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.