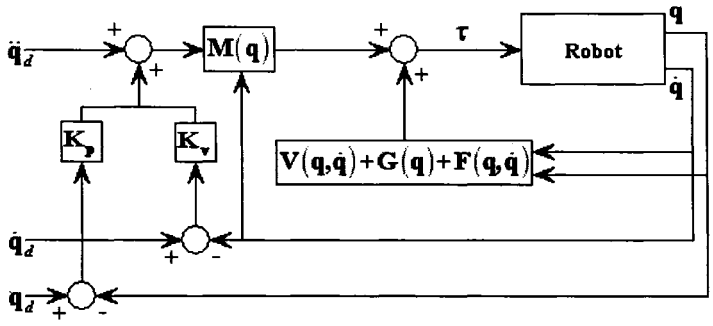

Illustration 3: Schéma de la loi de contrôle

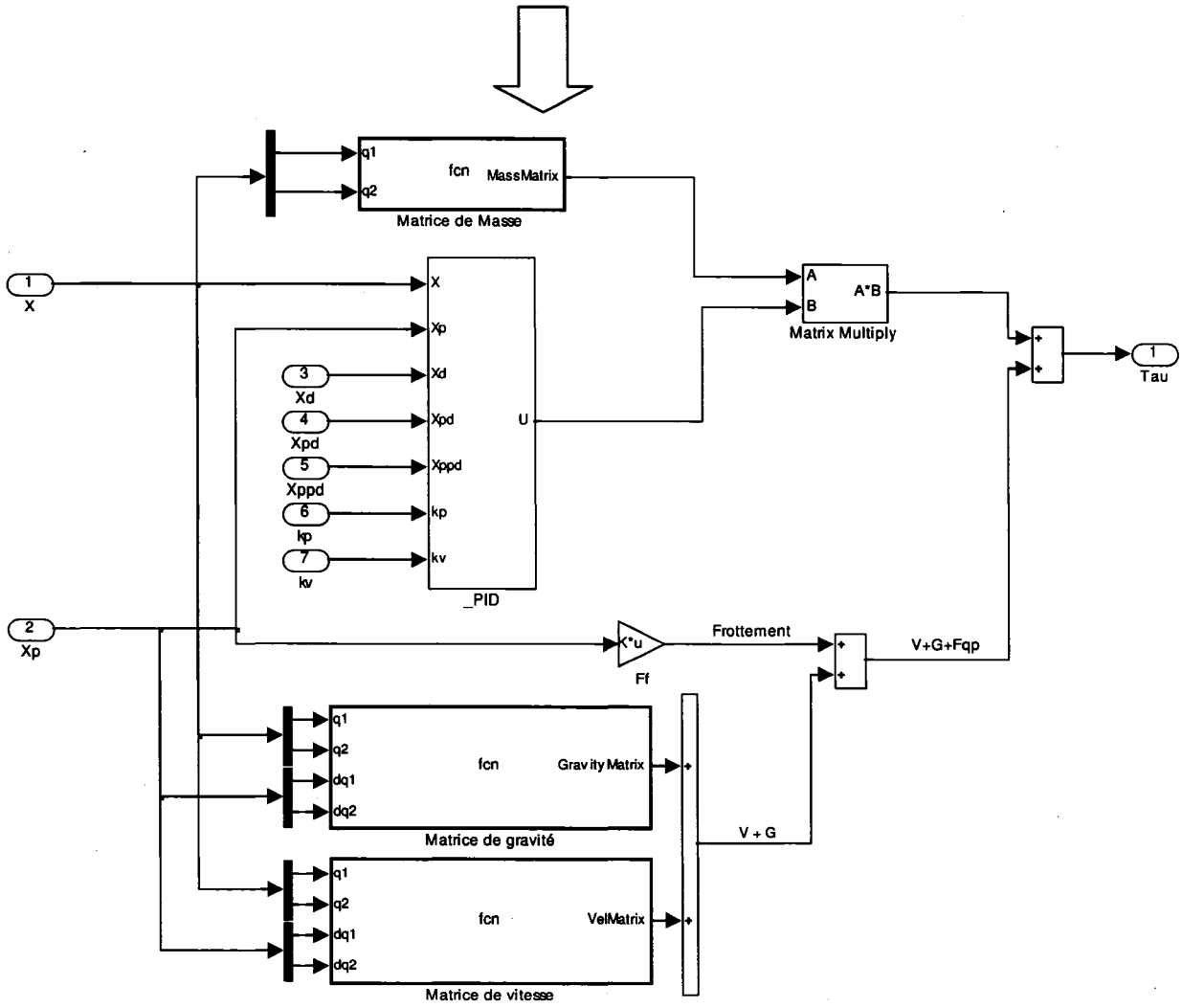

Illustration 4: Modèle Simu]]link résultant

Préalables

L'ordinateur qui sert à tester le protocole fonctionne sous Windows XP. Il doit avoir les logiciels suivants installés de façon fonctionnelle:

- Le cadre d'application robotique Microb (version mai 2006)
- Matlab (version 7.0.0.19920 (R14)) avec Real-Time Workshop (RTW 6.0)

Il semblerait que le protocole fonctionne avec d'autres versions mais aucun test approfondi n'a été fait pour valider cette hypothèse. Le robot Bart, basé sur l'exemple de la bibliothèque Microb, doit être également fonctionnel. Notez que l'ordinateur fourni pour l'évaluation a déjà tous les préalables nécessaires pour assurer le bon fonctionnement du protocole.

Installation

Installation de la cible RTW (win_dll)

Copier les deux nouvelles cibles RTW (*qnx_so* et *win_dll*) dans le répertoire des cibles RTW (*c:\Matlab7\rtw \c*).

Installation des fichiers Simulink (CTorque5)

Copier les fichiers Simulink (*CTorque5.mdl* et *init_Ctorque.m*) dans le répertoire de travail (*c:\Matlab7\work*).

Utilisation

1. Démarrer l'application Matlab.
2. Exécutez le fichier *init_CTorque.m.*
3. Ouvrir le modèle Simulink *CTorque5.mdl.*
4. Dans l'interface du modèle Simulink, allez dans le menu *Tools / Real-Time Workshop / Options*. Dans l'encadré *Target selection*, appuyez sur le bouton *Browse*. Choisissez dans la liste la cible *win_dll.tlc / Migration Windows (dll).* Pressez OK deux fois.
5. Lancez la compilation du modèle via le menu *Tools / Real-Time Workshop / Build Model.* La compilation du modèle devrait se terminer avec succès en affichant dans l'interface Matlab le texte suivant:

```
*** Created dynamic library: CTorque5.dll
### Successful completion of Real-Time Workshop build procedure for model: CTorque5
```

Vérification

1. Ouvrez une fenêtre de commande et exécutez le script *test.bat* situé à la racine du contrôleur Bart (*c:\Bart\test.bat*). Comme la bibliothèque dynamique n'a pas été copiée, le contrôleur de Bart n'est pas en mesure de la charger et retournera l'erreur suivante:

```
Bart server, $Id: server_bart.C,v 1.6 2003/08/01 21:46:24 remi Exp $
Copyright (c) 1999 Hydro-Quebec
Listenning to port 3791
Accepted connection from delta (192.168.0.102)
Accepted connection from delta (192.168.0.102)

Lancement du systeme dll
Initialisation du Compensateur dll
Chargement du dll...
La dll n'existe pas...
Erreur systeme: Le module spúcifiú est introuvable.
-> Erreur d'initialisation
```

2. Copier la bibliothèque dynamique (*c:\Matlab7\work\CTorque5_win_dll \CTorque5.dll*) dans le répertoire des exécutables de Bart (*c:\Bart\bin\WindowsNT\PentiumPro\CTorque5.dll*) en exécutant le fichier *copie_bibliothèque.bat* situé sur le bureau.

3. Relancer le script de test via une fenêtre de commande (*c:\Bart\test.bat*). Si l'exécution fonctionne bien, la fenêtre suivante devrait s'afficher:

```
Bart server, $Id: server_bart.C,v 1.6 2003/08/01 21:46:24 remi Exp $
Copyright (c) 1999 Hydro-Quebec
Listenning to port 3798
Accepted connection from delta (192.168.0.102)
Accepted connection from delta (192.168.0.102)

Lancement du systeme dll
Initialisation du Compensateur dll
Chargement du dll...
dll charge
debut d'appel des fonctions
fonction 1 initialisee
fonction 2 initialisee
fonction 3 initialisee
Initilisation du dll
Period (0.01), Multiple (1), Engine Id: (0)
-> Fermeture...
*** created CTorque5.mat ***

dll libre,
Controleur et DLL fermes.
```

Questionnaire d'évaluation

1. Est-ce que vous considérez que le protocole de migration remplit son mandat adéquatement? Si non, pourquoi?

2. Est-ce que vous saisissez bien ce que le protocole de migration fait? Si non, pourquoi?

3. Est-ce que vous avez trouvé l'apprentissage du protocole de migration aisé? Si non, pourquoi?

4. Est-ce que vous avez trouvé la migration facile à effectuer? Si non, pourquoi?

5. Considérez-vous Real-Time Workshop comme étant un générateur de code suffisamment fiable pour être utilisé dans le développement de contrôleurs de robot? Si non, pourquoi?

6. Le contrôleur à couple précalculé que vous avez migré cause une augmentation de 10% du temps de cycle par rapport au contrôleur codé manuellement. Considérez-vous cette hausse acceptable? Si non, pourquoi?

7. Considérez-vous que le temps de migration expérimenté comme satisfaisant? Si non, pourquoi?

8. Le contrôleur à couple précalculé que vous avez migré cause une augmentation d'environ 40% de la mémoire vive (RAM) utilisée par rapport au contrôleur codé manuellement. Considérez-vous cette hausse acceptable? Si non, pourquoi?

9. Le contrôleur à couple précalculé que vous avez migré cause une augmentation d'environ 10% de l'espace disque (ROM) utilisée par rapport au contrôleur codé manuellement. Considérez-vous cette hausse acceptable? Si non, pourquoi?

10. Avez-vous trouver l'installation du protocole suffisamment simple et claire? Si non, pourquoi?

11. Trouvez-vous que le protocole de migration s'intègre harmonieusement à l'environnement Matlab \ Simulink? Si non, pourquoi?

12. Avez-vous des suggestions pour améliorer le protocole, tant dans son produit que dans son utilisation?

13. Commentaires

Merci de votre collaboration!

ANNEXE 3

FICHIERS DE GÉNÉRATION WINDOWS (TLC)

Fichier win_dll.tlc permettant d'orchestrer la génération de code et d'appeler le fichier grt_shell.tlc listé plus bas.

```
%% SYSTLC: Migration Windows (dll) \
%% TMF: win_dll.tmf MAKE: make_rtw EXTMODE: ext_comm
%%
%% $RCSfile: grt.tlc,v $
%% $Revision: 1.31.4.7 $
%% $Date: 2004/04/14 23:43:44 $
%%
%% Copyright 1994-2004 The MathWorks, Inc.
%% Abstract: Generic real-time system target file.
%%
%selectfile NULL_FILE

%assign TargetType = "RT"
%assign Language = "C"
%assign GenRTModel = 1
%assign _GRT_  = 1

%assign MatFileLogging = 1

%include "codegenentry.tlc"


%% The contents between 'BEGIN_RTW_OPTIONS' and 'END_RTW_OPTIONS' are strictly
%% written by the standard format. We need to use this structure in RTW
%% options GUI function rtwoptionsdlg.m file.
%%
/%
 BEGIN_RTW_OPTIONS

 % second page of category item
 rtwoptions(1).prompt   = 'GRT code generation options';
 rtwoptions(1).type   = 'Category';
 rtwoptions(1).enable   = 'on';
 rtwoptions(1).default  = 2; % number of items under this category
       % excluding this one.
 rtwoptions(1).popupstrings = '';
 rtwoptions(1).tlcvariable = '';
 rtwoptions(1).tooltip  = '';
 rtwoptions(1).callback  = '';
 rtwoptions(1).opencallback = '';
 rtwoptions(1).closecallback = '';
 rtwoptions(1).makevariable = '';

 rtwoptions(2).prompt   = 'MAT-file variable name modifier';
 rtwoptions(2).type   = 'Popup';
 rtwoptions(2).default  = 'rt_';
 rtwoptions(2).popupstrings = 'rt_|_rt|none';
 rtwoptions(2).tlcvariable = 'LogVarNameModifier';
 rtwoptions(2).tooltip  = ...
 ['prefix rt_ to variable name,', sprintf('\n'), ...
 'append _rt to variable name,', sprintf('\n'), ...
 'or no modification'];

 rtwoptions(3).prompt  = 'Ignore custom storage classes';
 rtwoptions(3).type   = 'Checkbox';
 rtwoptions(3).default  = 'on';
 rtwoptions(3).tlcvariable = 'IgnoreCustomStorageClasses';
 rtwoptions(3).tooltip  = ['Treat custom storage classes as ''Auto''.'];
 rtwoptions(3).opencallback = [ ...
 'objTag = ''Ignore custom storage classes_CheckboxTag'';', ...
 'obj = findobj(DialogFig,''Tag'',objTag);', ...
 'set(obj, ''Enable'', sl(''onoff'',ecoderinstalled));'];

 rtwoptions(4).prompt   = 'External Mode code generation options';
 rtwoptions(4).type   = 'Category';
 rtwoptions(4).enable   = 'on';
 rtwoptions(4).default  = 5; % number of items under this category
```

```
        % excluding this one.
rtwoptions(4).popupstrings = '';
rtwoptions(4).tlcvariable = '';
rtwoptions(4).tooltip  = '';
rtwoptions(4).callback  = '';
rtwoptions(4).opencallback = '';
rtwoptions(4).closecallback = '';
rtwoptions(4).makevariable = '';

rtwoptions(5).prompt   = 'External mode';
rtwoptions(5).type   = 'Checkbox';
rtwoptions(5).default  = 'off';
rtwoptions(5).tlcvariable = 'ExtMode';
rtwoptions(5).makevariable = 'EXT_MODE';
rtwoptions(5).tooltip  = ...
['Adds communication support',sprintf('\n'), ...
'for use with Simulink external mode'];

% Enable/disable other external mode controls.
rtwoptions(5).callback  = [ ...
'DialogFig = get(gcbo,''Parent'');',...
'sl(''extmodecallback'', ''extmode_checkbox_callback'', DialogFig);', ...
];

rtwoptions(6).prompt   = 'Transport';
rtwoptions(6).type   = 'Popup';
rtwoptions(6).default  = 'tcpip';
rtwoptions(6).popupstrings = ['tcpip|', ...
        'serial_win32'];
rtwoptions(6).tlcvariable = 'ExtModeTransport';
rtwoptions(6).makevariable = 'EXTMODE_TRANSPORT';
rtwoptions(6).tooltip  = ...
['Chooses transport mechanism for external mode'];

% Synchronize with "External mode" checkbox option
rtwoptions(6).opencallback = [ ...
'ExtModeTable = {''tcpip''   ''ext_comm'';', ...
     '''serial_win32'' ''ext_serial_win32_comm''};', ...
'ud = DialogUserData;', ...
'ud =
extmodecallback(''transport_popup_opencallback'',model,DialogFig,ud,ExtModeTable);', ...
'DialogUserData = ud;', ...
];

% Set extmode mex-file according to extmode transport mechanism.
rtwoptions(6).closecallback = [ ...
'ExtModeTable = {''tcpip''   ''ext_comm'';', ...
    '''serial_win32'' ''ext_serial_win32_comm''};', ...
'ud = DialogUserData;', ...
'ud =
extmodecallback(''transport_popup_closecallback'',model,DialogFig,ud,ExtModeTable);', ...
'DialogUserData = ud;', ...
];

rtwoptions(7).prompt   = 'Static memory allocation';
rtwoptions(7).type   = 'Checkbox';
rtwoptions(7).default  = 'off';
rtwoptions(7).tlcvariable = 'ExtModeStaticAlloc';
rtwoptions(7).makevariable = 'EXTMODE_STATIC_ALLOC';
rtwoptions(7).tooltip  = ...
['Forces external mode to use static',sprintf('\n'), ...
'instead of dynamic memory allocation'];

% Enable/disable external mode static allocation size selection.
rtwoptions(7).callback  = [ ...
'DialogFig = get(gcbo,''Parent'');',...
'sl(''extmodecallback'', ''staticmem_checkbox_callback'', DialogFig);', ...
];

% Synchronize with "External mode" checkbox option
rtwoptions(7).opencallback = [ ...
'extmodecallback(''staticmem_checkbox_opencallback'',DialogFig);', ...
];

rtwoptions(8).prompt   = 'Static memory buffer size';
```

```
rtwoptions(8).type   = 'Edit';
rtwoptions(8).default  = '1000000';
rtwoptions(8).tlcvariable = 'ExtModeStaticAllocSize';
rtwoptions(8).makevariable = 'EXTMODE_STATIC_ALLOC_SIZE';
rtwoptions(8).tooltip  = ...
['Size of external mode static allocation buffer'];

% Synchronize with "External mode static allocation" option
rtwoptions(8).opencallback = [ ...
'extmodecallback(''staticmemsize_edit_opencallback'',DialogFig);', ...
];

rtwoptions(9).prompt  = 'External mode testing';
rtwoptions(9).type   = 'NonUI';
rtwoptions(9).default  = '0';
rtwoptions(9).tlcvariable = 'ExtModeTesting';
rtwoptions(9).makevariable = 'TMW_EXTMODE_TESTING';
rtwoptions(9).tooltip  = ...
['Internal testing flag for Simulink external mode'];

%------------------------------------%
% Configure RTW code generation settings %
%------------------------------------%

rtwgensettings.BuildDirSuffix = '_win_dll';

END_RTW_OPTIONS
%/

/%
BEGIN_CONFIGSET_TARGET_COMPONENT

targetComponentClass = 'Simulink.GRTTargetCC';

END_CONFIGSET_TARGET_COMPONENT
%/

%include "grt_shell.tlc"
```

Fichier grt_shell.tlc pour permettre de contrôler l'exécution de la bibliothèque dynamique d'un programme externe.

```
%%selectfile STDOUT %% Stream de la fen^tre de Matlab
%%G,n,rer le fichier
%openfile foo = "grt_shell.c"
%selectfile foo
/*
* Modified version of grt_main.c
* Modifications made by Roland Pfeiffer and Vincent Poir,
* File generated from grt_shell.tlc
*/
/* $Revision: 1.68.4.6 $
* Copyright 1994-2004 The MathWorks, Inc.
*
* File: grt_main.c.tmp
*
* Abstract:
* A Generic "Real-Time (single tasking or pseudo-multitasking,
* statically allocated data)" main that runs under most
* operating systems.

*
* This file may be a useful starting point when targeting a new
* processor or microcontroller.
*
*
* Compiler specified defines:
* RT - Required.
* MODEL=modelname - Required.
```

```
* NUMST=# - Required. Number of sample times.
* NCSTATES=# - Required. Number of continuous states.
* TID01EQ=1 or 0 - Optional. Only define to 1 if sample time task
* id's 0 and 1 have equal rates.
* MULTITASKING - Optional. (use MT for a synonym).
* SAVEFILE - Optional (non-quoted) name of .mat file to create.
* Default is <MODEL>.mat
* BORLAND - Required if using Borland C/C++
*/
#include <float.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include "tmwtypes.h"
# include "rtmodel.h"
#include "rt_sim.h"
#include "rt_logging.h"
#ifdef UseMMIDataLogging
#include "rt_logging_mmi.h"
#endif
#include "rt_nonfinite.h"
/* Signal Handler header */
#ifdef BORLAND
#include <signal.h>
#include <float.h>
#endif
#include "ext_work.h"
/*=========*
* Defines *
*=========*/
#ifndef TRUE
#define FALSE (0)
#define TRUE (1)
#endif
#ifndef EXIT_FAILURE
#define EXIT_FAILURE 1
#endif
#ifndef EXIT_SUCCESS
#define EXIT_SUCCESS 0
#endif
#define QUOTE1(name) #name
#define QUOTE(name) QUOTE1(name) /* need to expand name */
#ifndef RT
# error "must define RT"
#endif
#ifndef MODEL
# error "must define MODEL"
#endif
#ifndef NUMST
# error "must define number of sample times, NUMST"
#endif
#ifndef NCSTATES
# error "must define NCSTATES"
#endif
#ifndef SAVEFILE
# define MATFILE2(file) #file ".mat"
# define MATFILE1(file) MATFILE2(file)
# define MATFILE MATFILE1(MODEL)
#else
# define MATFILE QUOTE(SAVEFILE)
#endif
#define RUN_FOREVER -1.0
#define EXPAND_CONCAT(name1,name2) name1 ## name2
#define CONCAT(name1,name2) EXPAND_CONCAT(name1,name2)
#define RT_MODEL CONCAT(MODEL,_rtModel)
/*====================*
* External functions *
*====================*/
extern RT_MODEL *MODEL(void);
extern void MdlInitializeSizes(void);
extern void MdlInitializeSampleTimes(void);
extern void MdlStart(void);
extern void MdlOutputs(int_T tid);
extern void MdlUpdate(int_T tid);
```

```
extern void MdlTerminate(void);
#if NCSTATES > 0
extern void rt_ODECreateIntegrationData(RTWSolverInfo *si);
extern void rt_ODEUpdateContinuousStates(RTWSolverInfo *si);
# define rt_CreateIntegrationData(S) \
rt_ODECreateIntegrationData(rtmGetRTWSolverInfo(S));
# define rt_UpdateContinuousStates(S) \
rt_ODEUpdateContinuousStates(rtmGetRTWSolverInfo(S));
# else
# define rt_CreateIntegrationData(S) \
rtsiSetSolverName(rtmGetRTWSolverInfo(S),"FixedStepDiscrete");
# define rt_UpdateContinuousStates(S) \
rtmSetT(S, rtsiGetSolverStopTime(rtmGetRTWSolverInfo(S)));
#endif
/*==================================*
* Global data local to this module *
*==================================*/

// Export functions
__declspec(dllexport) char* initiateController();
__declspec(dllexport) char* getControllerOutput(int nbrInputArgs, double* inputArgs,int
nbrOutputArgs, double* outputArgs);
__declspec(dllexport) char* performCleanup();

static struct {
int_T stopExecutionFlag;
int_T isrOverrun;
int_T overrunFlags[NUMST];
const char_T *errmsg;
} GBLbuf;
static char errorMsg[200];
#ifdef EXT_MODE
# define rtExtModeSingleTaskUpload(S) \
{ \
int stIdx; \
rtExtModeUploadCheckTrigger(rtmGetNumSampleTimes(S)); \
for (stIdx=0; stIdx<NUMST; stIdx++) { \
if (rtmIsSampleHit(S, stIdx, 0 /*unused*/)) { \
rtExtModeUpload(stIdx,rtmGetTaskTime(S,stIdx)); \
} \
} \
}
#else
# define rtExtModeSingleTaskUpload(S) /* Do nothing */
#endif
/*=================*
* Local functions *
*=================*/
#ifdef BORLAND
/* Implemented for BC++ only*/
typedef void (*fptr)(int, int);
/* Function: divideByZero ======================================================
*
* Abstract: Traps the error Division by zero and prints a warning
* Also catches other FP errors, but does not identify them
* specifically.
*/
void divideByZero(int sigName, int sigType)
{
signal(SIGFPE, (fptr)divideByZero);
if ((sigType == FPE_ZERODIVIDE)||(sigType == FPE_INTDIV0)){
sprintf(errorMsg, "Warning: Division by zero");
return;
}
else{
sprintf(errorMsg, "Warning: Floating Point error");
return;
}
} /* end divideByZero */
#endif /* BORLAND */
/*
* Calculate the controller output. Supply input arguments.
*
* Also supply a pointer to an array capable of containing the output values
*/
```

```
static RT_MODEL *S;
char* getControllerOutput(int nbrInputArgs, double* inputArgs,int nbrOutputArgs, double*
outputArgs) {
real_T tnext;
%with CompiledModel
       /*
       My model is called %<Name>.
       It was generated on %<GeneratedOn>.
       It has %<ExternalOutputs.NumExternalOutputs> external output(s),
%<ExternalInputs.NumExternalInputs> external input(s) and %<NumContStates> continuous
state(s).
       Output:%<tOutput>, %<tOutputType>
       Input:%<tInput>, %<tInputType>
       */

       if (nbrInputArgs != %<ExternalInputs.NumExternalInputs>) {
              sprintf(errorMsg,"nbrInputArgs should be
%<ExternalInputs.NumExternalInputs> but was %d", nbrInputArgs);
              return errorMsg;
       }
       if (nbrOutputArgs != %<ExternalOutputs.NumExternalOutputs>) {
              sprintf(errorMsg, "nbrOutputArgs should be
%<ExternalOutputs.NumExternalOutputs> but was %d", nbrOutputArgs);
              return errorMsg;
       }

       %with ExternalInputs
              %assign inputCount=0
              %foreach inp = NumExternalInputs
                     // Input # %<inp> '%<ExternalInput[inp].Identifier>' width:
%<ExternalInput[inp].Width>
                     %if ExternalInput[inp].Width == 1
                            %<tInput>%<IOQualifier>%<ExternalInput[inp].Identifier> =
inputArgs[%<inputCount>];
                            %assign inputCount=inputCount+1
                     %else
                     %foreach iw = ExternalInput[inp].Width

       %<tInput>%<IOQualifier>%<ExternalInput[inp].Identifier>[%<iw>] =
inputArgs[%<inputCount>];
                                   %assign inputCount=inputCount+1
                            %endforeach
                     %endif
              %endforeach
       %endwith
       /*********************************************
* Check and see if base step time is too fast *
*********************************************/
if (GBLbuf.isrOverrun++) {
GBLbuf.stopExecutionFlag = 1;
sprintf(errorMsg, "GBLbuf.isrOverrun");
return errorMsg;
}
/*********************************************
* Check and see if error status has been set *
*********************************************/
if (rtmGetErrorStatus(S) != NULL) {
GBLbuf.stopExecutionFlag = 1;
sprintf(errorMsg, "rtmGetErrorStatus(S) != null");
return errorMsg;
}
/* enable interrupts here */
/*
* In a multi-tasking environment, this would be removed from the base rate
* and called as a "background" task.
*/
rtExtModeOneStep(rtmGetRTWExtModeInfo(S),
rtmGetNumSampleTimes(S),
(boolean_T *)&rtmGetStopRequested(S));
tnext = rt_SimGetNextSampleHit();
rtsiSetSolverStopTime(rtmGetRTWSolverInfo(S),tnext);
MdlOutputs(0);
rtExtModeSingleTaskUpload(S);
GBLbuf.errmsg = rt_UpdateTXYLogVars(rtmGetRTWLogInfo(S),
rtmGetTPtr(S));
```

```
if (GBLbuf.errmsg != NULL) {
GBLbuf.stopExecutionFlag = 1;
sprintf(errorMsg, "GBLbuf.errmsg != null");
return errorMsg;
}
MdlUpdate(0);
rt_SimUpdateDiscreteTaskSampleHits(rtmGetNumSampleTimes(S),
rtmGetTimingData(S),
rtmGetSampleHitPtr(S),
rtmGetTPtr(S));
if (rtmGetSampleTime(S,0) == CONTINUOUS_SAMPLE_TIME) {
rt_UpdateContinuousStates(S);
}
GBLbuf.isrOverrun--;
rtExtModeCheckEndTrigger();

        %with ExternalOutputs
                %assign outputCount=0
                %foreach outp = NumExternalOutputs
                        // Output # %<outp>
'%<System[ExternalOutput[outp].Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>
' width: %<ExternalOutput[outp].Width>
                        %%outputArgs[%<outp>] =
%<tOutput>%<IOQualifier>%<System[0].Block[ExternalOutput[outp].Block[1]].Identifier>;
                        %if ExternalOutput[outp].Width == 1
                                outputArgs[%<outputCount>] =
%<tOutput>%<IOQualifier>%<System[ExternalOutput[outp].Block[0]].Block[ExternalOutput[outp
].Block[1]].Identifier>;
                                %assign outputCount=outputCount+1
                        %else
                                %foreach ow = ExternalOutput[outp].Width

        outputArgs[%<outputCount>]=%<tOutput>%<IOQualifier>%<System[ExternalOutput[outp].
Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>[%<ow>];
                                        %assign outputCount=outputCount+1
                                %endforeach
                        %endif
                %endforeach
        %endwith
%endwith

return errorMsg;
}
/* Function: initiateController
================================================================
*
* Abstract:
* Execute model on a generic target such as a workstation.
*/
char* initiateController() {
const char *status;
/*****************************
* MathError Handling for BC++ *
*****************************/
#ifdef BORLAND
signal(SIGFPE, (fptr)divideByZero);
#endif
{
/***************************
* Initialize global memory *
***************************/
rtExtModeParseArgs(argc, argv, NULL);
(void)memset(&GBLbuf, 0, sizeof(GBLbuf));
/************************
* Initialize the model *
************************/
rt_InitInfAndNaN(sizeof(real_T));
S = MODEL();
if (rtmGetErrorStatus(S) != NULL) {
(void)fprintf(stderr,"Error during model registration: %s\n",
rtmGetErrorStatus(S));
exit(EXIT_FAILURE);
sprintf(errorMsg, "Error during model registration: %s", rtmGetErrorStatus(S));
return(errorMsg);
}
```

```
rtmSetTFinal(S,RUN_FOREVER);
MdlInitializeSizes();
MdlInitializeSampleTimes();
status = rt_SimInitTimingEngine(rtmGetNumSampleTimes(S),
rtmGetStepSize(S),
rtmGetSampleTimePtr(S),
rtmGetOffsetTimePtr(S),
rtmGetSampleHitPtr(S),
rtmGetSampleTimeTaskIDPtr(S),
rtmGetTStart(S),
&rtmGetSimTimeStep(S),
&rtmGetTimingData(S));
if (status != NULL) {
(void)fprintf(stderr,
"Failed to initialize sample time engine: %s\n", status);
exit(EXIT_FAILURE);
sprintf(errorMsg, "Failed to initialize sample time engine: %s", status);
return errorMsg;
}
rt_CreateIntegrationData(S);
GBLbuf.errmsg = rt_StartDataLogging(rtmGetRTWLogInfo(S),
rtmGetTFinal(S),
rtmGetStepSize(S),
&rtmGetErrorStatus(S));
rtExtModeCheckInit(rtmGetNumSampleTimes(S));
rtExtModeWaitForStartPkt(rtmGetRTWExtModeInfo(S),
rtmGetNumSampleTimes(S),
(boolean_T *)&rtmGetStopRequested(S));
MdlStart();
if (rtmGetErrorStatus(S) != NULL) {
GBLbuf.stopExecutionFlag = 1;
}
errorMsg[0] = 0;
return errorMsg;
/*
Initialization should be finished at this point
*/
}
}
char* performCleanup() {
#ifdef UseMMIDataLogging
rt_CleanUpForStateLogWithMMI(rtmGetRTWLogInfo(S));
#endif
rt_StopDataLogging(MATFILE,rtmGetRTWLogInfo(S));
rtExtModeShutdown(rtmGetNumSampleTimes(S));
if (GBLbuf.errmsg) {
(void)fprintf(stderr,"%s\n",GBLbuf.errmsg);
exit(EXIT_FAILURE);
sprintf(errorMsg, "%s", GBLbuf.errmsg);
return errorMsg;
}
if (GBLbuf.isrOverrun) {
(void)fprintf(stderr,
"%s: ISR overrun - base sampling rate is too fast\n",
QUOTE(MODEL));
exit(EXIT_FAILURE);
sprintf(errorMsg, "%s: ISR overrun - base sampling rate is too fast",QUOTE(MODEL));
return errorMsg;
}
if (rtmGetErrorStatus(S) != NULL) {
(void)fprintf(stderr,"%s\n", rtmGetErrorStatus(S));
exit(EXIT_FAILURE);
sprintf(errorMsg,"%s", rtmGetErrorStatus(S));
return errorMsg;
}
return errorMsg;
}
BOOL WINAPI __declspec(dllexport) LibMain(HINSTANCE hDLLInst,DWORD fdwReason,LPVOID
lpvReserved)
{
return 1;
}
%closefile foo
%% Sélectionner le stream NULL
%selectfile NULL_FILE
```

ANNEXE 4

FICHIERS DE COMPILATION WINDOWS (TMF)

Le fichier win_dll.tmf permet de générer un fichier de compilation utilisé par la commande Build de Real-Time Workshop. Il a été adapté du fichier grt.tlc fourni avec Real-Time Workshop [45].

```
# Copyright 1994-2004 The MathWorks, Inc.
#
# File: win_dll.tmf $Revision: 1.26.4.14 $
#
# Abstract:
#       Real-Time Workshop template makefile for building a PC-based
#       stand-alone generic real-time version of Simulink model using
#       generated C code and
#                       LCC compiler Version 2.4
#
#       This makefile attempts to conform to the guidelines specified in the
#       IEEE Std 1003.2-1992 (POSIX) standard. It is designed to be used
#  with GNU Make (gmake) which is located in matlabroot/rtw/bin/win32.
#
#       Note that this template is automatically customized by the Real-Time
#       Workshop build procedure to create "<model>.mk"
#
#  The following defines can be used to modify the behavior of the
#       build:
#        OPT_OPTS  - Optimization options. Default is none. To enable
#        debugging specify as OPT_OPTS=-g4.
#        OPTS    - User specific compile options.
#        USER_SRCS  - Additional user sources, such as files needed by
#                       S-functions.
#        USER_INCLUDES - Additional include paths
#                       (i.e. USER_INCLUDES="-Iwhere-ever -Iwhere-ever2")
#                       (For Lcc, have a '/'as file seperator before the
#                       file name instead of a '\' .
#                       i.e., d:\work\proj1/myfile.c - reqd for 'gmake')
#
#  This template makefile is designed to be used with a system target
#  file that contains 'rtwgensettings.BuildDirSuffix' see grt.tlc

#------------------------ Macros read by make_rtw ------------------------------
#
# The following macros are read by the Real-Time Workshop build procedure:
#
# MAKECMD    - This is the command used to invoke the make utility
# HOST    - What platform this template makefile is targeted for
#     (i.e. PC or UNIX)
# BUILD    - Invoke make from the Real-Time Workshop build procedure
#     (yes/no)?
# SYS_TARGET_FILE - Name of system target file.

MAKECMD    = "%MATLAB%\rtw\bin\win32\gmake"
HOST    = PC
BUILD    = yes
SYS_TARGET_FILE = win_dll.tlc
BUILD_SUCCESS  = *** Created

#---------------------- Tokens expanded by make_rtw ----------------------------
#
# The following tokens, when wrapped with "|>" and "<|" are expanded by the
# Real-Time Workshop build procedure.
#
# MODEL_NAME    - Name of the Simulink block diagram
# MODEL_MODULES  - Any additional generated source modules
# MAKEFILE_NAME  - Name of makefile created from template makefile <model>.mk
# MATLAB_ROOT    - Path to were MATLAB is installed.
# MATLAB_BIN    - Path to MATLAB executable.
# S_FUNCTIONS    - List of S-functions.
# S_FUNCTIONS_LIB  - List of S-functions libraries to link.
# SOLVER     - Solver source file name
# NUMST     - Number of sample times
```

```
# TID01EQ    - yes (1) or no (0): Are sampling rates of continuous task
#      (tid=0) and 1st discrete task equal.
# NCSTATES   - Number of continuous states
# BUILDARGS   - Options passed in at the command line.
# MULTITASKING  - yes (1) or no (0): Is solver mode multitasking
# EXT_MODE   - yes (1) or no (0): Build for external mode
# TMW_EXTMODE_TESTING - yes (1) or no (0): Build ext_test.c for external mode
#      testing.
# EXTMODE_TRANSPORT - Index of transport mechanism (e.g. tcpip, serial) for extmode
# EXTMODE_STATIC  - yes (1) or no (0): Use static instead of dynamic mem alloc.
# EXTMODE_STATIC_SIZE - Size of static memory allocation buffer.


MODEL    = |>MODEL_NAME<|
MODULES    = |>MODEL_MODULES<|
MAKEFILE    = |>MAKEFILE_NAME<|
MATLAB_ROOT    = |>MATLAB_ROOT<|
MATLAB_BIN    = |>MATLAB_BIN<|
S_FUNCTIONS    = |>S_FUNCTIONS<|
S_FUNCTIONS_LIB  = |>S_FUNCTIONS_LIB<|
SOLVER    = |>SOLVER<|
NUMST    = |>NUMST<|
TID01EQ    = |>TID01EQ<|
NCSTATES    = |>NCSTATES<|
BUILDARGS    = |>BUILDARGS<|
MULTITASKING    = |>MULTITASKING<|
EXT_MODE    = |>EXT_MODE<|
TMW_EXTMODE_TESTING = |>TMW_EXTMODE_TESTING<|
EXTMODE_TRANSPORT = |>EXTMODE_TRANSPORT<|
EXTMODE_STATIC  = |>EXTMODE_STATIC_ALLOC<|
EXTMODE_STATIC_SIZE = |>EXTMODE_STATIC_ALLOC_SIZE<|

MODELREFS    = |>MODELREFS<|
SHARED_SRC    = |>SHARED_SRC<|
SHARED_SRC_DIR    = |>SHARED_SRC_DIR<|
SHARED_BIN_DIR    = |>SHARED_BIN_DIR<|
SHARED_LIB    = |>SHARED_LIB<|

#--------------------------- Model and reference models -----------------------
MODELLIB     = |>MODELLIB<|
MODELREF_LINK_LIBS  = |>MODELREF_LINK_LIBS<|
MODELREF_INC_PATH    = |>START_MDLREFINC_EXPAND_INCLUDES<|-I|>MODELREF_INC_PATH<|
|>END_MDLREFINC_EXPAND_INCLUDES<|
RELATIVE_PATH_TO_ANCHOR = |>RELATIVE_PATH_TO_ANCHOR<|
# NONE: standalone, SIM: modelref sim, RTW: modelref rtw
MODELREF_TARGET_TYPE  = |>MODELREF_TARGET_TYPE<|

#--------------------------- Tool Specifications -------------------------------

LCC = $(MATLAB_ROOT)\sys\lcc
include $(MATLAB_ROOT)\rtw\c\tools\lcctools.mak

#------------------------------ External mode ---------------------------------

# Uncomment -DVERBOSE to have information printed to stdout
# To add a new transport layer, see the comments in
# <matlabroot>/toolbox/Simulink/Simulink/extmode_transports.m
ifeq ($(EXT_MODE),1)
 EXT_CC_OPTS = -DEXT_MODE -DWIN32 # -DVERBOSE
 ifeq ($(EXTMODE_TRANSPORT),0) #tcpip
 EXT_SRC = ext_svr.c updown.c ext_work.c ext_svr_tcpip_transport.c
 EXT_LIB = $(MATLAB_ROOT)\sys\lcc\lib\wsock32.lib
 endif
 ifeq ($(EXTMODE_TRANSPORT),1) #serial_win32
 EXT_SRC = ext_svr.c updown.c ext_work.c ext_svr_serial_transport.c
 EXT_SRC += ext_serial_pkt.c ext_serial_win32_port.c
 endif
 ifeq ($(TMW_EXTMODE_TESTING),1)
 EXT_SRC  += ext_test.c
 EXT_CC_OPTS += -DTMW_EXTMODE_TESTING
 endif
 ifeq ($(EXTMODE_STATIC),1)
 EXT_SRC  += mem_mgr.c
 EXT_CC_OPTS += -DEXTMODE_STATIC -DEXTMODE_STATIC_SIZE=$(EXTMODE_STATIC_SIZE)
 endif
```

```
endif

#---------------------------- Include Path ---------------------------------

# see MATLAB_INCLUDES and COMPILER_INCLUDES from lcctools.mak

ADD_INCLUDES = \
|>START_EXPAND_INCLUDES<|     -I|>EXPAND_DIR_NAME<| \
|>END_EXPAND_INCLUDES<|

SHARED_INCLUDES =
ifneq ($(SHARED_SRC_DIR),)
SHARED_INCLUDES = -I$(SHARED_SRC_DIR)
endif

INCLUDES = -I. -I$(RELATIVE_PATH_TO_ANCHOR) $(MATLAB_INCLUDES) $(ADD_INCLUDES) \
   $(COMPILER_INCLUDES) $(USER_INCLUDES) $(MODELREF_INC_PATH) $(SHARED_INCLUDES)

#----------------------- rtModel ----------------------------------------------
RTM_CC_OPTS = -DUSE_RTMODEL

#------------------------------- C Flags -------------------------------------

# Optimization Options
OPT_OPTS = $(DEFAULT_OPT_OPTS)

# General User Options
OPTS =

# Compiler options, etc:
CC_OPTS = $(OPT_OPTS) $(OPTS) $(ANSI_OPTS) $(EXT_CC_OPTS) $(RTM_CC_OPTS)

CPP_REQ_DEFINES = -DMODEL=$(MODEL) -DRT -DNUMST=$(NUMST) \
     -DTID01EQ=$(TID01EQ) -DNCSTATES=$(NCSTATES) \
     -DMT=$(MULTITASKING) -DHAVESTDIO

CFLAGS = $(CC_OPTS) $(CPP_REQ_DEFINES) $(INCLUDES) -noregistrylookup

ifeq ($(OPT_OPTS),$(DEFAULT_OPT_OPTS))
LDFLAGS = -s -L$(LIB)
else
LDFLAGS = -L$(LIB)
endif

#------------------------ Additional Libraries ------------------------------

LIBS =
|>START_PRECOMP_LIBRARIES<|
ifeq ($(OPT_OPTS),$(DEFAULT_OPT_OPTS))
LIBS += |>EXPAND_LIBRARY_LOCATION<|\|>EXPAND_LIBRARY_NAME<|_lcc.lib
else
LIBS += |>EXPAND_LIBRARY_NAME<|.lib
endif |>END_PRECOMP_LIBRARIES<|
|>START_EXPAND_LIBRARIES<|
LIBS += |>EXPAND_LIBRARY_NAME<|.lib |>END_EXPAND_LIBRARIES<|
LIBS += $(EXT_LIB) $(S_FUNCTIONS_LIB)

#---------------------------- Source Files -----------------------------------

ifeq ($(MODELREF_TARGET_TYPE), NONE)
 PRODUCT   = $(RELATIVE_PATH_TO_ANCHOR)/$(MODEL).exe
 BIN_SETTING  = $(LD) $(LDFLAGS) -dll $(SYSLIBS)
 BUILD_PRODUCT_TYPE = executable
 REQ_SRCS   = $(MODEL).c $(MODULES) $(EXT_SRC) \
       rt_sim.c
else
 # Model reference rtw target
 PRODUCT = $(MODELLIB)
 REQ_SRCS = $(MODULES)
endif

USER_SRCS =

USER_OBJS  = $(USER_SRCS:.c=.obj)
LOCAL_USER_OBJS = $(notdir $(USER_OBJS))
```

```
SRCS  = $(REQ_SRCS) $(S_FUNCTIONS) $(SOLVER)
OBJS  = $(SRCS:.c=.obj) $(USER_OBJS)
LINK_OBJS = $(SRCS:.c=.obj) $(LOCAL_USER_OBJS)

SHARED_OBJS:= $(addsuffix .obj, $(basename $(wildcard $(SHARED_SRC))))
FMT_SHARED_OBJS = $(subst /,\,$(SHARED_OBJS))

#---------------------------------- Rules --------------------------------------

ifeq ($(MODELREF_TARGET_TYPE),NONE)
$(PRODUCT): $(OBJS) $(SHARED_LIB) $(LIBS) $(MODELREF_LINK_LIBS)
        $(BIN_SETTING) $(LINK_OBJS) $(MODELREF_LINK_LIBS) $(SHARED_LIB) $(LIBS)
        @echo $(BUILD_SUCCESS) dynamic library: $(MODEL).dll
else

$(PRODUCT): $(OBJS) $(SHARED_LIB) $(LIBS)
        $(LIBCMD) /out:$(MODELLIB) $(LINK_OBJS)
        @echo $(BUILD_SUCCESS) static library $(MODELLIB)
endif

%.obj: %.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

%.obj: $(RELATIVE_PATH_TO_ANCHOR)/%.c
        $(CC) -c -Fo$(@F) -I$(RELATIVE_PATH_TO_ANCHOR)/$(<F:.c=cn_rtw) $(CFLAGS) $<

%.obj: $(MATLAB_ROOT)/rtw/c/grt/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

%.obj: $(MATLAB_ROOT)/rtw/c/src/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

%.obj: $(MATLAB_ROOT)/rtw/c/src/ext_mode/common/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

%.obj: $(MATLAB_ROOT)/rtw/c/src/ext_mode/tcpip/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

%.obj: $(MATLAB_ROOT)/rtw/c/src/ext_mode/serial/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

%.obj: $(MATLAB_ROOT)/rtw/c/src/ext_mode/custom/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

|>START_EXPAND_RULES<|%.obj: |>EXPAND_DIR_NAME<|/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

|>END_EXPAND_RULES<|

%.obj: $(MATLAB_ROOT)/Simulink/src/%.c
        $(CC) -c -Fo$(@F) $(CFLAGS) $<

# Libraries:

|>START_EXPAND_LIBRARIES<|MODULES_|>EXPAND_LIBRARY_NAME<| = \
|>START_EXPAND_MODULES<|      |>EXPAND_MODULE_NAME<|.obj \
|>END_EXPAND_MODULES<|

|>EXPAND_LIBRARY_NAME<|.lib: $(MAKEFILE) rtw_proj.tmw $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo ### Creating $@
        @if exist $@ del $@
        $(LIBCMD) /out:$@ $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo ### $@ Created

|>END_EXPAND_LIBRARIES<|

|>START_PRECOMP_LIBRARIES<|MODULES_|>EXPAND_LIBRARY_NAME<| = \
|>START_EXPAND_MODULES<|      |>EXPAND_MODULE_NAME<|.obj \
|>END_EXPAND_MODULES<|

|>EXPAND_LIBRARY_NAME<|.lib: $(MAKEFILE) rtw_proj.tmw $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo ### Creating $@
        @if exist $@ del $@
        $(LIBCMD) /out:$@ $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo ### $@ Created
```

```
|>END_PRECOMP_LIBRARIES<|

#----------------------------- Dependencies -------------------------------------

$(OBJS): $(MAKEFILE) rtw_proj.tmw

$(SHARED_OBJS): $(SHARED_BIN_DIR)/%.obj: $(SHARED_SRC_DIR)/%.c
        $(CC) -c -Fo$@ $(CFLAGS) $<

$(SHARED_LIB): $(SHARED_OBJS)
        @echo ### Creating $@
        @if exist $@ del $@
        $(LIBCMD) /out:$@ $(FMT_SHARED_OBJS)
        @echo ### $@ Created

#--------- Miscellaneous rules to purge, clean and lint (sol2 only) ------------

purge: clean
        @echo ### Deleting the generated source code for $(MODEL)
        @del $(MODEL).c $(MODEL).h $(MODEL)_types.h $(MODEL)_data.c \
          $(MODEL)_private.h $(MODEL).rtw $(MODULES) rtw_proj.tmw $(MAKEFILE)

clean:
        @echo ### Deleting the objects and $(PROGRAM)
        @del $(LINK_OBJS) ..\$(MODEL).exe
|>START_EXPAND_LIBRARIES<|    @del |>EXPAND_LIBRARY_NAME<|.lib
|>END_EXPAND_LIBRARIES<|
|>START_PRECOMP_LIBRARIES<|   @del |>EXPAND_LIBRARY_NAME<|.lib
|>END_PRECOMP_LIBRARIES<|

# EOF: win_dll.tmf
```

ANNEXE 5

FICHIERS DE GÉNÉRATION QNX (TLC)

Fichier qnx_so.tlc permettant d'orchestrer la génération de code et d'appeler le fichier grt_shell_qnx.tlc listé plus bas.

```
%% SYSTLC: QNX Neutrino Real-Time Target (Shared library)\
%% TMF: qnx_so.tmf MAKE: make_rtw EXTMODE: ext_comm
%%
%% $RCSfile: qnx.tlc,v $
%% $Revision: 1.0 $
%% $Date: 2004/04/25 22:03:30 $
%%
%% Copyright 1994-2002 The MathWorks, Inc.
%% Abstract: Generic real-time system target file.
%%
%selectfile NULL_FILE

%assign MatFileLogging = 1

%assign TargetType = "RT"
%assign Language = "C"

%assign GenRTModel = 1

%include "codegenentry.tlc"


%% The contents between 'BEGIN_RTW_OPTIONS' and 'END_RTW_OPTIONS' are strictly
%% written by the standard format. We need to use this structure in RTW
%% options GUI function rtwoptionsdlg.m file.
%%
/%
 BEGIN_RTW_OPTIONS

 % second page of category item
 rtwoptions(1).prompt   = 'QNX code generation options';
 rtwoptions(1).type   = 'Category';
 rtwoptions(1).enable   = 'on';
 rtwoptions(1).default  = 4; % number of items under this category
          % excluding this one.
 rtwoptions(1).popupstrings = '';
 rtwoptions(1).tlcvariable = '';
 rtwoptions(1).tooltip  = '';
 rtwoptions(1).callback  = '';
 rtwoptions(1).opencallback = '';
 rtwoptions(1).closecallback = '';
 rtwoptions(1).makevariable = '';

 rtwoptions(2).prompt   = 'MAT-file variable name modifier';
 rtwoptions(2).type   = 'Popup';
 rtwoptions(2).default  = 'rt_';
 rtwoptions(2).popupstrings = 'rt_|_rt|none';
 rtwoptions(2).tlcvariable = 'LogVarNameModifier';
 rtwoptions(2).tooltip  = ...
 ['prefix rt_ to variable name,', sprintf('\n'), ...
 'append _rt to variable name,', sprintf('\n'), ...
 'or no modification'];

 rtwoptions(3).prompt   = 'External mode';
 rtwoptions(3).type   = 'Checkbox';
 rtwoptions(3).default  = 'off';
 rtwoptions(3).tlcvariable = 'ExtMode';
 rtwoptions(3).makevariable = 'EXT_MODE';
 rtwoptions(3).tooltip  = ...
 ['Adds TCP/IP communication support',sprintf('\n'), ...
 'for use with Simulink external mode'];

 rtwoptions(4).prompt  = 'Ignore custom storage classes';
 rtwoptions(4).type   = 'Checkbox';
 rtwoptions(4).default  = 'on';
 rtwoptions(4).tlcvariable = 'IgnoreCustomStorageClasses';
 rtwoptions(4).tooltip  = ['Treat custom storage classes as ''Auto''.'];
 rtwoptions(4).opencallback = [ ...
```

```
'objTag = ''Ignore custom storage classes_CheckboxTag'';', ...
'obj = findobj(DialogFig,''Tag'',objTag);', ...
'set(obj, ''Enable'', sl(''onoff'',ecoderinstalled));'];


rtwoptions(5).prompt  = 'External mode testing';
rtwoptions(5).type    = 'NonUI';
rtwoptions(5).default  = '0';
rtwoptions(5).tlcvariable = 'ExtModeTesting';
rtwoptions(5).makevariable = 'TMW_EXTMODE_TESTING';
rtwoptions(5).tooltip  = ...
['Internal testing flag for Simulink external mode'];

%----------------------------------------%
% Configure RTW code generation settings %
%----------------------------------------%

rtwgensettings.BuildDirSuffix = '_qnx_so_rtw';

END_RTW_OPTIONS
*/
%include "grt_shell_qnx.tlc"
```

Fichier grt_shell_qnx.tlc pour permettre de contrôler l'exécution de la bibliothèque dynamique d'un programme externe. Ce fichier génère également un programme de test.

```
%%selectfile STDOUT %% Stream de la fen^tre de Matlab
%%Generer le fichier "test_%<CompiledModel.Name>.c"
%openfile foo = "../test_%<CompiledModel.Name>.c"
%selectfile foo
// Test program for the shared library of the model '%<CompiledModel.Name>'
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef __cplusplus
extern "C" {
#endif
%with CompiledModel
char* initiate_%<Name>();
char* compute_%<Name>(int nbrInputArgs, double* inputArgs,int nbrOutputArgs, double*
outputArgs);
char* cleanup_%<Name>();
#ifdef __cplusplus
}
#endif

int main()
{

        %with ExternalInputs
        %assign inputCount=0
        %foreach inp = NumExternalInputs
        %if ExternalInput[inp].Width == 1
        %assign inputCount=inputCount+1
        %else
        %foreach iw = ExternalInput[inp].Width
        %assign inputCount=inputCount+1
        %endforeach
        %endif
        %endforeach
        %endwith
        %with ExternalOutputs
        %assign outputCount=0
        %foreach outp = NumExternalOutputs
                %if ExternalOutput[outp].Width == 1
                        %assign outputCount=outputCount+1
                %else
```

```
                        %foreach ow = ExternalOutput[outp].Width
                        %assign outputCount=outputCount+1
                        %endforeach
                %endif
        %endforeach
        %endwith
        double input[%<inputCount>], output[%<outputCount>];
        int i;
        char * StrRet=(char *) malloc(20000);

        printf("Test program for model '%<Name>'\n");
        printf("This test was automatically generated on %<GeneratedOn>.\n");
        printf("It has %<ExternalOutputs.NumExternalOutputs> external output(s), ");
        printf("%<ExternalInputs.NumExternalInputs> external input(s) and
%<NumContStates> continuous state(s).\n");

        %with ExternalInputs
        %assign inputCount=0
        %foreach inp = NumExternalInputs
        // Input # %<inp> '%<ExternalInput[inp].Identifier>' width:
%<ExternalInput[inp].Width>
        %if ExternalInput[inp].Width == 1
        input[%<inputCount>]=%<inputCount>;
        %assign inputCount=inputCount+1
        %else
        %foreach iw = ExternalInput[inp].Width
        input[%<inputCount>]=%<inputCount>;
        %assign inputCount=inputCount+1
        %endforeach
        %endif
        %endforeach
        %endwith

        initiate_%<Name>();

        for(i=0;i<100000;i++){

        StrRet=compute_%<Name>(%<ExternalInputs.NumExternalInputs>,input,%<ExternalOutput
s.NumExternalOutputs>,output);
                if(StrRet[0]!=0) printf("-> Erreur de calcul: '%s'\r\n",StrRet);
        }

        %with ExternalOutputs
                %assign outputCount=0
                %foreach outp = NumExternalOutputs
        // Output # %<outp>
'%<System[ExternalOutput[outp].Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>
' width: %<ExternalOutput[outp].Width>
                        %%outputArgs[%<outp>] =
%<tOutput>%<IOQualifier>%<System[0].Block[ExternalOutput[outp].Block[1]].Identifier>;
                        %if ExternalOutput[outp].Width == 1
        printf("Output # %<outp>
'%<System[ExternalOutput[outp].Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>
' = %f\n",output[%<outputCount>]);
                                %assign outputCount=outputCount+1
                        %else
                                %foreach ow = ExternalOutput[outp].Width
        printf("Output # %<outp>
'%<System[ExternalOutput[outp].Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>
[%<ow>];' = %f\n",output[%<outputCount>]);
                                        %assign outputCount=outputCount+1
                                %endforeach
                        %endif
                %endforeach
        %endwith

        cleanup_%<Name>();
        printf("Test of %<Name> successful!\n");

        return 0;
}

%closefile foo
%selectfile NULL_FILE
%%selectfile STDOUT %% Stream de la fen^tre de Matlab
```

```
%%Generer le fichier "grt_shell_qnx.h"
%openfile foo = "grt_shell_qnx.h"
%selectfile foo

#ifdef __cplusplus
extern "C" {
#endif
char* initiate_%<Name>();
char* compute_%<Name>(int nbrInputArgs, double* inputArgs,int nbrOutputArgs, double*
outputArgs);
char* cleanup_%<Name>();

#ifdef __cplusplus
}
#endif
%endwith
%closefile foo


%% S,lectionner le stream NULL
%selectfile NULL_FILE

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%selectfile STDOUT %% Stream de la fen^tre de Matlab
%%Generer le fichier "grt_shell_qnx.c"
%openfile foo = "grt_shell_qnx.c"
%selectfile foo
/*
* Modified version of grt_main.c
* Modifications made by Roland Pfeiffer and Vincent Poir,
* File generated from grt_shell_qnx.tlc
*/
/* $Revision: 1.68.4.6 $
* Copyright 1994-2004 The MathWorks, Inc.
*
* File: grt_main.c.tmp
*
* Abstract:
* A Generic "Real-Time (single tasking or pseudo-multitasking,
* statically allocated data)" main that runs under most
* operating systems.

*
* This file may be a useful starting point when targeting a new
* processor or microcontroller.
*
*
* Compiler specified defines:
* RT - Required.
* MODEL=modelname - Required.
* NUMST=# - Required. Number of sample times.
* NCSTATES=# - Required. Number of continuous states.
* TID01EQ=1 or 0 - Optional. Only define to 1 if sample time task
* id's 0 and 1 have equal rates.
* MULTITASKING - Optional. (use MT for a synonym).
* SAVEFILE - Optional (non-quoted) name of .mat file to create.
* Default is <MODEL>.mat
* BORLAND - Required if using Borland C/C++
*/
#include <grt_shell_qnx.h>
#include <float.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tmwtypes.h"
# include "rtmodel.h"
#include "rt_sim.h"
#include "rt_logging.h"
#ifdef UseMMIDataLogging
#include "rt_logging_mmi.h"
#endif
#include "rt_nonfinite.h"
/* Signal Handler header */
#ifdef BORLAND
```

```
#include <signal.h>
#include <float.h>
#endif
#include "ext_work.h"
/*=========*
* Defines *
*=========*/
#ifndef TRUE
#define FALSE (0)
#define TRUE (1)
#endif
#ifndef EXIT_FAILURE
#define EXIT_FAILURE 1
#endif
#ifndef EXIT_SUCCESS
#define EXIT_SUCCESS 0
#endif
#define QUOTE1(name) #name
#define QUOTE(name) QUOTE1(name) /* need to expand name */
#ifndef RT
# error "must define RT"
#endif
#ifndef MODEL
# error "must define MODEL"
#endif
#ifndef NUMST
# error "must define number of sample times, NUMST"
#endif
#ifndef NCSTATES
# error "must define NCSTATES"
#endif
#ifndef SAVEFILE
# define MATFILE2(file) #file ".mat"
# define MATFILE1(file) MATFILE2(file)
# define MATFILE MATFILE1(MODEL)
#else
# define MATFILE QUOTE(SAVEFILE)
#endif
#define RUN_FOREVER -1.0
#define EXPAND_CONCAT(name1,name2) name1 ## name2
#define CONCAT(name1,name2) EXPAND_CONCAT(name1,name2)
#define RT_MODEL CONCAT(MODEL,_rtModel)
/*====================*
* External functions *
*====================*/
extern RT_MODEL *MODEL(void);
extern void MdlInitializeSizes(void);
extern void MdlInitializeSampleTimes(void);
extern void MdlStart(void);
extern void MdlOutputs(int_T tid);
extern void MdlUpdate(int_T tid);
extern void MdlTerminate(void);
#if NCSTATES > 0
extern void rt_ODECreateIntegrationData(RTWSolverInfo *si);
extern void rt_ODEUpdateContinuousStates(RTWSolverInfo *si);
# define rt_CreateIntegrationData(S) \
rt_ODECreateIntegrationData(rtmGetRTWSolverInfo(S));
# define rt_UpdateContinuousStates(S) \
rt_ODEUpdateContinuousStates(rtmGetRTWSolverInfo(S));
# else
# define rt_CreateIntegrationData(S) \
rtsiSetSolverName(rtmGetRTWSolverInfo(S),"FixedStepDiscrete");
# define rt_UpdateContinuousStates(S) \
rtmSetT(S, rtsiGetSolverStopTime(rtmGetRTWSolverInfo(S)));
#endif
/*==================================*
* Global data local to this module *
*==================================*/

static struct {
int_T stopExecutionFlag;
int_T isrOverrun;
int_T overrunFlags[NUMST];
const char_T *errmsg;
} GBLbuf;
```

```
static char errorMsg[200];
#ifdef EXT_MODE
# define rtExtModeSingleTaskUpload(S) \
{ \
int stIdx; \
rtExtModeUploadCheckTrigger(rtmGetNumSampleTimes(S)); \
for (stIdx=0; stIdx<NUMST; stIdx++) { \
if (rtmIsSampleHit(S, stIdx, 0 /*unused*/)) { \
rtExtModeUpload(stIdx,rtmGetTaskTime(S,stIdx)); \
} \
} \
}
#else
# define rtExtModeSingleTaskUpload(S) /* Do nothing */
#endif
/*==================*
* Local functions *
*==================*/
#ifdef BORLAND
/* Implemented for BC++ only*/
typedef void (*fptr)(int, int);
/* Function: divideByZero ==================================================
*
* Abstract: Traps the error Division by zero and prints a warning
* Also catches other FP errors, but does not identify them
* specifically.
*/
void divideByZero(int sigName, int sigType)
{
signal(SIGFPE, (fptr)divideByZero);
if ((sigType == FPE_ZERODIVIDE)||(sigType == FPE_INTDIV0)){
sprintf(errorMsg, "Warning: Division by zero");
return;
}
else{
sprintf(errorMsg, "Warning: Floating Point error");
return;
}
} /* end divideByZero */
#endif /* BORLAND */
/*
* Calculate the controller output. Supply input arguments.
*
* Also supply a pointer to an array capable of containing the output values
*/
static RT_MODEL *S;
%with CompiledModel
char* compute_%<Name>(int nbrInputArgs, double* inputArgs,int nbrOutputArgs, double*
outputArgs) {
real_T tnext;

       /*
       My model is called %<Name>.
       It was generated on %<GeneratedOn>.
       It has %<ExternalOutputs.NumExternalOutputs> external output(s),
%<ExternalInputs.NumExternalInputs> external input(s) and %<NumContStates> continuous
state(s).
       Output:%<tOutput>, %<tOutputType>
       Input:%<tInput>, %<tInputType>
       */

       if (nbrInputArgs != %<ExternalInputs.NumExternalInputs>) {
              sprintf(errorMsg,"nbrInputArgs should be
%<ExternalInputs.NumExternalInputs> but was %d", nbrInputArgs);
              return errorMsg;
       }
       if (nbrOutputArgs != %<ExternalOutputs.NumExternalOutputs>) {
              sprintf(errorMsg, "nbrOutputArgs should be
%<ExternalOutputs.NumExternalOutputs> but was %d", nbrOutputArgs);
              return errorMsg;
       }

       %with ExternalInputs
              %assign inputCount=0
              %foreach inp = NumExternalInputs
```

```
                        // Input # %<inp> '%<ExternalInput[inp].Identifier>' width:
%<ExternalInput[inp].Width>
                        %if ExternalInput[inp].Width == 1
                                %<tInput>%<IOQualifier>%<ExternalInput[inp].Identifier> =
inputArgs[%<inputCount>];
                                %assign inputCount=inputCount+1
                        %else
                        %foreach iw = ExternalInput[inp].Width

        %<tInput>%<IOQualifier>%<ExternalInput[inp].Identifier>[%<iw>] =
inputArgs[%<inputCount>];
                                        %assign inputCount=inputCount+1
                                %endforeach
                        %endif
                %endforeach
        %endwith
        /*********************************************
* Check and see if base step time is too fast *
*********************************************/
if (GBLbuf.isrOverrun++) {
GBLbuf.stopExecutionFlag = 1;
sprintf(errorMsg, "GBLbuf.isrOverrun");
return errorMsg;
}
/*********************************************
* Check and see if error status has been set *
*********************************************/
if (rtmGetErrorStatus(S) != NULL) {
GBLbuf.stopExecutionFlag = 1;
sprintf(errorMsg, "rtmGetErrorStatus(S) != null");
return errorMsg;
}
/* enable interrupts here */
/*
* In a multi-tasking environment, this would be removed from the base rate
* and called as a "background" task.
*/
rtExtModeOneStep(rtmGetRTWExtModeInfo(S),
rtmGetNumSampleTimes(S),
(boolean_T *)&rtmGetStopRequested(S));
tnext = rt_SimGetNextSampleHit();
rtsiSetSolverStopTime(rtmGetRTWSolverInfo(S),tnext);
MdlOutputs(0);
rtExtModeSingleTaskUpload(S);
GBLbuf.errmsg = rt_UpdateTXYLogVars(rtmGetRTWLogInfo(S),
rtmGetTPtr(S));
if (GBLbuf.errmsg != NULL) {
GBLbuf.stopExecutionFlag = 1;
sprintf(errorMsg, "GBLbuf.errmsg != null");
return errorMsg;
}
MdlUpdate(0);
rt_SimUpdateDiscreteTaskSampleHits(rtmGetNumSampleTimes(S),
rtmGetTimingData(S),
rtmGetSampleHitPtr(S),
rtmGetTPtr(S));
if (rtmGetSampleTime(S,0) == CONTINUOUS_SAMPLE_TIME) {
rt_UpdateContinuousStates(S);
}
GBLbuf.isrOverrun--;
rtExtModeCheckEndTrigger();

        %with ExternalOutputs
                %assign outputCount=0
                %foreach outp = NumExternalOutputs
                        // Output # %<outp>
'%<System[ExternalOutput[outp].Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>
' width: %<ExternalOutput[outp].Width>
                        %%outputArgs[%<outp>] =
%<tOutput>%<IOQualifier>%<System[0].Block[ExternalOutput[outp].Block[1]].Identifier>;
                        %if ExternalOutput[outp].Width == 1

        outputArgs[%<outputCount>]=%<tOutput>%<IOQualifier>%<System[ExternalOutput[outp].
Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>;
                                %assign outputCount=outputCount+1
```

```
                        %else
                                %foreach ow = ExternalOutput[outp].Width

        outputArgs[%<outputCount>]=%<tOutput>%<IOQualifier>%<System[ExternalOutput[outp].
Block[0]].Block[ExternalOutput[outp].Block[1]].Identifier>[%<ow>];
                                        %assign outputCount=outputCount+1
                                %endforeach
                        %endif
                %endforeach
        %endwith
%endwith

return errorMsg;
}
/* Function: initiateController
================================================================
*
* Abstract:
* Execute model on a generic target such as a workstation.
*/
char* initiate_%<CompiledModel.Name>() {
const char *status;
/*****************************
* MathError Handling for BC++ *
*****************************/
#ifdef BORLAND
signal(SIGFPE, (fptr)divideByZero);
#endif
{
/***************************
* Initialize global memory *
***************************/
rtExtModeParseArgs(argc, argv, NULL);
(void)memset(&GBLbuf, 0, sizeof(GBLbuf));
/************************
* Initialize the model *
************************/
rt_InitInfAndNaN(sizeof(real_T));
S = MODEL();
if (rtmGetErrorStatus(S) != NULL) {
(void)fprintf(stderr,"Error during model registration: %s\n",
rtmGetErrorStatus(S));
exit(EXIT_FAILURE);
sprintf(errorMsg, "Error during model registration: %s", rtmGetErrorStatus(S));
return(errorMsg);
}
rtmSetTFinal(S,RUN_FOREVER);
MdlInitializeSizes();
MdlInitializeSampleTimes();
status = rt_SimInitTimingEngine(rtmGetNumSampleTimes(S),
rtmGetStepSize(S),
rtmGetSampleTimePtr(S),
rtmGetOffsetTimePtr(S),
rtmGetSampleHitPtr(S),
rtmGetSampleTimeTaskIDPtr(S),
rtmGetTStart(S),
&rtmGetSimTimeStep(S),
&rtmGetTimingData(S));
if (status != NULL) {
(void)fprintf(stderr,
"Failed to initialize sample time engine: %s\n", status);
exit(EXIT_FAILURE);
sprintf(errorMsg, "Failed to initialize sample time engine: %s", status);
return errorMsg;
}
rt_CreateIntegrationData(S);
GBLbuf.errmsg = rt_StartDataLogging(rtmGetRTWLogInfo(S),
rtmGetTFinal(S),
rtmGetStepSize(S),
&rtmGetErrorStatus(S));
rtExtModeCheckInit(rtmGetNumSampleTimes(S));
rtExtModeWaitForStartPkt(rtmGetRTWExtModeInfo(S),
rtmGetNumSampleTimes(S),
(boolean_T *)&rtmGetStopRequested(S));
MdlStart();
```

```
if (rtmGetErrorStatus(S) != NULL) {
GBLbuf.stopExecutionFlag = 1;
}
errorMsg[0] = 0;
return errorMsg;
/*
Initialization should be finished at this point
*/
}
}
char* cleanup_%<CompiledModel.Name>() {
#ifdef UseMMIDataLogging
rt_CleanUpForStateLogWithMMI(rtmGetRTWLogInfo(S));
#endif
rt_StopDataLogging(MATFILE,rtmGetRTWLogInfo(S));
rtExtModeShutdown(rtmGetNumSampleTimes(S));
if (GBLbuf.errmsg) {
(void)fprintf(stderr,"%s\n",GBLbuf.errmsg);
exit(EXIT_FAILURE);
sprintf(errorMsg, "%s", GBLbuf.errmsg);
return errorMsg;
}
if (GBLbuf.isrOverrun) {
(void)fprintf(stderr,
"%s: ISR overrun - base sampling rate is too fast\n",
QUOTE(MODEL));
exit(EXIT_FAILURE);
sprintf(errorMsg, "%s: ISR overrun - base sampling rate is too fast",QUOTE(MODEL));
return errorMsg;
}
if (rtmGetErrorStatus(S) != NULL) {
(void)fprintf(stderr,"%s\n", rtmGetErrorStatus(S));
exit(EXIT_FAILURE);
sprintf(errorMsg,"%s", rtmGetErrorStatus(S));
return errorMsg;
}
return errorMsg;
}

%closefile foo
%% S,lectionner le stream NULL
%selectfile NULL_FILE
```

ANNEXE 6

FICHIERS DE COMPILATION QNX (TMF)

Le fichier qnx_so.tmf permet de générer un fichier de compilation utilisé par la commande make de QNX. Il a été adapté de [66].

```
# Copyright 1994-2002 The MathWorks, Inc.
#
# File: qnx_unix.tmf $Revision: 1.81 $
#
# Abstract:
#       Real-Time Workshop template makefile for building a UNIX-based
#       stand-alone generic real-time version of Simulink model using
#       generated C code.
#
#       This makefile attempts to conform to the guidelines specified in the
#       IEEE Std 1003.2-1992 (POSIX) standard. It is designed to be used
#  with GNU Make which is located in matlabroot/rtw/bin.
#
#       Note that this template is automatically customized by the Real-Time
#       Workshop build procedure to create "<model>.mk"
#
#  The following defines can be used to modify the behavior of the
#       build:
#        OPT_OPTS  - Optimization options. Default is -O. To enable
#        debugging specify as OPT_OPTS=-g.
#        Because of optimization problems in IBM_RS,
#        default is no-optimization.
#        OPTS    - User specific compile options.
#        USER_SRCS  - Additional user sources, such as files needed by
#                     S-functions.
#        USER_INCLUDES - Additional include paths
#                      (i.e. USER_INCLUDES="-Iwhere-ever -Iwhere-ever2")
#
#  This template makefile is designed to be used with a system target
#  file that contains 'rtwgensettings.BuildDirSuffix' see qnx.tlc

#------------------------ Macros read by make_rtw ------------------------------
#
# The following macros are read by the Real-Time Workshop build procedure:
#
# MAKECMD   - This is the command used to invoke the make utility
# HOST    - What platform this template makefile is targeted for
#      (i.e. PC or UNIX)
# BUILD    - Invoke make from the Real-Time Workshop build procedure
#      (yes/no)?
# SYS_TARGET_FILE - Name of system target file.

MAKECMD   = make
HOST    = UNIX
BUILD    = yes
SYS_TARGET_FILE = qnx_so.tlc

#---------------------- Tokens expanded by make_rtw ----------------------------
#
# The following tokens, when wrapped with "|>" and "|<" are expanded by the
# Real-Time Workshop build procedure.
#
# MODEL_NAME    - Name of the Simulink block diagram
# MODEL_MODULES  - Any additional generated source modules
# MAKEFILE_NAME  - Name of makefile created from template makefile <model>.mk
# MATLAB_ROOT    - Path to were MATLAB is installed.
# S_FUNCTIONS    - List of S-functions.
# S_FUNCTIONS_LIB  - List of S-functions libraries to link.
# SOLVER     - Solver source file name
# NUMST     - Number of sample times
# TID01EQ     - yes (1) or no (0): Are sampling rates of continuous task
#       (tid=0) and 1st discrete task equal.
# NCSTATES    - Number of continuous states
# COMPUTER    - Computer type. See the MATLAB computer command.
# BUILDARGS    - Options passed in at the command line.
# MULTITASKING  - yes (1) or no (0): Is solver mode multitasking
# EXT_MODE    - yes (1) or no (0): Build for external mode
# TMW_EXTMODE_TESTING - yes (1) or no (0): Build ext_test.c for external mode
```

```
#      testing.

MODEL    = |>MODEL_NAME<|
MODULES    = |>MODEL_MODULES<|
MAKEFILE    = |>MAKEFILE_NAME<|
MATLAB_ROOT    = /matlab
S_FUNCTIONS    = |>S_FUNCTIONS<|
S_FUNCTIONS_LIB  = |>S_FUNCTIONS_LIB<|
SOLVER    = |>SOLVER<|
NUMST    = |>NUMST<|
TID01EQ    = |>TID01EQ<|
NCSTATES    = |>NCSTATES<|
COMPUTER    = GLNX86
BUILDARGS    = |>BUILDARGS<|
MULTITASKING    = |>MULTITASKING<|
EXT_MODE    = |>EXT_MODE<|
TMW_EXTMODE_TESTING = |>TMW_EXTMODE_TESTING<|

#---------------------------- Tool Specifications -------------------------------

include $(MATLAB_ROOT)/rtw/c/tools/unixtools.mk

#------------------------------ Include Path -----------------------------------

MATLAB_INCLUDES = \
        -I$(MATLAB_ROOT)/Simulink/include \
        -I$(MATLAB_ROOT)/extern/include \
        -I$(MATLAB_ROOT)/rtw/c/src \
        -I$(MATLAB_ROOT)/rtw/c/libsrc \
        -I$(MATLAB_ROOT)/rtw/c/src/ext_mode/common \
        -I$(MATLAB_ROOT)/toolbox/pumatrc

# Additional file include paths
ADD_INCLUDES = \
|>START_EXPAND_INCLUDES<|     -I/matlab/rtw/c/libsrc \
|>END_EXPAND_INCLUDES<|

INCLUDES = -I. -I.. $(MATLAB_INCLUDES) $(ADD_INCLUDES) $(USER_INCLUDES) \
        $(INSTRUMENT_INCLUDES)

#------------------------------ External mode ----------------------------------
# Uncomment -DVERBOSE to have information printed to stdout (or specify
# OPTS=-DVERBOSE).
ifeq ($(EXT_MODE),1)
 EXT_SRC  = ext_svr.c updown.c ext_svr_transport.c ext_work.c
 EXT_CC_OPTS = -DEXT_MODE -D$(COMPUTER) #-DVERBOSE
 EXT_LIB  =
 ifeq ($(TMW_EXTMODE_TESTING),1)
 EXT_SRC  += ext_test.c
 EXT_CC_OPTS += -DTMW_EXTMODE_TESTING
 endif
 ifeq ($(COMPUTER),SOL2)
 EXT_LIB = -lsocket -lnsl
 endif
endif

#---------------------------- Real-Time Model ----------------------------------
RTM_CC_OPTS = -DUSE_RTMODEL

#-------------------------------- C Flags --------------------------------------

# Optimization Options
ifndef OPT_OPTS
OPT_OPTS = $(DEFAULT_OPT_OPTS)
endif

# General User Options
OPTS = -g -V 2.95.3qnx-nto

# Compiler options, etc:
CC_OPTS = $(OPT_OPTS) $(OPTS) $(ANSI_OPTS) $(EXT_CC_OPTS) $(RTM_CC_OPTS)

CPP_REQ_DEFINES = -DMODEL=$(MODEL) -DRT -DNUMST=$(NUMST) \
     -DTID01EQ=$(TID01EQ) -DNCSTATES=$(NCSTATES) -DUNIX \
     -DMT=$(MULTITASKING) -DHAVESTDIO
```

```
CFLAGS = $(CC_OPTS) $(CPP_REQ_DEFINES) $(INCLUDES)

LDFLAGS =

#------------------------- Additional Libraries -------------------------------

SYSLIBS = $(EXT_LIB) -lm -lsocket

LIBS =
|>START_PRECOMP_LIBRARIES<|
ifeq ($(OPT_OPTS),$(DEFAULT_OPT_OPTS))
LIBS += |>EXPAND_LIBRARY_NAME<|.a
else
LIBS += |>EXPAND_LIBRARY_NAME<|.a
endif
|>END_PRECOMP_LIBRARIES<| |>START_EXPAND_LIBRARIES<|
LIBS += |>EXPAND_LIBRARY_NAME<|.a |>END_EXPAND_LIBRARIES<|
LIBS += $(S_FUNCTIONS_LIB) $(INSTRUMENT_LIBS)

#---------------------------- Source Files ------------------------------------

REQ_SRCS = $(MODEL).c $(MODULES) rt_sim.c rt_nonfinite.c \
   $(EXT_SRC)

USER_OBJS  = $(USER_SRCS:.c=.o)
LOCAL_USER_OBJS = $(notdir $(USER_OBJS))

SRCS  = $(REQ_SRCS) $(S_FUNCTIONS) $(SOLVER)
OBJS  = $(SRCS:.c=.o) $(USER_OBJS)
LINK_OBJS = $(SRCS:.c=.o) $(LOCAL_USER_OBJS)

PROGRAM = lib$(MODEL).so

#------------- Test Compile using gcc -Wall to look for warnings ---------------
#
# DO_GCC_TEST=1 runs gcc with compiler warning flags on all the source files
# used in this build. This includes the generated code, and any user source
# files needed for the build and placed in this directory.
#
# WARN_ON_GLNX=1 runs the linux compiler with warnings flags. On hand-written
# code we use the max amount of flags availabe while on the generated code, a
# few less.
#
# See rtw/c/tools/unixtools.mk for the definition of GCC_WARN_OPTS
GCC_TEST_CMD:= echo
GCC_TEST_OUT:= > /dev/null
ifeq ($(DO_GCC_TEST), 1)
 GCC_TEST:= gcc -c -o /dev/null $(GCC_WARN_OPTS_MAX) $(CPP_REQ_DEFINES) \
        $(INCLUDES)
 GCC_TEST_CMD:= echo; echo "\#\#\# GCC_TEST $(GCC_TEST) $<"; $(GCC_TEST)
 GCC_TEST_OUT:=; echo
endif

GCC_WALL_FLAG :=
GCC_WALL_FLAG_MAX:=
ifeq ($(COMPUTER), GLNX86)
ifeq ($(WARN_ON_GLNX), 1)
 GCC_WALL_FLAG := $(GCC_WARN_OPTS)
 GCC_WALL_FLAG_MAX:= $(GCC_WARN_OPTS_MAX)
endif
endif


#----------------------------- Lint (sol2 only) -------------------------------

LINT_SRCS = $(MATLAB_ROOT)/rtw/c/src/rt_sim.c
ifneq ($(SOLVER), )
 LINT_SRCS += $(MATLAB_ROOT)/rtw/c/src/$(SOLVER)
endif
ifneq ($(EXT_SRC), )
 LINT_SRCS += $(MATLAB_ROOT)/rtw/c/src/$(EXT_SRC)
endif
LINT_SRCS += $(MODEL).c $(MODULES) $(USER_SRCS) $(S_FUNCTIONS)
LINTOPTSFILE = $(MODEL).lintopts
```

```
LINT_ERROFF1 = E_NAME_DEF_NOT_USED2,E_NAME_DECL_NOT_USED_DEF2
LINT_ERROFF2 = $(LINT_ERROFF1),E_FUNC_ARG_UNUSED
LINT_ERROFF = $(LINT_ERROFF2),E_INDISTING_FROM_TRUNC2,E_NAME_USED_NOT_DEF2

#---------------------------------- Rules ---------------------------------------

$(PROGRAM): $(OBJS) $(LIBS)
        $(LD) $(LDFLAGS) -shared -Wl,-soname,$@ -o $@ $(LINK_OBJS) $(LIBS) $(SYSLIBS)
        #$(LD) $(LDFLAGS) -fpic -c $@ $(LINK_OBJS) $(LIBS) $(SYSLIBS)
        @echo "### Created executable: $(MODEL)"
#ld libCTorque6.so CTorque6.o CTorque6_data.o grt_shell_qnx.o rt_sim.o rt_nonfinite.o
ode4.o rtwlib.a -lm -lc
#gcc -Wall hello.c -L. -lCTorque6 -o hello

%.o: %.c
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CC) -c -o $(@F) $(CFLAGS) $(GCC_WALL_FLAG) $<

%.o: $(MATLAB_ROOT)/rtw/c/qnx_so/%.c
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CC) -c $(CFLAGS) $(GCC_WALL_FLAG_MAX) $<

%.o: $(MATLAB_ROOT)/rtw/c/src/%.c
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CC) -c $(CFLAGS) $(GCC_WALL_FLAG_MAX) $<

%.o: $(MATLAB_ROOT)/rtw/c/libsrc/%.c
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CC) -c $(CFLAGS) $(GCC_WALL_FLAG_MAX) $<

%.o: $(MATLAB_ROOT)/rtw/c/%.cpp
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CPP) -c $(CFLAGS) $<

%.o: $(MATLAB_ROOT)/Simulink/src/%.c
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CC) -c $(CFLAGS) $(GCC_WALL_FLAG_MAX) $<

%.o: $(MATLAB_ROOT)/Simulink/src/%.cpp
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CPP) -c $(CFLAGS) $<

%.o: $(MATLAB_ROOT)/toolbox/pumatrc/%.c
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CC) -c $(CFLAGS) $(GCC_WALL_FLAG_MAX) $<

%.o: ../%.c
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CC) -c $(CFLAGS) $(GCC_WALL_FLAG) $<

%.o: ../%.cpp
        @$(GCC_TEST_CMD) $< $(GCC_TEST_OUT)
        $(CPP) -c $(CFLAGS) $<

#---------------------------------- Libraries ---------------------------------------

|>START_EXPAND_LIBRARIES<|MODULES_|>EXPAND_LIBRARY_NAME<| = \
|>START_EXPAND_MODULES<|      |>EXPAND_MODULE_NAME<|.o \
|>END_EXPAND_MODULES<|

|>EXPAND_LIBRARY_NAME<|.a: $(MAKEFILE) rtw_proj.tmw $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo "### Creating $@ "
        ar r $@ $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo "### Created $@"

|>END_EXPAND_LIBRARIES<|

|>START_PRECOMP_LIBRARIES<|MODULES_|>EXPAND_LIBRARY_NAME<| = \
|>START_EXPAND_MODULES<|      |>EXPAND_MODULE_NAME<|.o \
|>END_EXPAND_MODULES<|

|>EXPAND_LIBRARY_NAME<|.a: $(MAKEFILE) rtw_proj.tmw $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo "### Creating $@ "
        ar r $@ $(MODULES_|>EXPAND_LIBRARY_NAME<|)
        @echo "### Created $@"
```

```
|>END_PRECOMP_LIBRARIES<|

#----------------------------- Dependencies -------------------------------------

$(OBJS): $(MAKEFILE) rtw_proj.tmw

#--------- Miscellaneous rules to purge, clean and lint (sol2 only) ------------

purge: clean
        @echo "### Deleting the generated source code for $(MODEL)"
        @\rm -f $(MODEL).c $(MODEL).h $(MODEL)_types.h $(MODEL)_data.c \
          $(MODEL)_private.h $(MODEL).rtw $(MODULES) rtw_proj.tmw $(MAKEFILE)

clean:
        @echo "### Deleting the objects and $(PROGRAM)"
        @\rm -f $(LINK_OBJS) $(PROGRAM)

lint: rtwlib.ln
        @lint -errchk -errhdr=%user -errtags=yes -F -L. -lrtwlib -x -Xc \
          -erroff=$(LINT_ERROFF) \
          -D_POSIX_C_SOURCE $(CFLAGS) $(LINT_SRCS)
        @\rm -f $(LINTOPTSFILE)
        @echo
        @echo "### Created lint output only, no executable"
        @echo

rtwlib.ln: $(MAKEFILE) rtw_proj.tmw
        @echo
        @echo "### Linting ..."
        @echo
        @\rm -f llib-lrtwlib.ln $(LINTOPTSFILE)
        @echo "-dirout=. -errchk -errhdr=%user " >> $(LINTOPTSFILE)
        @echo "-errtags -F -ortwlib -x -Xc " >> $(LINTOPTSFILE)
        @echo "-erroff=$(LINT_ERROFF) " >> $(LINTOPTSFILE)
        @echo "-D_POSIX_C_SOURCE $(CFLAGS) " >> $(LINTOPTSFILE)
        @for file in $(MATLAB_ROOT)/rtw/c/libsrc/*.c; do \
         echo "$$file " >> $(LINTOPTSFILE); \
        done
        lint -flagsrc=$(LINTOPTSFILE)

# EOF: qnx_unix.tmf
```

ANNEXE 7

CODE D'APPEL POUR BIBLIOTHÈQUE (WINDOWS)

```
//redefinitions de type pour les pointeurs de fonctions
	typedef char* (CALLBACK* LPFNINIT) ();
	typedef char* (CALLBACK* LPFNOUT)(int,double*,int,double*);
	typedef char* (CALLBACK* LPFNTERMINATE) ();

// Tableau de caractères pour recevoir les codes d'erreur de la dll
	char * StrRet;


// fonction qui vérifie si la fonction existe (tiré de l'aide en ligne MSDN)
bool FunctionExist(FARPROC fp)
{

	if(fp == NULL){
		LPVOID lpMsgBuf;
		FormatMessage(
			FORMAT_MESSAGE_ALLOCATE_BUFFER |
			FORMAT_MESSAGE_FROM_SYSTEM |
			FORMAT_MESSAGE_IGNORE_INSERTS,
			NULL,
			GetLastError(),
			MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
			(LPTSTR) &lpMsgBuf,
			0,
			NULL
			);
		printf("la fct n'existe pas...\r\n");
		printf( (LPCTSTR)lpMsgBuf );
		LocalFree( lpMsgBuf );
		return false;
	}
	return true;
}

// initialisation des pointeurs de fonction et de la dll
bool Initialisation()
{
	StrRet= new char[20000];

// chargement de la dll
	hClcltr=LoadLibrary(<nom du dll>);
	// Si le pointeur est nul, la dll est introuvable
	if(hClcltr == NULL){
		return false;
	}

// Initialisation du pointeur de la fonction _initiateController()
	lpfnInit = (LPFNINIT)GetProcAddress(hClcltr,"_initiateController");
	if(!FunctionExist((FARPROC) lpfnInit)){
		FreeLibrary( hClcltr );
		return false;
	}

// Initialisation du pointeur de la fonction _getControllerOutput ()
	lpfnOut = (LPFNOUT)GetProcAddress(hClcltr,"_getControllerOutput");
	if(!FunctionExist((FARPROC) lpfnOut)){
		FreeLibrary( hClcltr );
		return false;
	}

// Initialisation du pointeur de la fonction _performCleanup ()
	lpfnTerm = (LPFNTERMINATE)GetProcAddress(hClcltr,"_performCleanup");
	if(!FunctionExist((FARPROC) lpfnTerm)){
		FreeLibrary( hClcltr );
		return false;
	}

// Initilisation du dll
	StrRet=lpfnInit();
	if(StrRet[0]!=0) {
		printf(StrRet);
		return false;
	}

	return true;
```

```
}

// Calculs de la sortie du dll
void Ctrl::Calcul()
{
        StrRet=lpfnOut(<nb d'entrée>,
                       <tableau des entrées(double)>,
                       <nb de sorties>,
                       <tableau des sorties(double)>);

        if(StrRet[0]!=0) printf(StrRet);

        return;
}

// Remise à zéro de la bibliothèque dynamique
bool Fermeture()
{
        bool ret=true;
        StrRet=lpfnTerm();
        if(StrRet[0]!=0) {
                printf(StrRet);
                ret= false;
        }

        return ret;
}
```

ANNEXE 8

CODE D'APPEL POUR BIBLIOTHÈQUE (QNX)

```
// Tableau de caractères pour recevoir les codes d'erreur de la bibliothèque partagée
        char * StrRet;
        StrRet= new char[20000];

// Initialisation de la bibliothèque partagée
        StrRet=initiate_<nom du modèle Simulink>();
        if(StrRet[0]!=0) printf("-> Erreur d'initialisation: '%s'\r\n",StrRet);

// Calculs de la sortie de la bibliothèque partagée
        StrRet= compute_<nom du modèle Simulink>(   <nb d'entrée>,
                                                    <tableau des entrées(double)>,
                                                    <nb de sorties>,
                                                    <tableau des sorties(double)>);

        if(StrRet[0]!=0) printf("-> Erreur de calcul: '%s'\r\n",StrRet);

// Remise à zéro de la bibliothèque partagée
        StrRet=cleanup_<nom du modèle Simulink> ();
        if(StrRet[0]!=0) printf("-> Erreur de remise à zéro: '%s'\r\n",StrRet);
```

ANNEXE 9

ROBOT BART

Modèle dynamique d'un robot planaire à deux degrés de liberté

Hypothèses (p.177):

- Des masses ponctuelles (*m1, m2*) sont situées à l'extrémité des membrures;
- Il n'y a pas de forces appliquées à l'extrémité de l'effecteur;
- Les masses (*m1, m2*) sont de 2 Kg et les membrures (*l1, l2*) mesurent 1 mètre de longeur.

Le modèle dynamque a la forme suivante:

$$\boldsymbol{\tau} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) + \mathbf{F}(\mathbf{q},\dot{\mathbf{q}})$$

Où:

$\boldsymbol{\tau}$ est le vecteur des couples resultants appliqués aux joints

$\mathbf{q}$ est le vecteur de position des joints

$\dot{\mathbf{q}}$ est le vecteur de vitesse des joints

$\ddot{\mathbf{q}}$ est le vecteur d'accélération des joints

M est la matrice de masse

V est la matrice de vitesse, qui contient les termes de Coriolis et de force centrifuge

G est la matrice de gravité

F est la matrice de frottement

Pour le bras présenté (le frottement est négligé):

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} l_2^2 m_2 + 2 l_1 l_2 m_2 c_2 + l_1^2 (m_1 + m_2) & l_2^2 m_2 + l_1 l_2 m_2 c_2 \\ l_2^2 m_2 + l_1 l_2 m_2 c_2 & l_2^2 m_2 \end{bmatrix}$$

$$\mathbf{V}(\mathbf{q},\dot{\mathbf{q}}) = \begin{bmatrix} -m_2 l_1 l_2 s_2 \dot{q}_2^2 - 2 m_2 l_1 l_2 s_2 \dot{q}_1 \dot{q}_2 \\ m_2 l_1 l_2 s_2 \dot{q}_1^2 \end{bmatrix}$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} m_2 l_2 g c_{12} + (m_1 + m_2) l_1 g c_1 \\ m_2 l_2 g c_{12} \end{bmatrix}$$

$$\mathbf{F}(\mathbf{q},\dot{\mathbf{q}}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

ANNEXE 10

EXEMPLE DE MODIFICATION DU FICHIER DE COMPILATION

Dans le fichier de compilation du contrôleur global, ici /Bart/src/Makefile, deux lignes doivent être modifiées pour permettre de lier la bibliothèque partagée migrée avec l'exécutable. Le paramètre à ajouter est « –l » suivi du nom du modèle. Dans ce cas, les paramètres ajoutés sont « –lCTorque4 » pour une bibliothèque partagée qui se nomme « libCTorque4.so ». Les modifications sont illustrées en gras dans les lignes qui suivent.

```
[…]
##############################################################################
#
# Regle pour creer un fichier executable
#
##############################################################################
OBJ_PATH = $(ROBOTS)/obj/$(MICPATH)

$(ROBOTS)/bin/$(MICPATH)/server_bart: $(OBJSRC)
echo ---- Creating $@;
#echo $(CC) -o $@ $(OBJSRC) $(CFLAGS) $(CLFLAGS) -L/Bart/lib/QNX6/Pentium -lCTorque6;
$(CC) -o $@ $(OBJSRC) $(CFLAGS) $(CLFLAGS) -L/Bart/lib/QNX6/Pentium -lCTorque6;

[…]
```

ANNEXE 11

EXEMPLE D'APPEL DU CONTRÔLEUR (WINDOWS)

Les fonctions « FunctionExist », « Initialisation », « Calcul » et « Fermeture » doivent être ajoutée au code du contrôleur global. La fonction « Initialisation » doit être appelée lors du démarrage du contrôleur et « Fermeture » lors de son arrêt. La fonction « calcul » doit être appelée lors de l'exécution de la boucle de contrôle.

```
// vérifie si la fonction existe
bool Ctrl::FunctionExist(FARPROC fp)
{

	if(fp == NULL){
		LPVOID lpMsgBuf;
		FormatMessage(
			FORMAT_MESSAGE_ALLOCATE_BUFFER |
			FORMAT_MESSAGE_FROM_SYSTEM |
			FORMAT_MESSAGE_IGNORE_INSERTS,
			NULL,
			GetLastError(),
			MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
			(LPTSTR) &lpMsgBuf,
			0,
			NULL
			);
		// Process any inserts in lpMsgBuf.
		// ...
		// Display the String.
		printf("la fct n'existe pas...\r\n");
		printf( (LPCTSTR)lpMsgBuf );
		// Free the buffer.
		LocalFree( lpMsgBuf );

		return false;
	}
	return true;
}

bool Ctrl::Initialisation()
{
	StrRet= new char[20000];
	printf("Chargement du dll...\r\n");
	hClcltr=LoadLibrary("CTorque4.dll");

	if(hClcltr == NULL){
		LPVOID lpMsgBuf;
		FormatMessage(
			FORMAT_MESSAGE_ALLOCATE_BUFFER |
			FORMAT_MESSAGE_FROM_SYSTEM |
			FORMAT_MESSAGE_IGNORE_INSERTS,
			NULL,
			GetLastError(),
			MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
	(LPTSTR) &lpMsgBuf,
			0,
			NULL
			);

		// Display the String.
		printf("la dll n'existe pas...\r\nErreur systeme: ");
		printf( (LPCTSTR)lpMsgBuf );
		// Free the buffer.
		LocalFree( lpMsgBuf );
		return false;
	}

	printf("dll charge\r\ndebut d'appel des fonctions\r\n");

	lpfnInit = (LPFNINIT)GetProcAddress(hClcltr,"_initiateController");
	if(!FunctionExist((FARPROC) lpfnInit)){
		FreeLibrary( hClcltr );
		return false;
```

```
        }

        printf("fonction 1 initialisee\r\n");

        lpfnOut = (LPFNOUT)GetProcAddress(hClcltr,"_getControllerOutput");
        if(!FunctionExist((FARPROC) lpfnOut)){
                FreeLibrary( hClcltr );
                return false;
        }
        printf("fonction 2 initialisee\r\n");

        lpfnTerm = (LPFNTERMINATE)GetProcAddress(hClcltr,"_performCleanup");
        if(!FunctionExist((FARPROC) lpfnTerm)){
                FreeLibrary( hClcltr );
                return false;
        }

        printf("fonction 3 initialisee\r\nInitilisation du dll\r\n");
        StrRet=lpfnInit();
        if(StrRet[0]!=0) {
                printf(StrRet);
                return false;
        }

        return true;

}

// remise à zéro de la bibliothèque dynamique
bool Ctrl::Fermeture()
{
        bool ret=true;
        StrRet=lpfnTerm();
        if(StrRet[0]!=0) {
                printf("-> Erreur terminate: '");
                printf(StrRet);
                printf("'\r\n");
                ret= false;
        }

        printf("dll libre,\r\n");

        return ret;
}

// Calculs de compensation à couple précalculé du système
void Ctrl::Calcul()
{

        // Calculs


        input[0]=q1;            //q1 actual_gen_coord[0]
        input[1]=q2;            //q2 actual_gen_coord[1]
        input[2]=qp1;           //qp1 actual_dgen_coord[0]
        input[3]=qp2;           //qp2 actual_dgen_coord[1]
        input[4]=qd1;           //qd1 desired_gen_coord[0]
        input[5]=qd2;           //qd2 desired_gen_coord[1]
        input[6]=qpd1;          //qpd1 desired_dgen_coord[0]
        input[7]=qpd2;          //qpd2 desired_dgen_coord[1]
        input[10]=kp;           //kp
        input[11]=kv;           //kv

        Temps[1]=mc_get_accurate_time();
        StrRet=lpfnOut(7,input,1,output);
        Temps[1]=mc_get_accurate_time()-Temps[1];

        if(StrRet[0]!=0) printf("-> Erreur de calcul: '%s'\r\n",StrRet);

        Tau_dll[0] = output[0];
        Tau_dll[1] = output[1];

        return;
}
```

ANNEXE 12

EXEMPLE D'APPEL DU CONTRÔLEUR (QNX)

Code à insérer dans le constructeur du contrôleur:

```
// initialiser le contrôleur à couple précalculé
StrRet= new char[20000];
StrRet=initiate_CTorque6();
if(StrRet[0]!=0) printf("-> Erreur d'initialisation: '%s'\r\n",StrRet);
```

Code à insérer dans la fonction de calcul du contrôleur:

```
// Code pour effectuer les calculs

input[0]=q1;  //q1 actual_gen_coord[0]
input[1]=q2;  //q2 actual_gen_coord[1]
input[2]=qp1;  //qp1 actual_dgen_coord[0]
input[3]=qp2;  //qp2 actual_dgen_coord[1]
input[4]=qd1;  //qd1 desired_gen_coord[0]
input[5]=qd2;  //qd2 desired_gen_coord[1]
input[6]=qpd1;  //qpd1 desired_dgen_coord[0]
input[7]=qpd2;  //qpd2 desired_dgen_coord[1]
input[8]=qppd1;  //qppd1 acceleration_joint[0]?
input[9]=qppd2; //qppd2 acceleration_joint[1]?
input[10]=kp; //kp
input[11]=kv; //kv
input[12]=ki; //ki

Temps[1]=mc_get_accurate_time();
StrRet=compute_CTorque6(8,input,1,output);
Temps[1]=mc_get_accurate_time()-Temps[1];

//printf("-> output .so       '%f,     %f'\r\n",output[0],output[1]);

if(StrRet[0]!=0) printf("-> Erreur de calcul: '%s'\r\n",StrRet);

Tau_dll[0] = output[0];
Tau_dll[1] = output[1];
```

Code à insérer dans l'arrêt du contrôleur:

```
// remise à zéro de la bibliothèque dynamique
StrRet=cleanup_CTorque6();
if(StrRet[0]!=0) printf("-> Erreur de cleanup: '%s'\r\n",StrRet);
```

ANNEXE 13

PROGRAMME DE TEST POUR QNX (CTORQUE4)

Fichier test_CTorque4.c généré automatiquement par la cible *qnx_so* de Real-time Workshop.

```
// Test program for the shared library of the model 'CTorque4'
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef __cplusplus
extern "C" {
#endif
char* initiate_CTorque4();
char* compute_CTorque4(int nbrInputArgs, double* inputArgs,int nbrOutputArgs, double*
outputArgs);
char* cleanup_CTorque4();
#ifdef __cplusplus
}
#endif

int main()
{

        double input[12], output[2];
        int i;
        char * StrRet=(char *) malloc(20000);

        printf("Test program for model 'CTorque4'\n");
        printf("This test was automatically generated on Mon Aug 21 23:50:09 2006.\n");
        printf("It has 1 external output(s), ");
        printf("7 external input(s) and 0 continuous state(s).\n");

        // Input # 0 'X' width: 2
        input[0]=0;
        input[1]=1;
        // Input # 1 'Xp' width: 2
        input[2]=2;
        input[3]=3;
        // Input # 2 'Xd' width: 2
        input[4]=4;
        input[5]=5;
        // Input # 3 'Xpd' width: 2
        input[6]=6;
        input[7]=7;
        // Input # 4 'Xppd' width: 2
        input[8]=8;
        input[9]=9;
        // Input # 5 'kp' width: 1
        input[10]=10;
        // Input # 6 'kv' width: 1
        input[11]=11;

        initiate_CTorque4();

        for(i=0;i<100000;i++){
                StrRet=compute_CTorque4(7,input,1,output);
                if(StrRet[0]!=0) printf("-> Erreur de calcul: '%s'\r\n",StrRet);
        }

        // Output # 0 'Tau' width: 2
        printf("Output # 0 'Tau[0];' = %f\n",output[0]);
        printf("Output # 0 'Tau[1];' = %f\n",output[1]);

        cleanup_CTorque4();
        printf("Test of CTorque4 successful!\n");

        return 0;
```

BIBLIOGRAPHIE

[1] Vian, B. (105 E.P.). Mémoire concernant le calcul numérique de Dieu par des méthodes simples et fausses. Cymbalum Pataphysicvm (cv), Paris.

[2] Mathworks. The MathWorks - MATLAB and Simulink for Technical Computing, [En ligne]. http://www.mathworks.com (Consulté le 5 juillet 2006).

[3] IREQ. MICROB, [En ligne]. http://robotique.ireq.ca (Consulté le 5 juillet 2006).

[4] Lemieux, S., Beaudry, J., Blain, M. (2006). Force Control Test Bench for Underwater Vehicle-Manipulator System Applications. IEEE - Industrial Electronics Conference (IECON) 2006, Paris.

[5] Poiré, V., Lemieux, S., Bigras, P., Blain, M. (2006). Migration Procedure of Control Laws from a Graphical Simulation Environment to a Robotic Framework. IASTED conference on Control and Application 2006, Montréal.

[6] Fiévet, C. (2002). Les robots (1re ed.). Paris: Presses universitaires de France.

[7] UNECE. Index orders placed for industrial robots: world total and by regions, [En ligne]. http://www.unece.org/stats/robotics/ (Consulté le 5 juillet 2006).

[8] Colestock, H. (2005). Industrial robotics: selection, design, and maintenance. New York: McGraw-Hill.

[9] IFR. Robot applications, [En ligne]. http://www.ifr.org/pictureGallery/robAppl.htm (Consulté le 5 juillet 2006).

[10] Associated Press. Europe Has the Hot Hand, [En ligne]. http://www.wired.com/news/medtech/0,1286,69749,00.html (Consulté le 25 juillet 2006).

[11] Dutkiewicz, P., Kozlowski, K. R., Wroblewski, W. S. (1993). Robot programming system for research purposes. Proceedings of COMPEURO '93, 24-27 May 1993, Paris-Evry, France.

[12] Nakamura, A., Ohyama, Y., Ito, K., Saito, K. (1986). Controller for industrial robots. IEEE International Conference on Robotics and Automation.

[13] Loffler, M. S., Dawson, D. M., Zergeroglu, E., Costescu, N. P. (2001). Object-oriented techniques in robot manipulator control software development. 2001 American Control Conference, Jun 25-27 2001, Arlington, VA.

[14] Kapoor, C. T., D. (1998). A reusable operational software architecture for advanced robotics. 12th CSIM-IFToMM Symp. Theory and Practice of Robots and Manips, Paris, France.

[15] MacDonald, G. B. B. (2003). A Survey of Robot Programming Systems. Proceedings of the Australasian Conference on Robotics and Automation, CSIRO, Brisbane, Australia.

[16] Haskell Homepage. Yampa: Functional Reactive Programming with Arrows, [En ligne]. http://www.haskell.org (Consulté le 7 août 2006).

[17] Haskell Homepage. Haskell - A Purely Functional Language, [En ligne]. http://www.haskell.org (Consulté le 7 août 2006).

[18] Dixon, W. E., Moses, D., Walker, I. D., Dawson, D. M. (2001). A Simulink-based robotic toolkit for simulation and control of the PUMA 560 robot manipulator. Proceedings of RSJ/IEEE International Conference on Intelligent Robots and Systems, 29 Oct.-3 Nov. 2001, Maui, HI, USA.

[19] Ford, W. E. (1994). What is an open architecture robot controller? Intelligent Control, Proceedings of the 1994 IEEE International Symposium on Intelligent Control.

[20] Kuka. KUKA Robot Controller (KR C), [En ligne]. http://www.kuka.com/usa/en/products/controllers/kr_c/start.htm (Consulté le 26 juillet 2006).

[21] OLF. Grand dictionnaire terminologique, [En ligne]. http://w3.granddictionnaire.com (Consulté le 4 juillet 2005).

[22] Fayad, M. E., Schmidt, D. C. (1997). Object-oriented application frameworks. Communications of the ACM, 40(10), 32-38.

[23] IEEE. IEEE Explore, [En ligne]. http://ieeexplore.ieee.org (Consulté le 25 juillet 2006).

[24] OROCOS. [En ligne]. http://www.orocos.org/ (Consulté le 27 juillet 2006).

[25] University of Texas at Austin. Operational Software Components for Advanced Robotics (OSCAR), [En ligne]. http://www.robotics.utexas.edu/rrg/research/oscarv.2/ (Consulté le 27 juillet 2006).

[26] IREQ. MICROB Framework, [En ligne]. http://robotique.ireq.ca (Consulté le 27 juillet 2006).

[27] Katholieke Universiteit Leuven. Katholieke Universiteit Leuven, [En ligne]. http://www.kuleuven.be/english/ (Consulté le 27 juillet 2006).

[28] Kungliga Tekniska Högskolan (KTH). Centre for autonomous systems, [En ligne]. http://www.cas.kth.se/ (Consulté le 27 juillet 2006).

[29] LAAS. Orocos - LAAS, [En ligne]. http://www.laas.fr/~mallet/orocos/ (Consulté le 27 juillet 2006).

[30] University of ULM. University of Ulm, [En ligne]. http://www.uni-ulm.de/indexen.html (Consulté le 27 juillet 2006).

[31] Bruyninckx, H. (2001). Open robot control software: the OROCOS project. Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 21-26 May 2001, Seoul, South Korea.

[32] ORCA. ORCA Robotics, [En ligne]. http://orca-robotics.sourceforge.net/orca_doc_about.html (Consulté le 27 juillet 2006).

[33] OROCOS::SmartSoft. OROCOS::SmartSoft, [En ligne]. http://www.rz.fh-ulm.de/~cschlege/orocos/index.html (Consulté le 27 juillet 2006).

[34] GNU. GNU Lesser General Public License - GNU Project - Free Software Foundation (FSF), [En ligne]. http://www.gnu.org/licenses/lgpl.html (Consulté le 26 juillet 2006).

[35] University of Texas at Austin. Robotics Research Group: The University of Texas At Austin, [En ligne]. http://www.robotics.utexas.edu/rrg/ (Consulté le 27 juillet 2006).

[36] Kapoor, C. (1996). A Reusable Operational Software Architecture for Advanced Robotics. The University of Texas at Austin.

[37] Blain, M. (2002). Plan d'action: Perspective de création de projets et commercialisation de MICROB: Unité Automatisation et systèmes de mesure, Technologie de production, Direction principale de recherche et développement - IREQ.

[38] Lessard, J., April, A., Beauregard, s. et coll. (2000). Hydro-Québec Inspection Robot for PHTS Feeder Pipes Supports. Canadian Nuclear Society's 5th International CANDU Maintenance Conference.

[39] Blain, M., Beruier, M.-A. (1999). Control system with Kalman observer for a submersible vehicle. Proceedings of the 3rd International Conference on Industrial Automation, 7-9 June 1999, Montreal, Que., Canada.

[40] WM ÉTS. Walking Machine - ÉTS, [En ligne]. http://wm.etsmtl.ca (Consulté le 27 juillet 2006).

[41] Robofoot ÉPM. Robofoot EPM, [En ligne]. http://robofoot.polymtl.ca (Consulté le 25 juillet 2006).

[42] SONIA. S.O.N.I.A. - Système d'Opération Nautique Intelligent et Autonome, [En ligne]. http://sonia.etsmtl.ca/ (Consulté le 27 juillet 2006).

[43] SAE Robotique. SAE Robotique - École Polytechnique Montréal, [En ligne]. http://www.saerobot.polymtl.ca (Consulté le 27 juillet 2006).

[44] Budgen, D. (2003). Software design (2nd ed.). New York: Addison-Wesley.

[45] Mathworks. Real-Time Workshop, [En ligne]. http://www.mathworks.com/products/rtw/ (Consulté le 5 juillet 2006).

[46] Scilab. Scilab Home Page, [En ligne]. http://www.scilab.org/ (Consulté le 26 juillet 2006).

[47] National Instruments. NI MATRIXx, [En ligne]. http://www.ni.com/matrixx/ (Consulté le 4 octobre 2006).

[48] Shangying, Z., Junwei, H., Hui, Z. (2004). RCP and RT control of 6-DOF parallel robot. Proceedings of the Fourth International Workshop on Robot Motion and Control, RoMoCo'04, Jun 17-20 2004, Puszczykowo, Poland.

[49] Chen, C.-H., Tsai, H.-L., Tu, J.-C. (2004). Robot control system implementation with rapid control prototyping technique. 2004 IEEE International Symposium on Computer Aided Control Systems Design, 2-4 Sept. 2004, Taipei, Taiwan.

[50] Becerra, V. M., Cage, C. N. J., Harwin, W. S., Sharkey, P. M. (2004). Hardware retrofit and computed torque control of a Puma 560 robot. IEEE Control Systems Magazine, 24(5), 78-82.

[51] Maclay, D. (2000). Click and code [automatic code generation]. IEE Review, 46(3), 25-28.

[52] Grega, W., Kolek, K. (2002). Simulation and real-time control: from Simulink to industrial applications. Proceedings of 2002 IEEE International Symposium on Computer Aided Control System Design.

[53] Ostrovrsnik, R., Hace, A., Terbuc, M. (2003). Use of open source software for hard real-time experiments. 2003 IEEE International Conference on Industrial Technology, 10-12 Dec. 2003, Maribor, Slovenia.

[54] Djenidi, R., Nikoukhah, R., Steer, S. (2001). Code generation in Scicos. Modelling and Simulation 2001. 15th European Simulation Multiconference 2001. ESM'2001, 6-9 June 2001, Prague, Czech Republic.

[55] RTAI. RTAI - Official Website, [En ligne]. https://www.rtai.org (Consulté le 14 juillet 2006).

[56] Bucher, R., Balemi, S. (2005). Scilab/Scicos and Linux RTAI - a unified approach. CCA 2005. Proceedings of 2005 IEEE Conference on Control Applications, 2005.

[57] Bucher R., D. L., Mantegezza P. Rapid Control Prototyping with Scilab/Scicos and Linux RTAI, [En ligne]. http://www.scilab.org/events/scilab2004/final_paper/1-bucher_rtai2scilab.pdf (Consulté le 12 février 2006).

[58] Tona, P. (2006). Teaching Process Control with Scilab and Scicos. American Control Conference, 2006.

[59] Pires, P. S. M., Rogers, D. A. (2002). Free/open source software: an alternative for engineering students. 32nd Annual Frontiers in Education (FIE), 2002.

[60] National Instruments. NI Autocode - Automatic code generation, [En ligne]. http://zone.ni.com/devzone/cda/tut/p/id/3704 (Consulté le 4 octobre 2006).

[61] National Instruments. Labview, [En ligne]. http://www.ni.com/labview/ (Consulté le 4 octobre 2006).

[62] PXI System alliance. PXI Overview, [En ligne]. http://www.pxisa.org/ (Consulté le 4 octobre 2006).

[63] International Electrotechnical Commission. (2001). International Standard ISO/IEC 9126-1. Geneva: International Organization for Standardization.

[64] IREQ. MICROB, Manuel d'application, [En ligne]. http://robotique.ireq.ca (Consulté le 5 juillet 2006).

[65] Pfeiffer, R. From Simulink model to DLL - A tutorial, [En ligne]. http://www.vok.lth.se/~ce/Links/pdf/simulink_dll.pdf (Consulté le 5 juillet 2006).

[66] Martin, R. F. K. Designing a Real-Time PUMA Controller Using Simulink and QNX, [En ligne]. http://www-users.cs.umn.edu/~martin/PUMA/index.html (Consulté le 15 juillet 2006).

[67] Craig, J. J. (2005). Introduction to robotics: mechanics and control (3rd ed.). Upper Saddle River, N.J.: Pearson/Prentice Hall.

[68] Bigras, P. (2003). Notes de cours, Systèmes robotiques en contact (SYS-824).