Université de Montréal

Une plateforme pour le raffinement des services d'OS pour les systèmes embarqués

par

Bruno Girodias

Département d'informatique et de recherche opérationnelle
Facultés des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
En vue de l'obtention du grade de
Maître ès sciences (M. Sc.)
En informatique

Avril, 2005

Université de Montréal
Facultés des arts et des sciences

Ce mémoire intitulé :

Une plateforme pour le raffinement des services d'OS pour les systèmes embarqués

présenté par :

Bruno Girodias

A été évalué par un jury composé des personnes suivantes:

Stefan Monnier
Président rapporteur

El Mostapha Aboulhamid
Directeur de recherche

Gabriela Nicolescu
Codirectrice

Guy Bois
Membre de jury

Mémoire accepté le 18 juillet 2005

RÉSUMÉ

L'industrie des systèmes embarqués est en plein essor. Ces systèmes embarqués deviennent de plus en plus complexes du fait que le coût de fabrication de processeur plus puissant ne fait que diminuer chaque année et donne au concepteur de plus grandes possibilités. Malheureusement, il existe toujours un très grand écart entre la conception, la simulation et l'implémentation.

L'industrie recherche sans cesse des outils et environnements de développement plus performants et efficaces pour la conception de tels systèmes. Des erreurs de conception peuvent s'avérer très graves lorsqu'un tel système est utilisé dans des applications critiques comme dans les environnements hospitaliers, spatiaux ou nucléaires.

Ce mémoire aborde le sujet de codesign, plus particulièrement la partie logicielle de tels systèmes et propose une plateforme pour le raffinement de logiciel pour les systèmes embarqués.

Cette solution permet au concepteur d'avoir une meilleure approche. De plus, elle lui permet une exploration des différents services offerts par un système d'exploitation et lui permet une exploration des différents systèmes d'exploitation sans nécessairement faire un port pour une architecture particulière et sans connaître les différents API attribués à ce système d'exploitation.

Mots clés: Simulation, Cosimulation, OS, Système d'exploitation, Système embarqué, µC/OS-II, cadre d'applications .NET, eSYS.net, C#

SUMMARY

The industry of embedded systems is soaring. These embedded systems become increasingly more complex every day, solely due to the fact that manufacturing costs of more powerful processors decreases every year, thus giving the creators of such systems more possibilities. Unfortunately, a very large gap between the design, simulation and the implementation still exists.

The industry continuously searches for more powerful and effective tools and environments for the design of such systems. Design errors have proven to be critical and potential dangerous, particularly in hospitals, outer space and nuclear environments.

This thesis addresses the issues of co-designing, particularly the software part of such systems and proposes a platform for refinement of software in embedded systems.

This proposed solution makes it possible for the creators to have a better overall approach. This solution allows them to explore the various services offered by an operating system. This also enables them to explore the various operating systems without necessarily making a port for a particular architecture and without knowing the different API from this operating system.

Keywords: Simulation, Cosimulation, OS, Operation System, Embedded System, µC/OS-II, .NET Framework, eSYS.net, C#

TABLE DES MATIÈRES

LISTES DES FIGURES

LISTE DES TABLEAUX

LISTE DES EXEMPLES DE CODES

LISTE DES ABRÉVIATIONS

CIL: Common Intermediate Language
CLB: Configurable Logic Bloc
CLR: Common Language Runtime
CLS: Common Language Specification
CPU: Central Processing Unit
FAA: Federation Aviation Administration
FCL: Framework Class Library
FPGA: Field Programmable Gate Array
HAL: Hardware Abstraction Layer
HDL: Hardware Description Language
IDE: Integrated Development Environment
IOB: Input Output Bloc
IP: Internet Protocol
ISS: Instruction Set Simulator
LUT: LookUp Table
PIP: Propriety Inheritance Propriety
OS: Operation System
RTOS: Real-Time Operation System
SLDL: System Level Description Language
SE: Système d'Exploitation
TCP: Transmission Control Protocol

REMERCIEMENTS

Je voudrais remercier le professeur Dr. El Mostapha Aboulhamid pour m'avoir donné la chance de réaliser ce projet au sein de son laboratoire et au professeur Dr. Gabriela Nicolescu pour avoir codirigé ce projet. Je les remercie de m'avoir guidé et conseillé tout au long de ce projet.

Je voudrais remercier aussi Gracias Hounkpe pour sa collaboration dans ce projet.

Finalement, je voudrais remercier ma famille. Des remerciements à ma fiancée pour ses encouragements et sa patience, à mes parents qui m'ont donné un environnement propice à la réussite et qui m'ont poussé sans cesse et à mes frères et soeurs qui m'ont montré le bon exemple, le courage et la détermination pour réussir.

INTRODUCTION

L'emprise de l'informatique dans notre quotidien est un phénomène incontournable. En regardant autour de nous, nous réalisons que nous sommes entourés de dispositifs utilisant la puissance du calcul. Ces objets ont tous quelques choses en commun; ils ont tous des microcontrôleurs.

Ces microcontrôleurs peuvent se retrouver dans de simples objets comme un four à micro-ondes, des dispositifs comme un téléphone portable, mais aussi dans des objets plus spécialisés comme le Mars Pathfinder ou un stimulateur cardiaque. Ces microcontrôleurs ne se retrouvent pas simplement sur des systèmes traditionnels, mais sur des systèmes embarqués qui sont une combinaison de logiciels et matériels.

Ces systèmes embarqués deviennent de plus en plus complexes du fait que le coût de fabrication de processeurs plus puissants ne fait que diminuer chaque année et donne au concepteur de plus nombreuses possibilités. Avec tous ces systèmes prenant une plus grande part du marché, l'industrie recherche des outils permettant de développer de tels systèmes. La Figure 1 montre le pourcentage du temps consacré à chaque étape de développement et la courbe démontre le coût associé pour réparer une erreur à chaque étape [6].

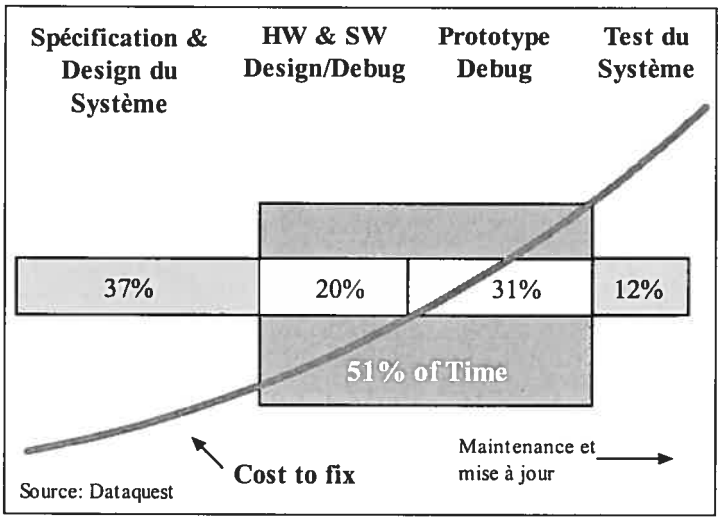


Figure 1: Coût associé au flot de développement

Les concepteurs de systèmes embarqués sont restreints à des contraintes importantes. La quantité limitée de mémoire, de puissance de calcul et de puissance d'énergie ne fait que rendre la tâche plus ardue. La création de systèmes embarqués demande des connaissances non seulement en informatique, mais aussi en génie électrique. Tous ces différents aspects font qu'il est difficile de trouver des outils efficaces. Les outils d'aujourd'hui laissent toujours passer des failles et des erreurs. Quelques exemples sont bien connus dans le domaine:

- Le 4 juin 1996, le lancement du vaisseau spatial européen Ariane fut un échec. Quarante secondes après son décollage, le vaisseau change de trajectoire et par ce fait explose dans les airs. Une enquête a été ensuite ouverte pour trouver quelle était la cause du problème. Après un certain temps, l'enquête a révélé qu'une défaillance dans les caractéristiques et le design du logiciel dans le système de référence de l'inertie en était la cause [17].
- Un exemple un peu plus récent et plus connu est celui du Mars Pathfinder de la NASA. La NASA avait proclamé leur petit robot exempt de défaut. Quelques jours après avoir atterri sur la planète rouge, le robot se mit à faire sans cesse des démarrages complets. À chaque démarrage, des données récoltées étaient perdues. Il s'avérait encore une fois que le problème provenait du logiciel se trouvant dans le Mars Pathfinder. Plus spécifiquement, le problème se trouvait

au niveau du système d'exploitation temps réel utilisé où une inversion de propriété figeait l'application [17].

1.1 Problématique

Ces exemples nous montrent qu'il est important de développer des outils qui nous permettraient d'éviter, de détecter et de prévoir des erreurs comme celles que nous venons de voir. Le facteur humain joue un très grand rôle dans la conception, c'est pourquoi il est important que ces outils apportent une sorte de méthodologie et une meilleure approche pour guider les concepteurs dans les différentes étapes de la réalisation de leurs projets.

Les systèmes embarqués comme cité ci-dessus sont fortement liés au matériel, par ce fait beaucoup d'outils actuels utilisent les langages de description de matériel ou de niveau système (SLDL). Les concepteurs essayent d'utiliser ces langages, dits «idéaux» pour la conception de matériel à différents niveaux abstractions, mais le fait est qu'il reste toujours une faiblesse pour la modélisation logicielle. Un bon exemple est donné par SystemC [38], un SLDL bien connu manquant dans des dispositifs pour la modélisation de logiciels. Dans SystemC les fils d'exécution logiciel et matériel sont ordonnancés de la même manière. L'ordonnanceur de SystemC est approprié au matériel, mais n'a pas le concept de préemption qui s'avère critique pour la simulation d'applications multi-tâches complexes. Même dans la version sortante de SystemC, il n'arrive toujours pas à contrôler les fils d'exécution (suspendre, résumer, tuer ...), à créer dynamiquement des tâches, à modéliser un ordonnanceur, à créer dynamiquement des primitives (mutex, sémaphore...) et à supporter la préemption [11].

D'autres outils existent, mais ils se spécialisent dans un système d'exploitation seulement (OS). Un exemple très connu est celui de WindRiver [40] qui s'appelle Tornado et qui sert seulement pour le développement d'applications utilisant leur système d'exploitation temps réel appelé VxWork. Nous avons très vite remarqué que le concepteur ne veut pas nécessairement choisir un système d'exploitation au

début de la conception. Une raison simple est le fait qu'une panoplie de systèmes d'exploitation lui est offerte et il ne sait pas nécessairement lequel répond parfaitement à ses besoins.

Avec l'exploration d'OS (par exemple Nucleus [36], eCOS [30], μC/OS [32]), il a été noté qu'afin d'employer un certain OS, trois étapes doivent être franchises:

- un environnement pour simuler sur la machine hôte doit être trouvé
- on doit apprendre chaque bibliothèque (API) différente de chacun des OS pour développer une application avec celui-ci
- en dernier lieu, il est nécessaire de trouver ou créer un port pour l'OS pour l'architecture de la cible.

Ces trois étapes s'avèrent difficiles, coûteuses et longues à franchir. Le manque de simulateurs ayant une base commune à tous les OSs, rend la tâche très difficile au concepteur pour tester son application. Le fait de ne pas être familier avec les APIs utilisées par le système d'exploitation permet l'intrduction d'erreurd lors de la conception. Et la tâche qui s'avère encore la plus difficile est de créer un port pour l'architecture souhaitée. Pour créer un port, le développeur doit connaître et maîtriser l'architecture matérielle exécutant l'OS et connaître le fonctionnement interne de l'OS. La tâche se montre encore plus difficile lorsque l'OS est un système fermé. Parfois il faut payer des montants exorbitants pour avoir accès aux informations nécessaires pour porter un OS. Tout ceci entraîne un besoin de modèle de haut niveau pour les OS dans le marché. Il est difficile de trouver une représentation de haut niveau d'un système d'exploitation. Ce niveau d'abstraction nous permettrait d'explorer certains services de systèmes d'exploitations dont nous avons besoin dans notre application ou design. De même, nous pourrions explorer différents systèmes d'exploitation sans nécessairement avoir besoin d'apprendre comment programmer avec ses bibliothèques. Ensuite, il nous serait possible de tester le comportement de notre application sur ce système d'exploitation. Cette abstraction permettrait surtout d'éviter d'investir du temps et de l'argent pour l'exploration d'OS.

1.2 Objectifs

Plusieurs des caractéristiques (simulateur, modèle de haut-niveau et méthodologie) présentées ci-dessus que nous considérons importantes manquent dans les outils existants. Ces caractéristiques répondent à un besoin général de l'industrie qui demande des outils efficaces pour le développement de systèmes embarqués.

Dans le cadre d'un projet du laboratoire d'analyse et de synthèse des systèmes ordinés (LASSO), un langage de description de matériel a été développé. Ce langage s'appelle eSYS.net[14]. Ce dernier est idéal pour décrire le matériel, mais comporte des lacunes pour la description du logiciel.

Dans ce contexte, cette maîtrise a pour objectif d'explorer les différents modèles de simulation logiciels et matériels et de définir différents niveaux d'abstraction pour essayer de répondre au besoin de l'industrie. Comme un langage de description de matériel a été développé par le groupe de recherche, notre attention portera plus spécialement sur le côté logiciel ainsi que son interaction avec le matériel. Nous définirons et implanterons différents niveaux d'abstractions de modèles de simulation. Par la suite nous ferons une étude de cas avec μC/OS.

1.3 Contributions

Nous définirons ensuite une plateforme pour le raffinement, intégrant des niveaux d'abstraction élevés de systèmes d'exploitation et basés sur un environnement standard (cadre d'applications .NET). La plateforme pour le raffinement permet l'exploration d'une application embarquée, l'exploration de différents services d'OS et l'exploration en détail d'un OS choisi. La plateforme permet par profilage d'avoir une estimation des ressources pour le prochain niveau. Tous ces dispositifs sont dans un environnement multilangage qui permet l'utilisation de mécanismes d'interopérabilité pouvant être employés lors de la cosimulation. Ainsi, nous allons

définir un modèle de simulation logicielle/matérielle et le mettre à l'épreuve avec une application qui comporte des caractéristiques d'un système embarqué typique.

De plus, comme eSYS.net a été développé avec le cadre d'applications .NET, notre projet sera intégré dans ce même environnement. Comme nour verrons plus tard, cette infrastructure nous apporte beaucoup de bénéfices.

1.4 Plan du document

Le chapitre 2 introduit les concepts de base et l'environnement de travail. Une introduction sur les systèmes embarqués, sur eSYS.net et sur l'environnement .NET permettra au lecteur d'avoir une meilleure compréhension de ce document.

Le chapitre 3 présente l'état de l'art du domaine. Plusieurs articles ont été publiés sur le sujet et quelques-uns seront introduits brièvement. Certains proposent différentes méthodes de modélisation, d'autres présentent la problématique et certains font un survol sur les différentes techniques de codesign. Une brève discussion sur les différents articles suivra.

Le chapitre 4 décrira l'approche utilisée dans ce projet. Les choix de cette méthodologie y seront détaillés. Les différentes étapes de conception seront décrites avec plus de détails ainsi que les différents niveaux d'abstraction.

Le chapitre 5 décrit les différentes classes et fonctions dans la bibliothèque créée pour la simulation d'un système d'exploitation. Les avantages et inconvénients d'utiliser C# ou tout autre langage de haut niveau seront discutés dans ce chapitre.

Le chapitre 6 montre une application illustrative utilisant notre plateforme pour son développement. Ce chapitre montre un exemple permettant de comprendre la méthodologie derrière la plateforme pour le raffinement.

Le chapitre 7 expose nos différentes expérimentations et résultats obtenus. Ce chapitre montrera également les différentes forces et faiblesses de ce modèle.

Le chapitre 8 relate des travaux futurs. Les objectifs atteints seront énumérés et une courte discussion sur l'ensemble du projet conclura ce document.

CONCEPTS DE BASE/ENVIRONNEMENT DE TRAVAIL

Avant de rentrer dans le vif du sujet, une bonne introduction aux concepts de base serait utile pour le lecteur de ce document. Nous commençons donc par présenter les différentes techniques de validation de logiciels et de systèmes d'exploitation.

2.1 Modèle de simulation pour logiciel embarqué

Tout d'abord, nous introduirons les différentes approches pour la validation de systèmes embarqués. Nous retrouvons dans ces approches des techniques de simulation ainsi que des modèles de systèmes d'exploitation. Deux grandes catégories distinctes s'y retrouvent:

1. la validation de systèmes par la technique des simulateurs du processeur (ISS)
2. la validation de systèmes par la technique de la simulation native.

Les simulateurs de processeurs sont largement utilisés, car ils permettent de simuler le système avec une très grande précision. Cette précision vient du fait que le simulateur émule la fonctionnalité du processeur cible. La modélisation du processeur se fait au niveau du jeu d'instructions. En général, les ISS sont composés d'une structure de données représentant l'état du processeur. Cette structure se situe dans la mémoire de la machine hôte. Ensuite, le ISS s'occupe de faire les étapes de chargement, décodage, d'aiguillage et d'exécution de chaque instruction qui lui est envoyée. Chaque instruction est donc interprétée par le ISS d'où vient le nom de ISS interprétatif. Cette technique s'avère très mauvaise au point de vue de la performance, du fait que le simulateur doit interpréter chaque instruction. Les ISS d'aujourd'hui peuvent, exécuter entre 10 000 et 100 000 instructions par seconde [42]

qui, pour une application de grosseur moyenne, n'est pas suffisant. Les performances de ISS ont été un peu plus accélérées avec l'introduction des ISS compilés sur la machine hôte[43]. Au lieu d'interpréter les instructions, le ISS prend les instructions et les associe avec les instructions de la machine. Les performances sont nettement améliorées, mais ce type d'ISS n'est pas portable et utilise beaucoup de mémoire, ce qui fait que cette technique n'est pas souvent utilisée. Dans les deux techniques citées ci-dessus, les ISS interprétatifs sont toujours les plus répandus, car ils procurent au concepteur une validation plus précise [43].

La simulation native, qui est moins utilisée que la technique ci-dessus, simule le logiciel sans tenir compte de l'architecture de notre processeur cible. Comme le logiciel est indépendant du processeur, il peut être écrit dans un langage de plus haut niveau. Cette technique demande parfois un modèle de simulation pour un système d'exploitation dans le cas où l'application est composée de plusieurs tâches et d'un système d'exploitation.

La technique par simulation native permet une validation rapide du logiciel, mais le compromis se fait au niveau de la précision. Cette technique est plus performante, mais comme elle ne tient pas compte de l'architecture, ceci demande plus de travail au développeur pour extraire les informations sur les temps d'exécution du logiciel à simuler.

Tel que mentionné dans les paragraphes précédents, les modèles de simulation de systèmes d'exploitation sont utilisés pour la simulation de logiciels embarqués complexes. Cette approche est de plus en plus utilisée, car elle permet une validation plus rapide du logiciel embarqué. Les différentes caractéristiques des principaux types de modèles seront décrites ci-dessous.

Dans le modèle du système d'exploitation abstrait, comme son nom l'indique, le système d'exploitation peut être abstrait et fournit un environnement de simulation qui s'occupe entre autres de l'ordonnancement des tâches. L'application sera

représentée en plusieurs tâches parallèles et les primitives de communication entre celles-ci seront données par les bibliothèques offertes par le langage.

Dans le modèle du système d'exploitation virtuel, le modèle simule le comportement du système d'exploitation final. L'objectif de ce modèle est de faire une simulation rapide de la fonctionnalité du système d'exploitation final et de l'application qui s'exécutera dessus. Ce modèle consiste à une abstraction du système d'exploitation à un plus haut niveau où simplement la fonctionnalité et le comportement sont respectés.

Finalement, dans le modèle de la réalisation finale du système d'exploitation, l'exécution est faite directement sur l'architecture finale. Ceci permet de faire des tests précis ainsi que de faire les derniers ajustements pour le bon fonctionnement de l'application.

Notre recherche se situe dans la catégorie de la validation par la technique de la simulation native. Pour la simulation native du système d'exploitation, notre recherche utilise un modèle du système d'exploitation abstrait et un modèle du système d'exploitation virtuel.

2.2 Systèmes embarqués

Un système embarqué est un système composé d'éléments informatiques matériels et logiciels qui sont dédiés à une tâche spécifique. Le fait que ces systèmes doivent répondre à un besoin précis, le concepteur de systèmes embarqués doit souvent faire face à beaucoup de contraintes. Les systèmes doivent parfois traiter beaucoup de flots de données dans un court laps de temps. Le système demande parfois, des temps de réponse rapides, surtout dans une situation où le système doit traiter des informations en temps réel. Le système doit être stable et fiable sans avoir besoin d'intervention extérieure. Les composantes logicielles sont souvent limitées par la quantité de puissance de calcul et de l'espace mémoire. De leur côté, les composantes matérielles

sont souvent limitées par la consommation électrique restreinte. De plus, le concepteur est souvent contraint par le coût du projet.

2.3 Environnement .NET et langage C# [10]

Le cadre d'applications .NET est une plateforme de développement offrant des classes prédéfinies et un moteur d'exécution qui facilitent le développement d'applications. C# est un langage de programmation de haut niveau de Microsoft propre à l'utilisation du cadre d'applications .NET.

Les caractéristiques de la plateforme .NET et du langage C# sont l'objet de publications, d'analyses et de modifications par un organisme international de standard appelé European Computer Manufacturers Association (ECMA). Ceci permet à de tiers partis de développer des outils autour de ce langage et plateforme.

La plate-forme de développement .NET a été présentée en 2000 par Microsoft. Les langages de programmation officiels pour le développement sont C#, VB.NET et Managed C++. D'autres langages peuvent interagir avec le cadre d'applications en autant qu'ils respectent le Common Language Specification (CLS). Tous les concepts de .NET sont offerts dans le Framework Class Library (FCL). Tous les langages respectant le CLS sont compilés vers un langage intermédiaire (Common Intermediate Language) permettant d'interagir avec les bibliothèques .NET. Comme .NET est un langage interprété avec un compilateur Just in Time (JIT), la machine virtuelle est portable.

Le cadre d'applications .NET offre un environnement:

- distribué de programmation orientée objet où le code peut être sauvegardé localement, mais être exécuté soit localement, soit à distance.
- qui aide au déploiement de logiciels et qui gère les conflits de version.
- qui s'assure de la sécurité d'exécution de code y compris du code venant d'un tiers parti.

- d'exécution qui élimine les problèmes entourant les performances d'environnement interprétés ou programmés en scripts.
- riche de types d'applications comme les applications Windows et les applications Web.
- qui gère toutes les communications normalisées par l'industrie pour assurer le bon fonctionnement et l'intégration d'applications basées sur le cadre d'applications .NET avec n'importe quel autre code.

Le cadre d'applications .NET est composé de deux éléments : un environnement d'exécution appelé le Common Language Runtime (CLR) et une bibliothèque de classes appelée le Framework Class Library (FCL). Le FCL est construit sur le CLR et offre tous les services dont une application moderne aurait besoin.

Le CLR gère l'exécution du code, la gestion de la mémoire, des fils d'exécution et l'environnement distribué. Il assure une stricte sécurité des types pour garantir un code sécurisé et robuste. Tout cela mène au concept que le code qui s'exécute sur le CLR est un code géré. Un code géré peut profiter de tous les bienfaits du cadre d'applications .NET.

Le FCL s'occupe d'offrir une collection de classes orientées objet, de types réutilisables. Il nous fournit toutes les bibliothèques nécessaires pour développer des applications en ligne de commande, des interfaces graphiques incluant des applications qui exploitent ASP, XML et l'Internet.

Les deux concepts qui vont être les plus utilisés sont les techniques d'interopérabilité et la gestion de fils d'exécution. Ces deux concepts sont expliqués plus en détail dans ce qui suit.

Toute application contient au moins un fil d'exécution. Dans le cadre d'applications .NET, on retrouve la distinction entre les fils d'exécution physiques et logiques. Les fils d'exécution physiques sont les fils d'exécution du système d'exploitation. Le cadre d'applications .NET introduit le concept de fils d'exécution

logique. Le fil d'exécution peut avoir cinq différentes propriétés d'exécution: «Highest», «AboveNormal», «Normal», «BelowNormal», «Lowest» et peut se trouver dans les états suivants : «Unstarted», «Running/Background», «SuspendRequest», «Suspended», «WaitSleepJoin», «AbortRequested», «Aborted» et «Stopped». Ces différents états sont présentés dans la Figure 2 .

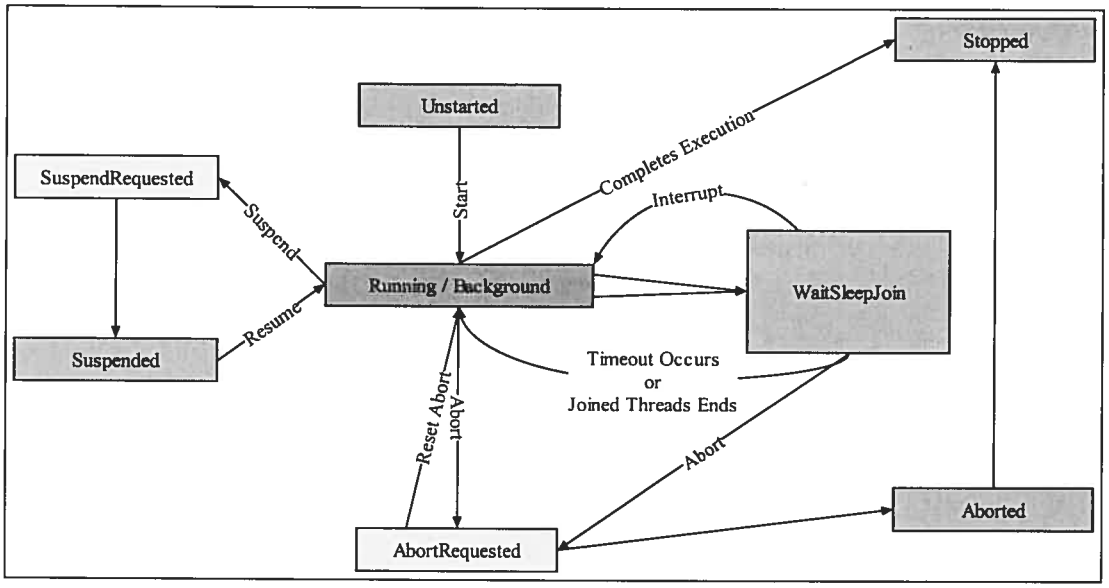


Figure 2: Diagramme d'état des fils d'exécution dans .NET

L'ordonnancement des fils d'exécution dans l'environnement .NET est fait automatiquement. L'algorithme d'ordonnancement ressemble à un tourniquet. Des bibliothèques de synchronisation sont fournies au développeur comme les mutex et les monitors.

Un nouveau concept que le cadre d'applications .NET introduit est du code géré et non géré. Du code géré est du code qui voit son exécution être gérée par le CLR. À n'importe quel moment de l'exécution, le moteur d'exécution peut être arrêté pour récupérer des informations sur l'adresse de l'instruction courante du CPU et obtenir des informations sur l'état du moteur d'exécution, des registres et de la mémoire. Du code non géré est simplement du code binaire, exécuté sur la machine hôte et qui n'est pas gérée par le CLR.

L'idéal serait que tous les programmes soient écrits en code géré, mais ce n'est pas le cas. Pour interagir avec du code non géré, le cadre d'applications .NET offre une multitude de techniques d'interopérabilité. On y retrouve: les fonctions statiques, la mémoire partagée, les sockets, le COM, le JIT et les PInvoke. Dans le cadre de cette recherche, le PInvoke a été choisi comme technique d'interopérabilité et sera la seule technique dont nous discuterons dans ce document [21]. Le choix de cette technique est justifié dans le chapitre 4 sur la plateforme pour le raffinement.

Le PInvoke qui découle de l'anglais Plateform Invocation Services, permet à C# d'accéder les fonctions, structures dans un DLL écrit en code non géré. Par exemple, la fonction MessageBox (voir sa signature dans le Code 2) peut être appelée en ajoutant dans le code géré les parties illustrées dans le Code 3.

```
01   int MessageBox(HWND hWnd, LPCTSTR lpText, LPCSTR lpCaption, UINT uType);
```

Code 1: Fonction MessageBox()

```
01   [DLLInport("user32.dll")]
02   static extern int MessageBox(int hWnd, string text, string caption, int
     type);
03   [DLLInport("user32.dll")]
04   static extern int MessageBox(int hWnd, string text, string caption, int
     type);
```

Code 2: Exemple d'enregistrement d'une fonction

Le PInvoke s'occupe de trouver et de récupérer le DLL et associe la déclaration avec la bonne fonction. Le CLR s'occupe de convertir les paramètres et la valeur de retour de la fonction entre les types .NET et les types non gérés.

Le PInvoke s'occupe également de faire des appels de fonctions entre du code géré au code non géré, mais permet aussi d'appeler une fonction gérée par un code non géré. Ce concept s'appelle un CallBack. Nous avons mentionné précedemment que le CLR s'occupait de convertir les paramètres. Pour un délégué, son équivalent en code non géré est un pointeur de fonction. Donc, pour faire appeler une fonction gérée dans du code non géré, il suffit de passer à un PInvoke un délégué comme

paramètre. Il faut toutefois s'assurer que la signature du délégué et du paramètre est la même. Le Code 3 présente ce concept.

```
01  delegate bool CallBack(int hWnd, int lParam);
02  [DLLInport("user32.dll")]
03  static extern int EnumWindows(CallBack hWnd, int lParam);
04  static bool PrintWindow(int hWnd, int lParam) {…}
05  static void Main() {
06   CallBack e = new CallBack(PrintWindow);
07   EnuWindows(e,0);
08  }
```

Code 3: Exemple de PInvoke et CallBack

2.4 eSYS.net

La bibliothèque intitulée eSYS.net a vu le jour au dernier quart de 2003 dans le Laboratoire d'Analyse et de Synthèse des Systèmes Ordinés de l'Université de Montréal. eSYS.net est un outil de développement de matériel qui permet entre autres la modélisation de description de matériels, mais aussi la modélisation d'algorithmes logiciels et de modélisation de systèmes et réseaux sur puce.

eSYS.net, acronyme pour Embedded System .NET, est une bibliothèque pour le cadre d'applications .NET de Microsoft. La bibliothèque eSYS.net ajoute au cadre d'applications de .NET les concepts fondamentaux comme les signaux, modules et ports. Un noyau de simulation est disponible pour simuler et exécuter les designs conçus, avec eSYS.net. eSYS.net a été développé en respectant les standards de Microsoft.

Le code source eSYS.net est disponible. Le maintien du code est géré par www.eSYS-net.org et quiconque peut l'enrichir. Développé par James Lapalme dans le cadre d'un projet de maîtrise [14], il est le fruit d'une collaboration entre professeurs et étudiants de l'Université de Montréal et de l'École Polytechniques de Montréal.

L'apparition de eSYS.Net a été motivée par le fait que les outils actuels pour le développement de matériel comportent certaines lacunes. Les langages comme VHDL et Verilog, pionniers dans le domaine, restent d'actualité, mais ne permettent plus d'explorer pleinement un modèle et de modéliser un système logiciel. De plus, les outils de synthèse, d'analyse et de simulation ne sont pas disponibles facilement et sont souvent très coûteux. Développé en 1999, SystemC présente de nombreux avantages sur ces prédécesseurs, mais ce système a aussi ses limites. Son développement progresse lentement, car toutes les modifications doivent être revues et acceptées par un comité. Comme SystemC, une librairie pour la conception système, est basé sur le langage C++ et que ce langage est très vaste, il est parfois difficile de prévoir le comportement du simulateur. Par ailleurs, il est difficile de spécifier précisément certains concepts de description de matériel. Mais le principal inconvénient de SystemC, c'est de ne pas avoir un mécanisme d'introspection permettant de concevoir des outils efficaces de synthèse et des CAD. L'arrivée de SystemVerilog montre bien qu'il existe une forte demande pour de nouvelles solutions. SystemVerilog répond à plusieurs besoins, mais il demeure un environnement d'accès difficile et coûteux [27].

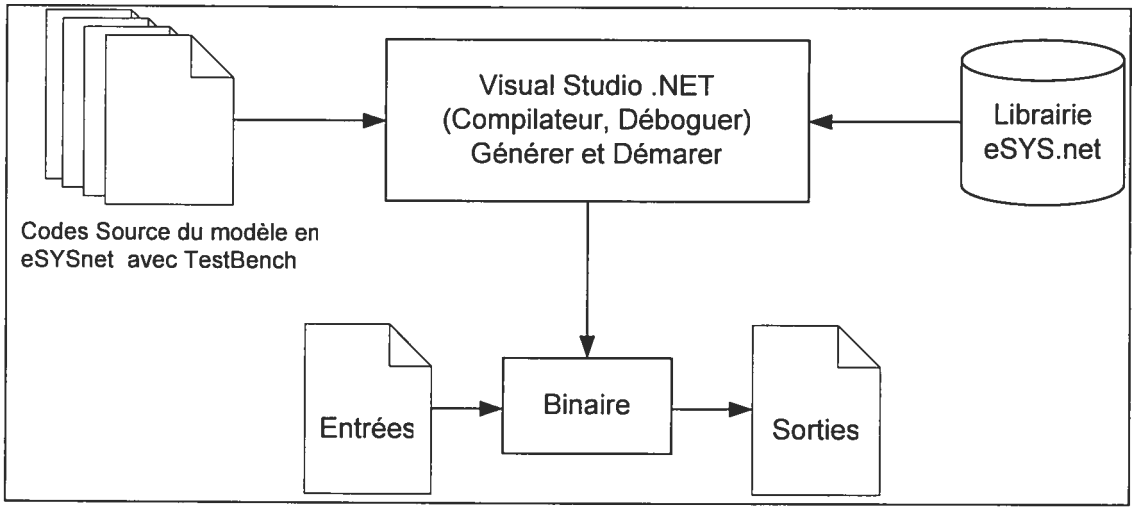


Figure 3: Environnement de développement de eSYS.net [14]

Il est facile de programmer avec eSYS.net, ce qui s'explique par le cadre d'applications .NET qui comporte plusieurs syntaxes de programmation. La Figure 3 présente l'environnement de développement de eSYS.net. Il est possible de développer son modèle en C++, J#, C# et VB.NET. À noter que tout autre langage qui respecte le standard ECMA [29] et l'infrastructure .NET peut être utilisé. eSYS.net est une programmation orientée objet dont la structure ressemble beaucoup à SystemC. Les programmeurs en SystemC n'auront aucune difficulté à développer en eSYS.net. La méthodologie adoptée par eSYS.net est la même que SystemC. Jusqu'à ce jour et tel qu'illustré dans la Figure 4, le développeur fait la description d'un design en C et ensuite le convertit manuellement dans un langage HDL, une conversion souvent difficile et coûteuse, susceptible de générer plusieurs erreurs d'implémentation dans le design. Avec une méthodologie comme celle présentée par la Figure 5, le développeur programme dans un seul environnement qui lui est souvent familier et il peut librement raffiner son design et le décomposer aisément en plusieurs composantes distinctes [2]. À l'intérieur d'un même environnement, il est alors possible de programmer sur différents niveaux. La puissance d'une telle méthodologie permet de développer des designs complexes en très peu de code, de les simuler facilement et d'appliquer les mêmes banc d'essai pour les divers niveaux en un minimum de temps [14].

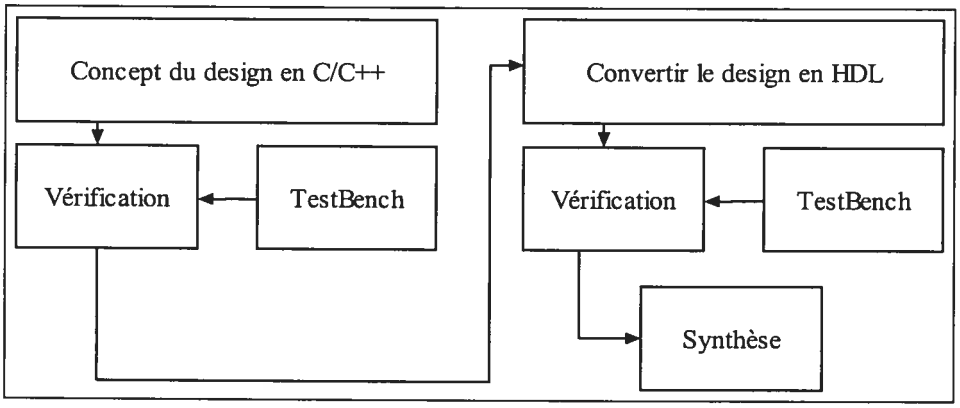


Figure 4: Méthodologie de développement utilisée jusqu'à ce jour

Pour le développement rapide, eSYS.net permet de programmer dans un niveau d'abstraction très élévé. eSYS.net est composé d'un groupe de classes abstraites englobant la plupart des concepts au niveau matériel [15]. Il offre les concepts de

modules, processus, port, interface, signaux, événement, horloges, simulateur ainsi que certains concepts plus avancés tels que les canaux.

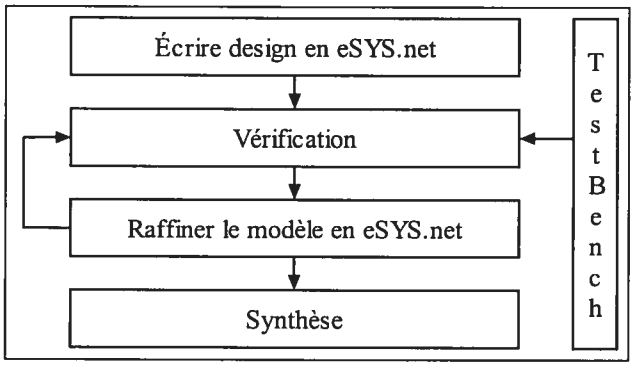


Figure 5: Méthodologie de eSYS.net

Un design consiste en des modules interconnectés par des canaux de communication par l'intermédiaire d'interfaces. Un module est l'entité de base pour un design. Il possède un certain nombre d'entrées et de sorties et on y retrouve un ou plusieurs processus qui donnent la fonctionnalité du module. Un module peut être représenté comme une boîte de traitement dépendant de ses entrées et sorties.

La structure d'un module en eSYS.net est présentée ci-dessous. Cette structure modulaire comprend : la déclaration des ports et des interfaces (lignes 3-4), la déclaration de variables locales (lignes 6), un constructeur de modules (ligne 8) et de divers processus (ligne 11). Toutes les déclarations de modules doivent dériver de la classe *BaseModule* de eSYS.net. Cette classe permet l'enregistrement automatique du module auprès du simulateur. L'enregistrement auprès du simulateur se fait automatiquement dès qu'un module, un canal, un événement ou une interface est instancié dans le design. La définition d'un module dans eSYS.net ressemble à ceci:

```
01  public class Producer: BaseModule
02  {
03   public Clock clock;
04   public write_if output;
05
06   private bool m_verbose;
07
08   public Producer(string name_,bool verbose):base(name_){m_verbose=verbose;}
09
10   [Process()] [EventList("posedge","clock")]
```

```
11    public void main(){}
12    }
```

Code 4: Exemple de Module dans eSYS.net

Les différents modules communiquent par l'entremise des interfaces ou «ports». Le concept de port en eSYS.net est différent de celui de SystemC. Dans eSYS.net, les ports ne sont pas des objets, mais des interfaces définies dans des modules spécialisés appelés « canaux ». Certaines interfaces sont déjà prédéfinies par eSYS.net. (Exemples: *inInt*, *outInt inoutInt*, *inBool*, etc.) On peut observer dans le Code 5 une déclaration d'un port de sortie (ligne 4). Une autre façon de communiquer entre modules se fait par l'entremise de signaux. Un signal représente un fil qui interconnecte un ensemble de composantes. Un signal transporte des données, mais ne se détermine pas la direction du transfert comme les ports. Les signaux ne sont pas déclarés avec un attribut *in*, *out*, *inout*. La direction du transfert dépend des associations aux ports respectifs.

```
01    public IntSignal signal_in;
```

Code 5: Exemple de signal dans eSYS.net

Différents signaux sont prédéfinis dans eSYS.net. Chaque type de donnée standard comme les entiers, les booléens, les entiers longs et plusieurs autres sont à la disposition du programmeur. L'usager peut définir ses propres signaux, mais il doit dériver de la classe *BaseSignal* qui permet de conserver la valeur ancienne, actuelle et future du signal. La valeur mise dans un signal ne sera pas disponible avant le prochain delta cycle. Un concept intéressant de eSYS.net est la visibilité des signaux; si un signal est déclaré avec le mot clé «public», ce signal peut être vu par tous les modules, mais si le mot clé «private» est utilisé, la visibilité du signal sera restreinte au module où il a été déclaré. À noter qu'en C#, «private» est le mot clé par défaut si rien n'a été mis à la déclaration.

Contrairement à SystemC les signaux ne sont pas des canaux. La structure d'un canal ressemble beaucoup à celle d'un module. En fait, les canaux doivent dériver de la classe *BaseChannel* qui, elle-même, dérive de *BaseModule*. *BaseChannel* permet

la mise à jour à la fin du cycle. Un canal est un objet qui possède des concepts de communication et de synchronisation. Un canal doit implémenter une ou plusieurs interfaces. La définition d'un canal dans eSYS.net ressemble à ceci :

```
01  public class Fifo: BaseChannel,write_if,read_if
02  {
03   public Clock clock;
04
05   public Fifo(string name_,bool verbose):base(name_){m_verbose=verbose;}
06
07   public void write(char c){…}
08   public char read(){…}
09   public void reset(){…}
10   public int num_available(){…}
11
12   private bool m_verbose = false;
13   private enum e {max = 10};
14   private char[] data = new char[(int)e.max];
15   private int num_elements,first;
16   private Event write_event = new Event();
17   private Event read_event = new Event();
18  }
```

Code 6: Exemple de canal dans eSYS.net

Les développeurs de eSYS.net n'ont pas jugé nécessaire de définir des canaux primitifs comme dans SystemC. Si le futur impose l'ajout de canaux primitifs ou d'autres concepts, il suffira d'enrichir eSYS.net de nouvelles classes. Ceci est un des grands avantages d'un tel outil, car on ne fait qu'enrichir la bibliothèque pour changer la sémantique de la simulation. Il est difficile de faire la même chose avec des HDLs.

Dans l'exemple du Code 6, deux événements ont été déclarés (ligne 16-17). Les événements permettent d'actionner les différents processus se retrouvant dans les modules. Un processus peut soit attendre un certain événement en utilisant la fonction *Wait()*, soit réveiller un autre événement en utilisant la fonction *Notify()*. eSYS.net définit deux types de processus. Le premier genre, les «PMethod», équivaut au «sc_method» de SystemC et le corps du processus est exécuté au complet. Ce type de processus ne conserve pas le contexte d'exécution. Le deuxième type équivaut aux «sc_thread» et aux «sc_cthreads». Le corps du processus est contrôlé par les

fonctions *Wait()* et *Notify()* appliquées sur les événements. Voici comment est déclaré un processus dans eSYS.net:

```
01   [Process()][EventList("posedge", "clock")]
02   public void main(){…}
```

Code 7: Exemple de processus dans eSYS.net

Le type du processus est indiqué avant la déclaration du processus ainsi que la liste de sensibilité. Cette syntaxe particulière vient du cadre d'applications .NET. Cette syntaxe s'appelle les attributs. Un processus peut être sensible à «posedge» et donc sensible à un front montant, à «negedge», à «sensitive» et à «transaction». Ces deux derniers sont identiques, mais pour l'événement «transaction», le processus est toujours réveillé même si la valeur n'a pas changé au dernier delta cycle.

Maintenant que le design est fait, il est utile de vérifier la fonctionnalité du modèle. Un banc d'essai est facilement implanté comme montré dans le Code 8:

```
01   public class top: SystemModel
02   {
03    public Clock clk;
04    public Fifo fifo_inst;
05    public Producer prod_inst;
06    public Consumer cons_inst;
07
08   public top(ISystemManager manager):base(manager)
09   {
10    clk = new Clock("Clock",10);
11
12    fifo_inst = new Fifo("Fifo");
13    prod_inst = new Producer("Producer");
14    cons_inst = new Consumer("Consumer");
15
16    prod_inst.output = fifo_inst ;
17    cons_inst.input = fifo_inst;
18
19    fifo_inst.clock = clk;
20    prod_inst.clock = clk;
21    cons_inst.clock = clk;
22   }
```

Code 8: Exemple de banc d'essai dans eSYS.net

Dans un banc d'essai, on inclut la déclaration des modules (lignes 3-4), l'initialisation des modules (lignes 10-14) et l'association des interfaces de chaque

module (lignes 16-22). Le banc d'essai doit dériver de *SystemModel* pour indiquer au simulateur que cette classe est le banc d'essai. Pour démarrer la simulation du modèle, il reste à écrire une classe qui lancera la simulation. Cette classe est définie comme suit:

```
01  public class Simulator_Main
02  {
03   static public void Main()
04   {
05      Simulator sim = new Simulator();
06      top fifo_tb = new top(sim);
07      sim.sysm = fifo_tb;
08      sim.Run(10);
09   }
10  }
```

Code 9: Exemple de Main dans eSYS.net

Dans cette classe se retrouvent la création du simulateur (ligne 5), l'instanciation du banc d'essai (ligne 6), l'association du banc d'essai avec le simulateur (ligne 7) et le lancement de la simulation (ligne 8). Cette classe est l'équivalent de la classe «sc_main» de SystemC.

L'ordonnanceur de eSYS.net ressemble beaucoup aux autres ordonnanceurs utilisés dans le monde du matériel. C'est un ordonnanceur basé sur les événements. Quelques ajouts ont été amenés par les développeurs de eSYS.net; différents points d'ancrage sont disponibles sur la simulation et des méthodes définies par l'usagé peuvent être exécutées avant l'exécution d'un processus. La structure de eSYS.net est présentée dans la Figure 6.

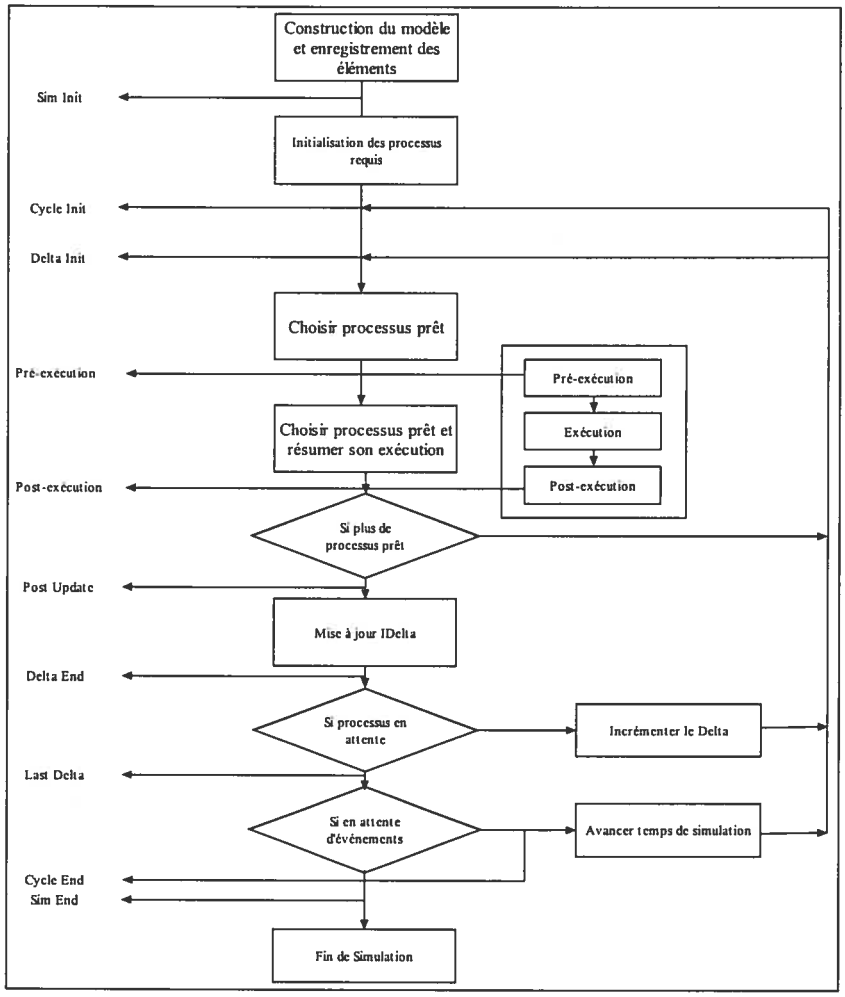


Figure 6: Structure de eSYS.net

Les différents points d'ancrage et méthodes pré/post processus sont représentés par des attributs. Voici la liste d'attributs que l'on retrouve dans eSYS.net. La Figure 7 présente chaque attribut avec une brève description et indique les concepts auxquels ils s'appliquent.

Attribute	Description		Applied to concept
Process	Associate a thread to a class method		Class Method
PMethod	Associate a method process to a class method		Class Method
EventList (list of events)	Add sensitive list for a process		Process
ManualRegistration	Manually registration of the element		Field
PreCall (Name of Method)	Directives indicating methods execution in explicit points of the execution flow	Method to be called before the process	Process
PostCall (Name of Method)		Method to be called after the process	Process
SimInit (Name of Method)		Simulation init	Class Method
SimEnd (Name of Method)		Simulation end	Class Method
CycleInit (Name of Method)		Cycle initialization	Class Method
CycleEnd (Name of Method)		Cycle and	Class Method
DeltaInit		Delta cycle initialization	Class Method
DeltaEnd		Delta end	Class Method
FinalDelta		Last delta	Class Method
Reset		Simulator reset	Class Method

Figure 7: Attributs avec leur rôle dans eSYS.net [14]

eSYS.net est présentement en cours de développement. Des bibliothèques de synthèse seront apportées au langage et un moteur d'assertion est prévu.

ÉTAT DE L'ART

Dans le chapitre précédent, nous avons présenté différentes techniques pour la validation des systèmes embarqués. On retrouve dans les travaux existants l'utilisation de la technique de validation native et la technique de validation à base d'un simulateur du processeur (ISS). Plus particulièrement, ces travaux utilisent le système d'exploitation abstrait et le système d'exploitation virtuel pour la validation native. Pour la validation à base d'un ISS, la plupart des travaux utilisent la réalisation finale du système d'exploitation.

En utilisant un système d'exploitation abstrait dans la simulation native, le système d'exploitation abstrait est fourni par des environnements de simulation (ex. SystemC). Ces modèles permettent, tel qu'indiqué auparavant, une validation rapide accompagnée d'une grande perte de précision [19].

L'industrie propose deux approches pour la technique par simulation native en utilisant un système d'exploitation virtuel. L'approche de CarbonKernel [41] et SoCOS permet la modélisation de systèmes d'exploitation, mais le code du système virtuel ne sera pas le même que le code du système d'exploitation final. Ceci entraîne que le concepteur doit ensuite faire valider son code final pour s'assurer des résultats de son application embarquée. L'autre approche est celle proposée par WindRiver Systems [40]. Cette compagnie fournit VxSim un simulateur de leur système d'exploitation. Le problème avec cette approche est qu'elle ne nous permet pas de tester du matériel ajouté à notre design, et par le fait même, ne nous permet pas d'évaluer les performances de communication entre le logiciel et le matériel.

Pour la technique de validation à base d'un simulateur du processeur (ISS) avec la réalisation finale du système d'exploitation, le ISS affecte grandement les performances de simulation et le temps de conception [19].

Comme nous venons de voir, la technique de validation par simulation native ne tient pas vraiment compte de la notion de temps et une solution est proposée dans [23] pour améliorer la précision. Cette solution permet d'utiliser la simulation native du système d'exploitation avec la réalisation finale du système, tout en y rajoutant des informations temporelles. Cependant, les auteurs de l'article considèrent toujours que leur modèle n'est pas assez précis. Également, ce modèle reste fortement relié à SystemC. Cette recherche a été ensuite poussée par [1]. Ce dernier présente un modèle de cosimulation qui permet de faire une simulation logicielle rapide et précise en combinant une simulation native avec une cosimulation matérielle et logicielle temporisée. Cette méthode permet la simulation avec notion de temps des interruptions du processeur pendant l'exécution native d'une partie logicielle. De plus, ce modèle permet d'avoir plusieurs niveaux de granularité dans la notion de temps pour faire le compromis entre la performance et la précision d'une simulation. Ce modèle reste à être mis à l'épreuve sur des exemples concrets pour démontrer son efficacité.

Dans [12], un cosimulateur rapide et flexible pour système embarqué est présenté. Le cosimulateur supporte tous les services offerts par le standard japonais ITRON pour les systèmes à temps réel. Le logiciel est simulé d'une façon native et le matériel est simulé à l'aide d'un simulateur HDL. Leurs travaux futurs se concentrent sur l'amélioration de la notion de temps et la précision de leur modèle. La Figure 8 présente le modèle.

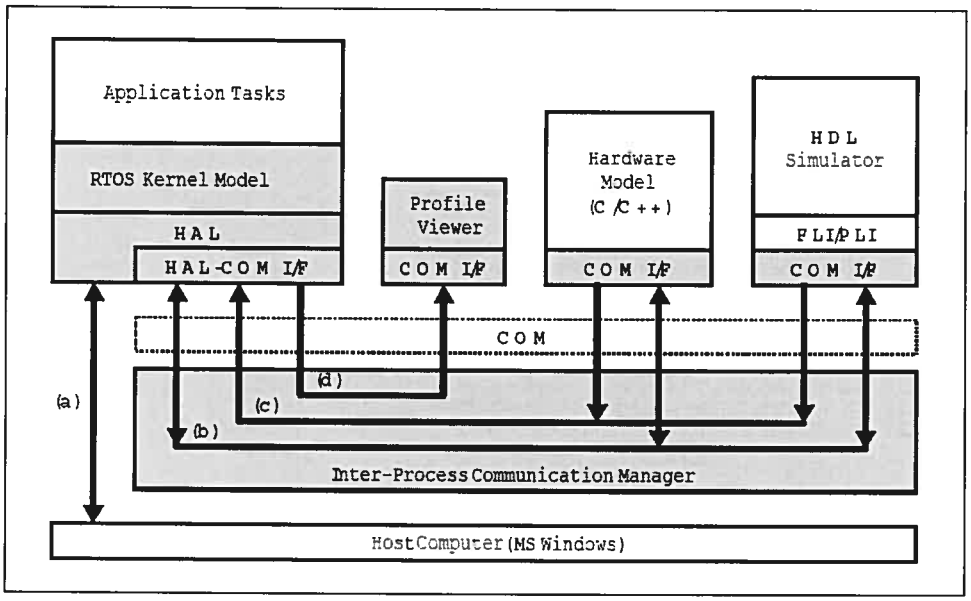


Figure 8: Vue générale du cosimulateur RTOS-centric [12]

L'outil Tornado dans l'environnement VxSim comme vu plutôt n'offre pas la possibilité de faire de la cosimulation. Dans [4], on démontre une solution pour un environnement de cosimulation avec le simulateur VxSim. Il utilise un outil appelé Seamless. La combinaison de Seamless avec VxSim, Modelsim et un modèle de PCI Transactor semble être un bon outil, mais la partie logicielle est limitée. Le concepteur doit utiliser VxWork comme système d'exploitation. La Figure 9 présente le modèle.

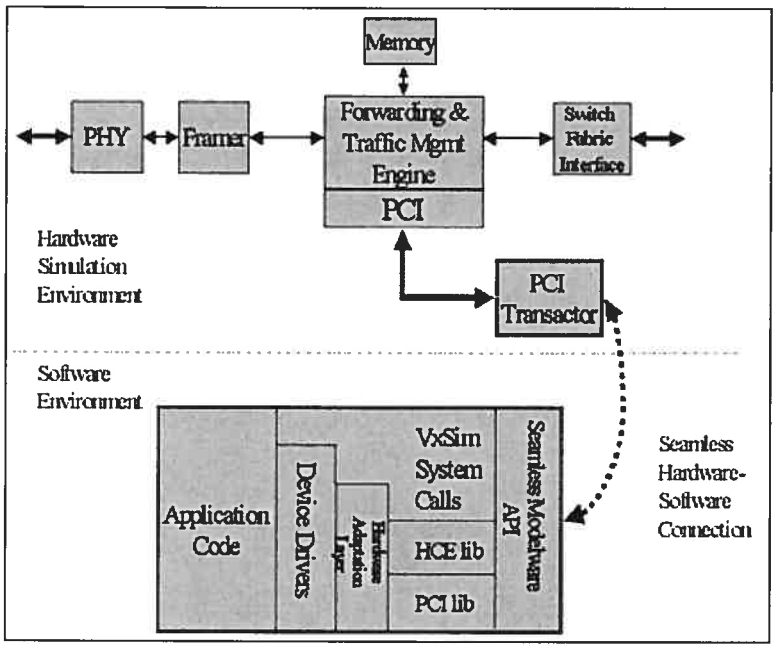


Figure 9: Environnement du cosimulateur avec VxSim [4]

Depuis son arrivée, uC/OS-II ne cesse d'être utilisé comme outil d'apprentissage et de recherche. Dans [20], un simulateur pour uC/OS-II sur la plateforme Win32 a été réalisé. Ce simulateur de RTOS est composé de deux parties: la première partie (WCOS) s'occupe d'exécuter le RTOS avec une application embarquée tout en respectant les caractéristiques d'uC/OS-II et de ses API; la seconde partie (WSIM) s'occupe d'offrir des fonctions pour exécuter et contrôler la simulation et simule le comportement de la plateforme matérielle. La Figure 10 présente le modèle de simulation.

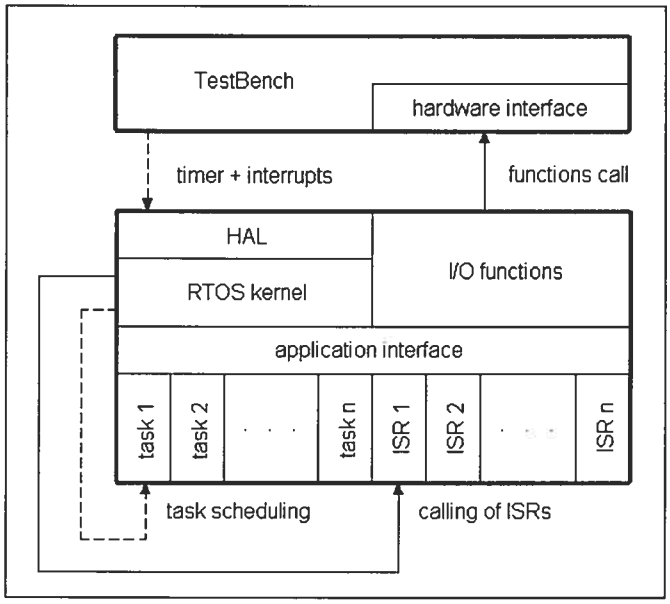


Figure 10: Structure du simulateur WSIM [20]

La partie TestBench utilise l'API de WSIM pour générer le rythme de simulation, aviser la partie logicielle d'une interruption matérielle et s'occuper de simuler le comportement du matériel hôte. Elle permet de contrôler la simulation, de changer le contenu de la mémoire, des entrées et des sorties du système et de visualiser les temps de réponse et les différentes statistiques du RTOS. La seconde partie est l'application même avec l'API de WCOS qui englobe le noyau d'uC/OS-II. Ce simulateur WSIM/WCOS permet de faire du temps réel avec un pas d'exécution d'une milliseconde. Il permet d'utiliser les outils de débogage de Windows et le code de l'application reste presque la même de la simulation à la cible. Le fait que le cosimulateur comporte trois composantes le rend moins flexible et malléable.

Dans [5], une plateforme pour le raffinement du logiciel et matériel est présentée. Cette plateforme permet le partitionnement de matériel et logiciel en se basant sur les résultats de la simulation du modèle. SystemC est utilisé pour la modélisation des parties matérielles et logicielles. La plateforme comporte un système d'exploitation à temps réel et d'un ISS. Leur travail peut être perçu comme étant similaire à cette recherche, mais leurs niveaux d'abstraction sont plus bas. Nous considérons leur

recherche complémentaire à la nôtre, mais la flexibilité et l'introspection offertes par l'environnement .NET sont absentes.

Dans [15], l'auteur fait part d'une des difficultés pour la simulation de systèmes d'exploitation - porter un système d'exploitation sur une autre architecture. L'auteur nous explique qu'afin de pouvoir porter un RTOS, le processeur doit posséder quelques conditions minimales. Pour uC/OS-II, le processeur doit avoir un compilateur pour le langage C qui supporte du code réentrant. Le processeur doit supporter des interruptions et pouvoir lancer des interruptions à intervalles régulières. Des fonctions C reliées au processeur doivent exister pour activer et désactiver le mécanisme d'interruption. Le processeur doit supporter une pile matérielle qui peut contenir un bon nombre de données. Le processeur doit posséder, dans son jeu d'instructions, des mécanismes de chargement et de stockage du pointeur de la pile et des registres dans la mémoire ou dans la pile. Toutes ces informations ne sont pas nécessairement évidentes à trouver. De plus, faire un port d'un RTOS demande une bonne compréhension de l'architecture et de bien connaître son jeu d'instructions.

Pour développer des outils de simulation, il existe différentes plateformes intéressantes dans le domaine de la recherche, dont ProActive [37] qui est une librairie Java offrant un environnement pour la programmation parallèle et distribuée. ProActive offre beaucoup de caractéristiques intéressantes, mais toutes ses caractéristiques se retrouvent dans le cadre d'applications .NET. Certaines caractéristiques proviennent de bibliothèques externes de .NET, mais la plupart sont des parties intégrales de .NET. Ce standard offre un net avantage au cadre d'applications .NET sur la plateforme ProActive.

Caractéristiques de ProActive	Bibliothèque	.NET
Asynchronous calls: Typed Messages (Request and Reply)	System.Remoting + System.Messaging MSMQ	Yes
Automatic future-based synchronizations: wait-by-necessity	SCOOPLI	Yes
Migration, Mobile Agents	System.Remoting + System.Web	Yes
Remote creation of remote objects	System.Remoting	Yes
Reuse: polymorphism between standard objects and remote objects	System.Remoting	Yes
Group Communications with dynamic group management	System.Remoting + System.Web	Yes
Libraries for sophisticated synchronizations, collaborative applications	System.Threading	Yes
Transparent, dynamic code loading (up and down)	System.Reflection + System.CodeDom	Yes
XML Deployment Descriptors	System.XML	Yes
Security Framework	System.Security	Yes

Tableau 1: ProActive vs .NET [37] [22] [35]

Le Tableau 1 montre les principales caractéristiques de ProActive et leur équivalent dans le cadre d'applications .NET. En plus de posséder les caractéristiques de ProActive, le cadre d'applications .NET offre beaucoup plus comme un environnement standard ECMA et ISO [29].

Les recherches mentionnées ci-dessus possèdent des caractéristiques intéressantes et utiles, dont plusieurs peuvent être facilement intégrées dans notre environnement flexible. L'originalité de notre travail est que notre approche n'est pas limitée à un type d'OS et tient compte d'un haut niveau abstraction pour les composantes logicielles. Elle offre des possibilités d'explorer aussi bien que d'évaluer l'exécution à différents niveaux d'abstraction.

PLATEFORME POUR RAFFINEMENT

Ce chapitre décrit notre plateforme pour le raffinement qui guidera le concepteur dans les étapes de réalisation d'un système embarqué et l'aidera à éviter des erreurs de conception.

Si l'on prête attention aux différents concepts offerts par les systèmes d'exploitation (sémaphore, mutex, monitor, évènement, fils d'exécution, ...), on remarque que plusieurs d'entre eux se retrouvent plus souvent dans les langages de programmation de haut niveau comme Java ou C#. Ces langages viennent souvent avec des environnements qui leur permettent d'avoir beaucoup d'autres fonctionnalités. Avant de présenter notre plateforme pour le raffinement, notre choix de l'environnement est exposé et celui-ci sera présenté, car il est à l'origine de notre plateforme.

4.1 Environnement

Cette recherche se base beaucoup sur l'environnement offert au concepteur pour le développement d'un système embarqué. Un environnement idéal pour le concepteur doit permettre de: décrire facilement les composantes logicielles, programmer aisément dans un environnement qui aide à éviter les erreurs de conception, faire de l'introspection pour faciliter le débogage, le profilage et l'analyse de système, annoter le code pour faciliter la modélisation, compiler vers un langage intermédiaire standardisé pour avoir une bonne fondation pour le développement d'outils, avoir un environnement multi plateformes et multi langages pour de la flexibilité et portabilité et gérer facilement et automatiquement la mémoire pour faciliter et accélérer la conception de systèmes.

4.2 Cadre d'applications .NET

Il a été constaté que le cadre d'applications .NET respecte toutes les caractéristiques mentionnées plus tôt et en offre d'avantages. On retrouve dans le cadre d'applications .NET les éléments suivants: un langage de haut niveau pour la programmation propre et simple, la réflectivité pour l'introspection, les métadonnées pour l'annotation, un langage intermédiaire commun (CIL) pour une norme de base pour des outils, une infrastructure de langage commun (CLI) pour des environnements multiplateforme et multilangage et un ramasse-miettes pour la gestion de la mémoire. Les dispositifs comme le ramasse-miettes, la réflectivité, les métadonnées et le CLI définissent le concept du code géré présenté par .NET.

Dans [14], un nouveau SLDL a été défini et les avantages du cadre d'applications .NET dans ce contexte ont été prouvés. Notre recherche prend l'origine de ce projet et y rajoute la simulation et le raffinement de composantes logicielles.

Niveau d'abstraction	Composantes logicielles			Composantes matérielles	Raffinement nécessaire pour avancer dans la méthodologie
	Spécification des modèles	Abstractions	Environnement d'exécution		
Exploration de l'application	Description de haut niveau (c.-à-d., C#, J#)	OS, Architecture matérielle exécutant le logiciel	Cadre d'applications .NET	Description de haut niveau (c.-à-d., C#, J#)	Partitionnement logiciel matériel
Exploration de système d'exploitation	Groupe de tâches communiquant à l'aide d'API générique	OS Final, « bis »	Cadre d'applications .NET + API pour la simulation de l'OS exploré	Modèle comportemental (c.-à-d., eSYS.net) communiquant à l'aide de l'API .NET	Sélection de l'OS, composantes matérielles description comportemental e
Exploration détaillée de système d'exploitation	Groupe de tâches communiquant à l'aide d'API propre à un OS	Codes OS propre à l'architecture matérielle, « bis »	Cadre d'applications .NET + Windows OS	Modèle comportemental (c.-à-d., eSYS.net, SystemC, VHDL) communiquant à l'aide de l'API d'interopérabilité .NET	Port Windows de l'OS

Tableau 2: Résumé des différents niveaux

4.3 Niveaux d'abstraction

La plateforme présentée se compose de plusieurs niveaux d'abstraction. Ces niveaux d'abstraction permettent au développeur d'adopter une meilleure approche pour la conception de systèmes embarqués. Une conception progressive pas à pas permet au développeur de se concentrer sur différentes facettes du design, de détecter des bogues et des problèmes au début du flot de conception du système final. Présentement, notre plateforme pour le raffinement comporte trois niveaux distincts

qui se nomment: exploration de l'application, exploration de systèmes d'exploitation et exploration détaillée de système d'exploitation spécifique. Les différents niveaux sont résumés dans le Tableau 2 et seront présentés dans les prochaines sections.

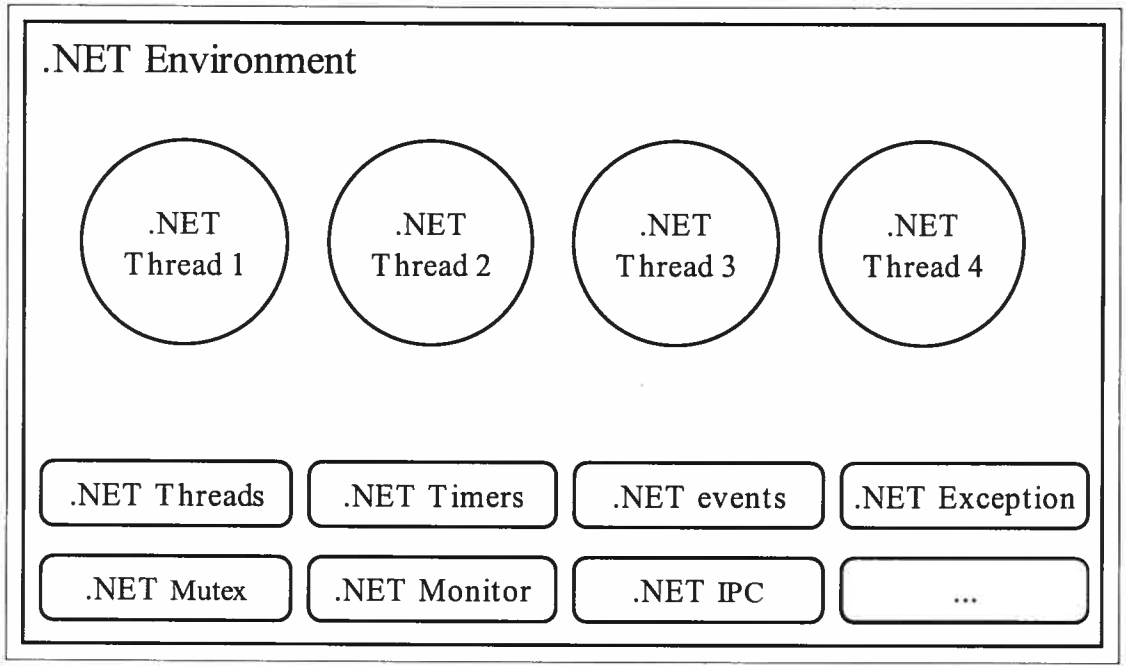


Figure 11: Exploration de l'application

4.3.1 Exploration de l`application (Niveau 1)

Au premier niveau, le système embarqué est décrit dans une langue de haut niveau (c.-à-d., C #, J #) comme représenté sur la Figure 11. À ce niveau, le matériel et le logiciel n'ont pas été définis et seule la fonctionnalité de conception est vérifiée. Les fils d'exécution .NET sont employés pour décrire le modèle à ce niveau. Le mécanisme d'ordonnancement est lancé en lançant chacun des fils d'exécution un par un. L'ordonnancement des différents fils d'exécution est pris en charge par .NET et les contraintes de temps sont ajoutées en employant les bibliothèques de .NET. Puisqu'il n'y a aucune distinction de matériel et de logiciel, il n'y a aucun mécanisme de communication nécessaire.

À ce niveau, le système d'exploitation est complètement abstrait. Le développeur qui veut simuler son application se sert des classes offertes par le cadre d'application .NET. Pour la simulation de différentes tâches OS, la classe .NET Threads peut être utilisée. Des mécanismes de synchronisation comme les Mutex et les Monitors sont offerts par .NET ainsi que des temporisateurs, des événements, des structures de données et des mécanismes de communication. Un langage de haut niveau comme C# et un environnement comme .NET offrent la plupart des composantes se trouvant dans un système d'exploitation. Tout ceci permet au développeur de tester rapidement la fonctionnalité de son application.

Au niveau 1, par un processus itératif et en employant des outils de profilage, le concepteur peut créer différents fils d'exécution simulant le comportement de son système embarqué et peut identifier des erreurs de fonctionnalité, les sections critiques, des insuffisances d'exécution et des ressources qui ont besoin de synchronisation pour avoir une meilleure idée du partitionnement de l'application. Tout ceci lui permettra de se faire une meilleure idée sur les composantes qui doivent être matérielles et celles qui doivent rester logicielles. Bien sûr, à ce niveau, il n'est pas possible d'obtenir une précision au niveau du temps.

Tout ceci est possible grâce au mécanisme de réflexibilité offert par l'environnement .NET. Ce mécanisme permet à des outils d'explorer et même modifier les structures de haut niveau pendant l'exécution d'un code. Ce mécanisme se trouve souvent dans les langages de haut niveau comme C# et Java.

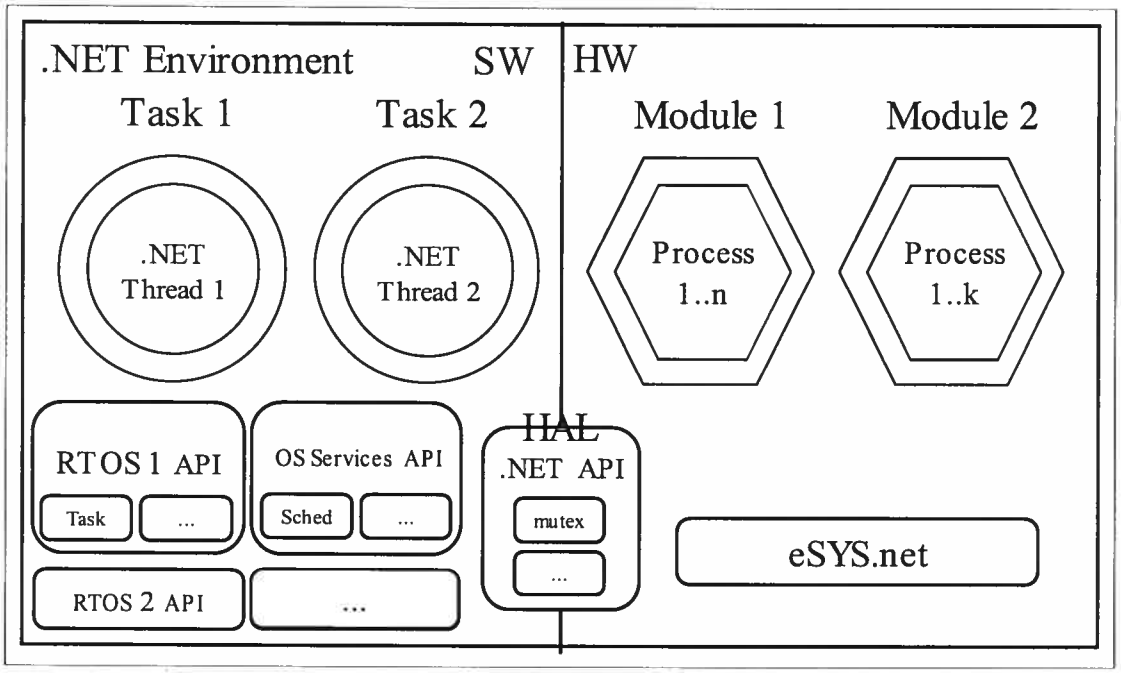


Figure 12: Exploration de système d'exploitation

4.3.2 Exploration de système d'exploitation (Niveau 2)

Au deuxième niveau, le système embarqué est décrit par un langage introspectable. À ce niveau, l'OS n'est pas encore déterminé et l'architecture du processeur exécutant le logiciel est complètement abstraite. Comme représenté sur la Figure 12, le modèle logiciel présente des APIs génériques (OS indépendant) qui encapsulent quelques API de .NET (c.-à-d., gestion de fils d'exécution). Le concept de tâche est introduit à ce niveau, où des tâches et des modules sont employés pour décrire le modèle. Le mécanisme d'ordonnancement et les contraintes de temps sont activés dans notre plateforme en appelant les fonctions appropriées dans l'API générique employé pour représenter l'OS ou les services. Ainsi, le concepteur peut configurer la plateforme pour un OS donné et l'API appelé dans le modèle logiciel respectera la fonctionnalité de cet OS. Par différentes configurations, le concepteur peut comparer le comportement de son application sous différents OS. Pour l'exploration d'OS, notre plateforme doit fournir des bibliothèques de simulation consistante de modèles fonctionnels pour chacun des OS différents. Nous avons appelé notre bibliothèque initiale SharpOS puisqu'elle est écrite entièrement en C#. Ici, eSYS.net sera employé

pour simuler la composante matérielle puisque ce SLDL réside dans l'environnement de .NET. Le matériel et le logiciel communiquent en employant l'API .NET.

Par conséquent, nous employons un modèle natif de haut niveau pour la partie logicielle. Ce modèle facilite la tâche au développeur en l'aidant à explorer et à valider différents services d'OS aussi bien que d'explorer différents OS sans l'effort d'apprendre comment fonctionnent les API des OS. Puisque l'architecture de la cible est encore complètement abstraite, les développeurs n'ont pas besoin de trouver un port de l'OS, car ils peuvent simuler leur comportement. En étant si riche avec des dispositifs comme le ramasse-miettes, l'environnement de .NET est connu pour être non déterministe au point de vue d'exécution (bien que certains considèrent ceci un problème) [18]. Les tests dans un environnement non déterministe peuvent aider les développeurs en trouvant des erreurs de fonctionnalité dues à l'intercalage de processus. Tout ceci permet au développeur de s'orienter vers un système d'exploitation qui répond à ses besoins. Ce niveau ne lui permet pas de faire des simulations avec un système d'exploitation optimisé pour son application, mais lui permet plutôt d'explorer facilement plusieurs OS. Chaque système d'exploitation reste générique dans le sens qu'il ne comporte aucune optimisation. La précision de temps est un peu améliorée, mais il n'est toujours pas possible d'obtenir des précisions de temps au niveau temps réel. À notre connaissance, il n'est pas possible de trouver un simulateur permettant de simuler une application avec différents OS.

À ce niveau, le mécanisme .NET de réflexibilité est toujours présent puisque les différents API pour ce niveau sont développés en C#. Ce mécanisme, à l'aide du mécanisme d'annotation offert par .NET, permet de faire une exploration approfondie du système d'exploitation choisie pour la simulation. Les interblocages, les inversions de priorité, les changements de contexte et d'autres mécanismes d'OS peuvent être observés lors de la simulation grâce à un environnement tel que celui de .NET.

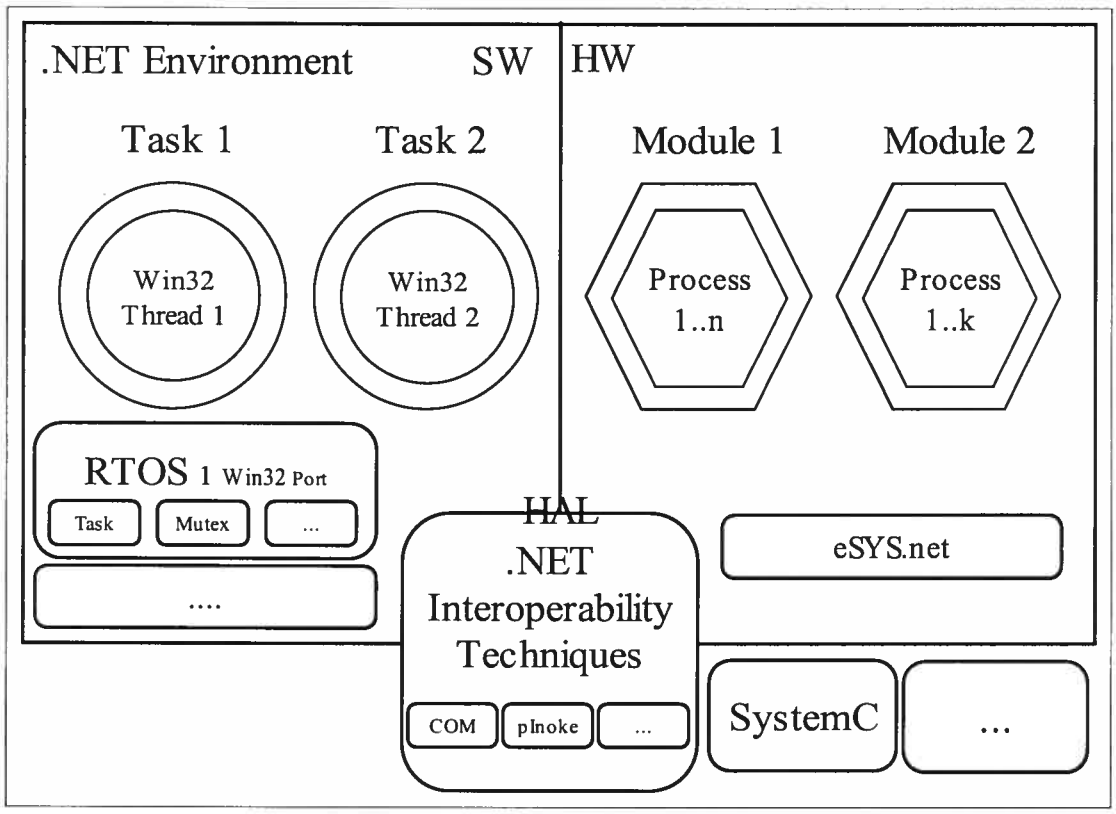


Figure 13: Exploration détaillée de système d'exploitation spécifique

4.3.3 Exploration détaillée de système d'exploitation spécifique (Niveau 3)

Au troisième niveau, le système embarqué est décrit en utilisant différents languages. Comme représenté à la Figure 13, le port de l'OS choisi pour Win32 (encapsule souvent les appels système Win32, c.-à-d. les fils d'exécution Win32) peut être employé pour examiner l'application avec différentes optimisations. De même, des tâches et les modules sont encore employés pour décrire le modèle. Le mécanisme d'ordonnancement et les contraintes de temps sont activés en appelant les fonctions appropriées dans l'API de l'OS choisi. Pour la composante matérielle, eSYS.net peut être sélectionné, où l'environnement est assez flexible pour examiner la composante matérielle avec un SLDL/HDL comme SystemC ou VHDL. Pour la cosimulation du matériel et logiciel, ces deux éléments communiquent en employant les techniques d'interopérabilité de .NET [21]. Comme du code non géré, qui est du code qui n'est pas contrôlé par l'environnement de .NET, est employé pour l'aspect du

logiciel et du code contrôlé est utilisé pour le matériel, les techniques d'interopérabilité offertes par l'environnement de .NET doivent être employées. Étant donné l'environnement .NET, la technologie COM peut être employée, et on peut interagir avec des outils comme ModelSim.

À ce niveau, le développeur simule son application avec les optimisations de l'OS sélectionné. La simulation au niveau 3 procure des informations sur les ressources matérielles comme l'espace mémoire pouvant guider le développeur sur le choix de la cible qu'il doit utiliser pour la réalisation de son projet. Les précisions de temps au niveau temps réel se feront sur la cible. Certains simulateurs intègrent certains ports réel d'OS, mais il est rare de trouver un environnement de simulation comme celui-ci où il est facile d'intégrer différents ports d'OS existants.

En encapsulant les mécanismes d'interopérabilité d'une couche de haut niveau écrite en .NET, des estimations de temps peuvent être obtenues pour la communication entre le matériel et le logiciel grâce encore au mécanisme de réflexibilité.

SHARPOS

Dans ce chapitre est présenté la bibliothèque qui a été développée pour rendre possible la simulation au niveau 2. (Exploration de système d'exploitation). Cette bibliothèque permet d'avoir une représentation à haut niveau d'un système d'exploitation. Un chapitre entier est consacré à ce niveau, car les niveaux inférieurs et supérieurs utilisent des composantes développées par de tierces parties.

Présenté dans le chapitre précédent, le cadre d'applications de .NET nous offre les outils nécessaires pour faire le développement de bibliothèques pour la simulation de systèmes d'exploitation. Le cadre d'applications .NET nous propose différents langages de programmation. Pour ce projet nous avons choisi C#, car il est le langage officiel de Microsoft pour la plateforme et il est le langage qui a été utilisé pour le développement de eSYS.net. L'utilisation du même langage facilite l'intégration avec ce dernier.

On retrouve dans le cadre d'applications .NET le principe de fil d'exécution qui peut être utilisé pour représenter les différentes tâches d'un système d'exploitation. Comme les fils d'exécution de .NET sont associés directement sur les fils d'exécution de Windows, nous ne pouvons pas contrôler l'ordonnancement de ceux-ci. La seule manière qu'un usager puisse contrôler les fils d'exécution est avec l'aide des priorités. Le cadre d'applications .NET n'offre que 5 priorités, qui rendent la tâche difficile pour simuler le comportement d'un système d'exploitation comme µC/OS. Dès le début du développement, il est clair que nous ne pouvions pas réaliser un système d'exploitation à temps réel dur avec l'aide du cadre d'applications .NET. Il est impossible de savoir quand le ramasse-miettes sera lancé et combien de temps il prendra pour s'exécuter. Comme dit plutôt, ceci rend l'environnement .NET non déterministe. Les fonctions sur les fils d'exécution manquent de précision et le

nombre de priorités est insuffisant. Ce sujet est plus longuement discuté dans [18]. Malgré cela, le cadre d'application .NET reste un très bon environement pour le développement. Il nous procure les avantages suivants: un langage de haut niveau, un mécanisme de réflexibilité, un mécanisme d'annotation, un langage intermédiaire commun et une infrastructure de langage commun.

La bibliothèque développée tel que mentionnée précédemment est basée sur µC/OS. On y retrouve les concepts généraux comme l'ordonnancement, la structure de tâche, les sémaphores, les boîtes aux lettres et les événements. Avant de présenter l'API de SharpOS, les sections suivantes présentent brièvement les systèmes d'exploitations à temps réel et µC/OS.

5.1 Système d'exploitation à temps réel

Il existe plusieurs genres de systèmes d'exploitation, mais l'un d'entre eux nous est plus particulièrement important: le système d'exploitation temps réel (RTOS).

Les RTOS sont utilisés quand les différentes applications demandent que la notion de temps soit suivie d'une façon rigide, soit pour contrôler des machines, des instruments scientifiques et des systèmes industriels. Un concept important dans les RTOS est la gestion des ressources. La gestion doit être très précise et la notion de temps doit être déterministe et prévisible. Dans un système d'exploitation régulier, on ne retrouve en général pas de limites de temps, tandis que dans les RTOS les contraintes temporelles sont cruciales.

On peut distinguer deux catégories de contraintes dans les systèmes temps réel:

- contraintes durs: dépasser une limite de temps peut engendrer des résultats erronés et être catastrophique pour le système. Chaque tâche est garantie d'être exécutée à temps. (ex. réacteur nucléaire)

- contraintes douces: dépasser une limite de temps engendre une diminution de précision du système, mais le système est toujours considéré un succès même avec quelques violations de contrainte de temps. (ex. traitement d'images)

5.1.1 µC/OS

Un exemple de RTOS très connu est µC/OS, qui a été développé par Jean J. Labrosse. µC/OS est un RTOS fiable et robuste qui a été certifié par la Federation Aviation Administration (FAA) pour son utilisation dans des systèmes de l'avionique [13].

Lorsque µC/OS s'initialise, les applications du concepteur sont représentées comme plusieurs tâches qui sont créées lors de l'initialisation. Pour chaque tâche, le concepteur doit spécifier le pointeur sur le code de la tâche, un autre sur les données des tâches, l'adresse du début de la pile et degré de propriété. Ensuite, la tâche la plus prioritaire est prête à être exécutée.

Le noyau µC/OS supporte jusqu'à 64 priorités (56 précisément, car certaines sont réservées). Comme deux tâches ne peuvent pas avoir la même priorité, l'utilisateur ne peut avoir que 64 tâches. Plus exactement, le concepteur peut créer 56 tâches, car certaines priorités sont réservées comme la priorité 63 pour la tâche «idle» et la priorité 62 pour la tâche de statistiques qui calcule le pourcentage d'utilisation du CPU. µC/OS offre un mécanisme de prévention d'inversion de propriété et la latence des interruptions et le temps requis pour le changement de contexte respecte les contraintes d'un système à temps réel dur. L'ordonnanceur de µC/OS est préemptif, une caractéristique importante pour un RTOS [15].

Les tâches qui sont prêtes à être exécutées et qui sont en attente sont mises dans une table, classées en fonction de leur priorité. L'ordonnanceur s'occupe d'exécuter la prochaine tâche la plus prioritaire selon un algorithme d'ordonnancement.

Voici les caractéristiques les plus importantes de µC/OS:

- création et gestion de tâches
- création et gestion de moyens de synchronisation
- fonction d'attente de tâches
- changement de propriété des tâches
- effacement des tâches
- suspension et continuation de tâches
- création et gestion de moyens de communication

Pour illustrer un peu le fonctionnement de l'API d'un système d'exploitation comme µC/OS un exemple sur les tâches est présenté. La création d'une tâche se fait par la fonction suivante:

```
01   OSTaskCreate(AppTask1, (void *)(), (void *)&AppTask1Stk[255],10);
```

Code 10: Création de tâche dans µC/OS

Le premier paramètre est le nom de la fonction de la tâche, le second est l'adresse des données, le troisième indique l'adresse du haut de la pile de la tâche et le dernier paramètre indique le degré de priorité de la tâche.

Le corps d'une tâche contient:

- une initialisation de la tâche
- une initialisation des variables
- une boucle infinie contenant la fonctionnalité de la tâche
- une instruction *OSTimeDly()* qui permet de donner la main à d'autres tâches

Voici les autres fonctions principales de µC/OS:

- initialisation: *OSInit()*, *OSStart()*

- gestion des tâches: *OSTaskCreate()*, *OSTaskDel()*, *OSTaskDelReq()*, *OSTaskChangePrio()*, *OSTaskSuspend()*, *OSTaskResume()*, *OSScheclock()*, *OSSchedUnlock()*
- gestion des sémaphores: *OSSemCreate()*, *OSSemAccept()*, *OSSemPost()*, *OSSemPend()*, *OSSemInit()*
- gestion des boîtes aux lettres: *OSMboxCreate()*, *OSMboxAccept()*, *OSMboxPost()*, *OSMboxPend()*
- gestion des files d'attente: *OSQCreate()*, *OSQAccept()*, *OSQPost()*, *OSQPend()*
- gestion d'interruption: *OSIntEnter()*, *OSIntExit()*

µC/OS s'occupe principalement de la gestion de tâches, mais il assure aussi la gestion des interruptions, du temps, de la communication et de la mémoire. Pour plus d'informations sur µC/OS, se référez à [15].

5.2 SharpOS API

La bibliothèque est divisée en plusieurs classes qui sont présentées dans les sections suivantes:

5.2.1 Les tâches: Tasks.cs

Cette classe contient tout ce qui est nécessaire pour créer, modifier, détruire et contrôler les tâches. Chaque tâche µC/OS est associée avec un fil d'exécution de .NET. Dans la structure d'une tâche on retrouve, en autre, sa priorité µC/OS, son nom, un délégué sur la fonction de la tâche et d'autres variables et fonctions utilisées pour le système interne du système d'exploitation. Lors de la création d'une tâche, le fil d'exécution de .NET associé à celle-ci est mis dans un état inactif «UnStarted». Ceci permet à l'ordonnanceur de .NET d'ignorer ce fil d'exécution. Lorsque l'ordonnanceur de µC/OS sélectionne cette tâche comme étant la plus prioritaire, le fil d'exécution .NET associé est mis dans l'état actif «Running». Si cette tâche attend après un événement, il le met dans l'état d'attente «Suspended». Chaque fil

d'exécution .NET est initialisé à la propriété la plus basse «Lowest» lors de la création des tâches. Ceci permet qu'une tâche ne puisse jamais prendre la main sur l'ordonnanceur de µC/OS. Des fonctions *raisePriority()* et *lowerPriotity()* permettent de changer la priorité de la tâche au niveau du fil d'exécution .NET pour empêcher l'ordonnanceur de reprendre la main lorsqu'une tâche rentre dans une section critique. La dernière fonction qui vaut la peine d'être citée est la méthode *Start()*. Cette méthode est composée d'un «case» qui permet selon l'état du fil d'exécution de la tâche de mettre actif «Running» le fil d'exécution.

5.2.2 Les événements: OSEvent.cs

Cette classe sert de base pour tous les types d'événements (sémaphore, mailbox, mutex) se trouvant dans µC/OS et définit les différentes caractéristiques d'un événement. Comme dans µC/OS, on y retrouve une variable *OSEventType* qui sert à déterminer le type d'événement; un compteur *OSEventCnt* qui sert pour le cas d'un événement de type sémaphore, la variable de propriété de propriété d'héritage (PIP) utilisée dans le cas d'un mutex, une variable *OSEventMsg* utilisée pour l'usage de boîte aux lettres. La fonction *EventTaskRdy()* permet de retourner la tâche la plus prioritaire des tâches qui sont en attente d'un événement. Les tâches en attente sont déterminées par la variable *OSEventGrp* et d'un tableau *OSEventTbl*. Ces deux variables sont mises à jour par la fonction *EventTaskWait()*.

5.2.3 Les sémaphores: Semaphore.cs

Cette classe permet de gérer l'accès de ressources partagées. Elle contient un objet *OSEvent* et trois méthodes. Les méthodes *SendPost()* et *SendPend()* permettent à un utilisateur de relâcher et d'obtenir une sémaphore pour une certaines ressources. Dans le cas où une tâche ne peut pas obtenir une sémaphore, la tâche est mise en attente. Si l'utilisateur ne veut pas que la tâche soit bloquée lors de l'obtention d'une sémaphore, la méthode *SemAccept()* essaie d'obtenir une sémaphore, mais ne se bloque pas. La variable *OSEventCnt* de *OSEvent* est utilisée tandis que la variable *OSEventMsg* reste à nulle.

5.2.4 Les boîtes aux lettres: Mailbox.cs

Cette classe permet aux tâches de s'envoyer des messages entre elles. Comme la classe sémaphore, elle contient un objet *OSEvent()* et trois méthodes. Les méthodes *MboxPost()* et *MboxPend()* permettent à un utilisateur d'écrire et de lire un message. Dans le cas où une tâche ne peut pas écrire un message dans la boîte aux lettres, la tâche est mise en attente. Si l'utilisateur ne veut pas que la tâche soit bloquée lors de l'écriture, la méthode *MboxAccept()* essaie d'écrire, mais ne se bloque pas. La variable *OSEventCnt* de *OSEvent* reste à nulle, tandis que la variable *OSEventMsg* est utilisée.

5.2.5 L'ordonnanceur: Scheduler.cs

Cette classe regroupe tout ce qui se rapporte à l'ordonnanceur. Elle contient les fonctions qui s'occupent de l'ordonnancement des tâches. La fonction la plus importante est la fonction *Start()*. Cette fonction détermine la tâche la plus prioritaire à être exécutée et la met active. Pour pouvoir contrôler les différentes tâches, cette fonction utilise quelques tactiques. Tout d'abord, comme dit plutôt, l'ordonnanceur tient la priorité la plus forte, donc aucune tâche ne peut l'interrompre. Lorsqu'une tâche a été choisie pour être exécutée, l'ordonnanceur augmente la priorité de la tâche choisie à «Normal», ce qui la rend plus prioritaire que toutes autres tâches qui sont à «Lower». Dépendamment de l'état de cette tâche, l'ordonnanceur de µC/OS va soit la démarrer, soit la réveiller ou soit la résumer pour qu'elle puisse s'exécuter. Ensuite, l'ordonnanceur, qui est un fil d'exécution .NET, donne la main à la tâche la plus prioritaire en faisant un *Wait()*. Tout le long de l'exécution d'une application, on retrouve seulement deux fils d'exécution actifs et ordonnancés par le cadre d'applications .NET: le fil d'exécution rattaché à l'ordonnanceur et le fil d'exécution rattaché à la tâche la plus prioritaire. Ceci nous permet de garder le contrôle sur l'exécution de fils d'exécution ordonnancé par le cadre d'applications .NET. Comme l'ordonnanceur de µC/OS est plus prioritaire que la tâche, après un certain délai de temps, notre ordonnanceur reprend la main et vérifie si la tâche courante est toujours la plus prioritaire.

5.2.6 Le noyau: Core.cs

Cette classe représente le noyau du système d'exploitation. Toutes les structures représentant l'état du système se retrouvent dans cette classe. Lors de l'initialisation du système, cette classe est responsable de la création du fils d'exécution rattachés à l'ordonnanceur de Micro/OS et le fait ensuite démarrer. On retrouve toutes les fonctions comme la création de tâches et la gestion d'événements. Toutes ces fonctions font ensuite appel à leur classe respective. Les méthodes reliées à la notion de temps: *TimeDly()* et *TimeDlyResume()* se retrouvent également dans cette classe. Ces fonctions permettent de faire dormir une tâche pendant un certain laps de temps. Comme la plupart des fonctions influencent l'état du système, elles doivent être appelées à partir de la classe représentant le noyau.

Dans la Figure 14 suivante est présenté le diagramme de classe de la bibliothèque développée pour ce projet.

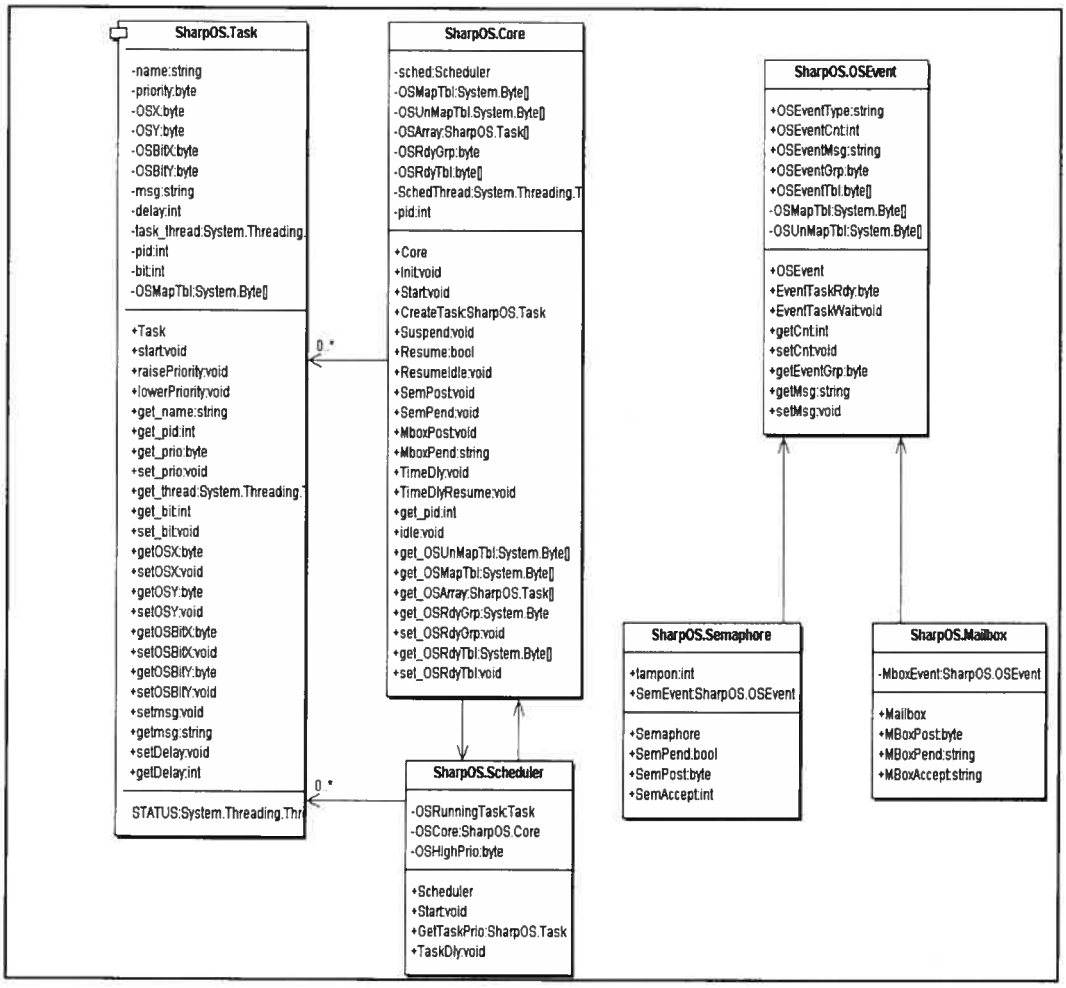


Figure 14: Diagramme des classes du modèle de simulation

5.2.7 État de SharpOS

SharpOS a été développé pour démontrer qu'il était possible de créer une bibliothèque qui permet de simuler le comportement d'un système d'exploitation. De manière générale, les diverses fonctionnalités de µC/OS y sont présentes. Certaines caractéristiques comme les mutex et la gestion d'interruptions propre à µC/OS n'ont pas encore été développées dans la bibliothèque. D'autres caractéristiques comme la gestion de la mémoire et la destruction de tâches, de sémaphores, et de boîte aux lettres n'ont pas été développées, car le cadre d'applications .NET offre un mécanisme pour la gestion de l'espace mémoire.

APPLICATION ILLUSTRATIVE: RÉGULATEUR DE VITESSE

Pour une meilleure compréhension de notre plateforme, nous avons créé une application qui est un modèle simplifié d'un régulateur de vitesse.

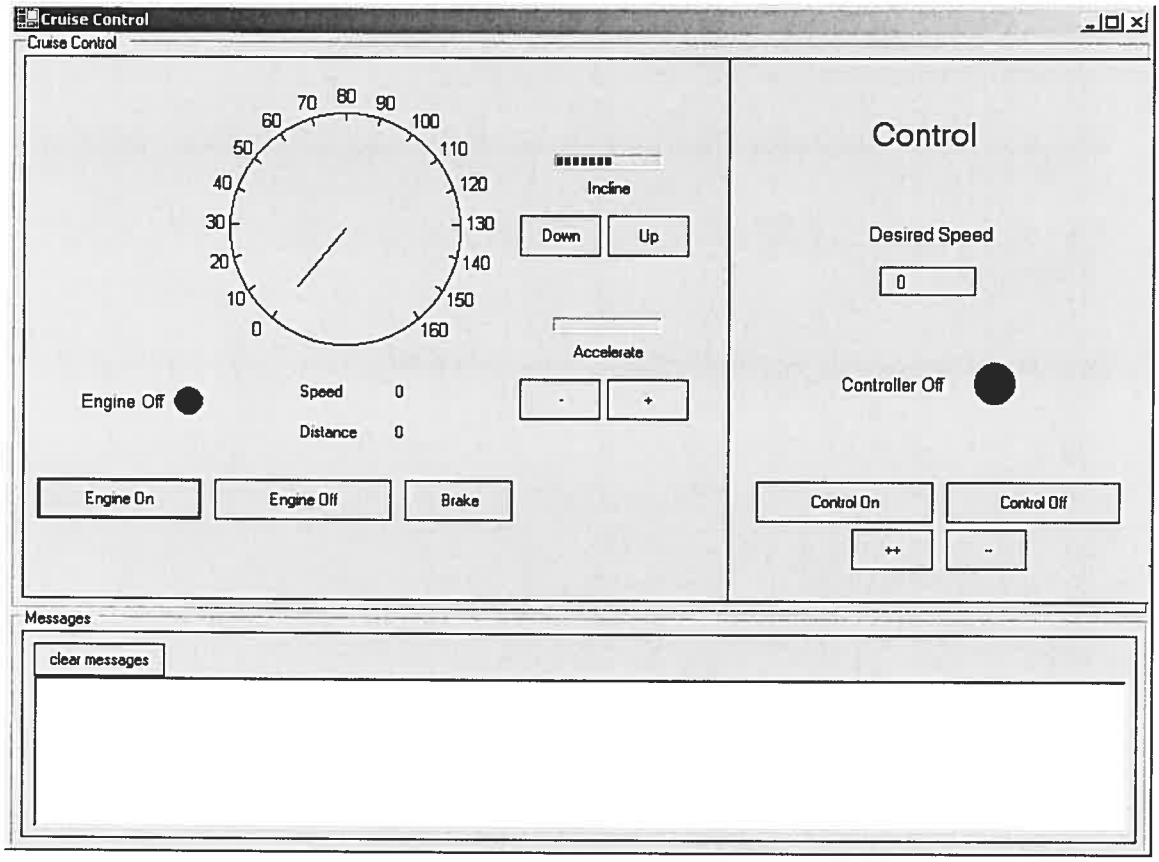


Figure 15: Capture d'écran de l'application

Le modèle du régulateur de vitesse permet à un usager de mettre en marche et d'arrêter le moteur d'une voiture ou activer et désactiver le régulateur de vitesse en poussant des boutons. Quand le moteur est démarré, une boucle calcule sans cesse la vitesse de la voiture. La vitesse de la voiture peut être modifiée par l'accélération qui

peut être augmentée ou diminuée par des boutons. La vitesse peut également être modifiée par l'inclination de la route qui est également commandée par des boutons. La vitesse peut être réglée en activant la commande de croisière. Quand cette dernière est activée, une boucle calcule toujours l'accélération exigée pour maintenir la vitesse désirée. La vitesse peut être augmentée et diminuée par des boutons. Dès que le régulateur de vitesse est désactivé, l'accélération du régulateur de vitesse est mise à zéro et la voiture commence à ralentir. De même, lorsque le bouton de frein est appuyé, le régulateur de vitesse est désactivé et la voiture commence à ralentir, mais puisque l'événement de freiner est important, elle élimine tout autre événement qui a été envoyé.

Comme montrée par la description de sa fonctionnalité, cette application effectue une commande complexe selon les divers calculs qui peuvent être exécutés dans le matériel ou par plusieurs tâches logicielles sous un OS.

```
01  public speed;
02  public acc;
03
04  Thread T1 = new Thread(CalculateSpeed);
05  T1.Start();
06
07  CalculateSpeed()
08  {
09   while(1)
10      Thread.Sleep(10);
11      ...
12  }
13
14   // Update Speed
15  {
16   speed = _speed;
17    ...
18  }
19
20  // Increment Acceleration
21  {
22   acc++;
23    ...
24  }
```

Code 11: Exemple partie logicielle au niveau 1

6.1 Niveau 1: Exploration de l'application

Au niveau 1, l'application est modélisée en J#. La création dynamique de fils d'exécution et des dispositifs, comme arrêter des fils d'exécution, sont employés. Pour chaque niveau, une interface de GUI est employée pour permettre à un utilisateur d'interagir avec l'application. Puisqu'il n'y a aucune architecture à ce niveau, chaque résultat obtenu à partir des calculs est assigné directement aux variables publiques qui sont protégées par des mécanismes de synchronisation. Le Code 11 montre à quoi ressemble le code de l'application au niveau 1.

```
01  private speed;
02  private acc;
03
04  SharpOS.Task T1 = CreateTask("T1", CalculateSpeed ,11);
05
06  OS_Core.Start();
07
08  CalculateSpeed()
09  {
10    while(1)
11      OS_Core.TimeDly(T1,10);
12      ...
13  }
14
15  // Software Update Speed
16  {
17   hw.mem["speed"] = _speed;
18    ...
19  }
20
21  // Hardware Increment Acceleration
22  {
23      hw.mem["acc"] = acc++;
24   sw.Acceleration();
25   ...
26  }
```

Code 12: Exemple partie logicielle au niveau 2

6.2 Niveau 2: Exploration de système d'exploitation

Au niveau 2, tous les aspects qui sont reliés au calcul de la vitesse et de l'accélération sont modélisés en logiciel et le reste qui est relié à l'interaction de

l'utilisateur, tel que l'augmentation d'accélération, le contrôle de la vitesse, et le freinage, a été modélisé en matériel. La partie logicielle appelle des fonctions définies dans l'API de SharpOS. Pour l'instant, tel que mentionné dans le chapitre précédent, nous avons programmé une bibliothèque initiale pour simuler le comportement d'un OS. L'interface entre le matériel et le logiciel est simple puisque ces deux parties partagent le même environnement et plusieurs variables. Pour créer une tâche, la fonction générique *OSTaskCreate()* est appelée où un nom, une fonction et un numéro de priorité doivent être indiqués comme argument à la fonction (voir ligne 4 du Code 12). En appelant la fonction *OSCore.Start()*, l'ordonnanceur de SharpOS est démarré (voir ligne 6 du Code 12). Des contraintes de temps sont représentées par des fonctions comme *OSCore.TimeDly()* (voir ligne 11 du Code 12).

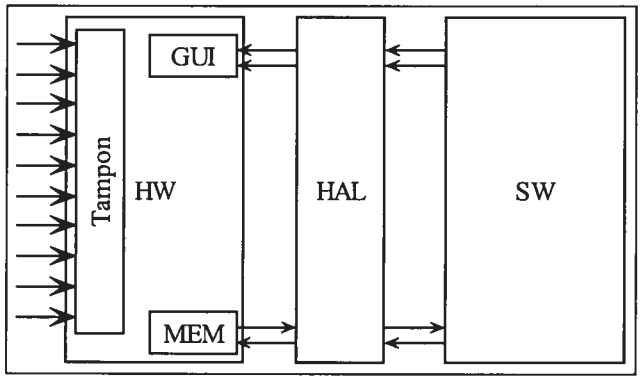


Figure 16: Structure de l'application

À ce niveau, puisqu'un partitionnement matériel et logiciel est introduit, nous avons décidé de modéliser un HAL pour la communication entre le matériel et le logiciel. La couche HAL contient des fonctions permettant de communiquer soit avec le matériel ou le logiciel. Pour la cosimulation avec le matériel, nous employons une mémoire. La mémoire est écrite dans l'environnement de .NET et emploie une structure synchronisée où .NET s'occupe de la synchronisation des accès à la structure; elle peut être perçue comme une mémoire partagée entre le matériel et le logiciel (voir Figure 16). Typiquement le matériel écrit dans la mémoire et envoie une interruption au logiciel pour la lui faire lire. À ce niveau, le matériel et le logiciel partagent des fonctions qu'ils peuvent appeler directement, car ils se trouvent dans le même environnement.

Comme mentionné précédemment, le matériel est modélisé avec eSYS.net. Chaque bouton de l'application est relié à un module eSYS.net. Notre matériel est modélisé comme un clavier où chaque bouton est considéré comme un événement. Un tampon (voir Figure 16) est utilisé pour accumuler les événements envoyés par l'usager. Chaque module est sensible à un événement particulier. À chaque coup d'horloge, l'ordonnanceur de eSYS.net traite un signal et réveille le module approprié.

```
01  private speed;
02  private acc;
03
04  OSTaskCreate(CalculateSpeed, (void *) 22, &CalculateSpeedStk[TASK_STK_SIZE],
    11);
05
06  OSStart();
07
08  CalculateSpeed()
09  {
10          while(1)
11      OSTimeDly(10);
12      ...
13   }
14  // Software Update Speed
15  {
16      hw.mem["speed"] = _speed;
17      Register CallBack
18       ...
19  }
20
21  // Hardware Increment Acceleration
22  {
23      hw.mem["acc"] = acc++;
24      OpenEventA((int)(0x000F0000L|0x00100000L|0x3),0,irqName);
25      SetEvent(handle);
26      ...
27  }
```

Code 13: Exemple partie logicielle au niveau 3

6.3 Niveau 3: Exploration détaillée de système d'exploitation spécifique

Au niveau 3, nous avons utilisé, pour la modélisation de la partie logicielle, le port de µC/OS pour WIN32 [25] qui est intégré dans l'environnement de notre plateforme

(Pour plus de détails sur ce port référez-vous à [25]). Le code logiciel n'a pas beaucoup changé sauf qu'il est maintenant programmé avec le langage C et appelle les fonctions définies par l'API de µC/OS. La fonction *OSTaskCreate()* de µC/OS est appelée maintenant pour créer une tâche. La signature de fonction inclut non seulement une fonction, un numéro de priorité, mais également un pointeur sur les arguments de la fonction et un pointeur sur la pile de la tâche (voir ligne 4 dans Code 13). L'ordonnanceur de µC/OS est lancé pour toutes les tâches avec la fonction *OSStart()* (voir ligne 6 dans Code 13) et les contraintes de temps sont ajoutées en appelant les fonctions spécifiques à µC/OS comme *OSTimeDly()* (voir ligne 11 dans Code 13). Dans un fichier de configuration de µC/OS, il est maintenant possible d'optimiser le système d'exploitation pour choisir les services qui vont être employés ou bien pour déterminer ce qui ne sera pas nécessaire.

Un changement doit être fait dans l'interface entre le matériel et le logiciel. Pour la simplicité de nos expériences, nous avons choisi de garder eSYS.net pour la description du matériel. Cependant, puisque eSYS.net est employé, «PInvoke» et les «CallBack» ont été choisis, car ce sont des techniques rapides et directes d'interopérabilité entre du code géré (eSYS.net) et du code non géré (µC/OS). À ce niveau, grâce au port de µC/OS, le concept d'interruption est introduit et peut être utilisé par le matériel pour communiqué avec le logiciel.

Notre modèle matériel est toujours modélisé avec eSYS.net, il n'y a donc aucune différence entre le niveau 2 et niveau 3 pour la description du matériel.

```
01  private speed;
02  private acc;
03
04  OSTaskCreat(CalculateSpeed, (void *) 22, &CalculateSpeedStk[TASK_STK_SIZE],
    11);
05
06  OSStart();
07
08  CalculateSpeed()
09  {
10    while(1)
11      OSTimeDly(10);
12      ...
```

```
13  }
14
15  // Software Update Speed
16  {
17   write_to_lcd
18   ...
19  }
20
21  // Hardware Increment Acceleration
22  {
23   if (type == 'c')
24    {
25      switch (edge_capture)
26      {
27         case 0x1:
28            acc++;
29            break;
30         }
31         …
32      }
33   }
34  }
```

Code 14: Exemple partie logicielle au prototypage

6.4 Prototypage

Après avoir passé à travers notre plateforme pour le raffinement, nous avons jugé bon de tester la fonctionnalité de notre application sur une plateforme de prototypage. Pour cet exercice, nous avons choisi le kit de développement NIOS. Cette plateforme, offerte par Altera [26], offre les outils nécessaires pour le développement d'un système embarqué complet. Ce dernier vient avec un port de µC/OS pour NIOS. µC/OS a été complètement porté par Altera sur l'architecture NIOS sans faire de compromis et en suivant les procédures standards [44].

Nios est le processeur logiciel d'Altera utilisé sur leur plateforme de développement. Ce processeur nous est intéressant, car c'est celui que nous utilisons pour le prototypage de ce projet de recherche. Altera nous offre tous les outils pour faire le développement de systèmes embarqués.

Dans la Figure 17, l'environnement pour la conception d'un logiciel embarqué avec les outils d'Altera est présenté.

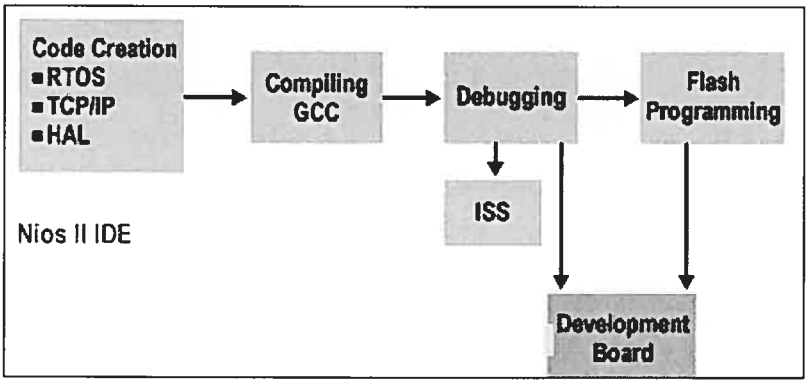


Figure 17: Structure de l'outil Nios II IDE [26]

Le Nios II IDE est une application logicielle graphique pour le processeur logiciel Nios. Il nous permet d'éditer, compiler et déboguer les applications développées avec cet environnement. Le Nios IDE comprend une chaîne d'outil GCC, une couche d'abstraction de matériel (HAL), des bibliothèques de systèmes temps réel, une pile TCP/IP et un simulateur d'ensemble d'instructions.

Cet environnement nous permet de développer une application embarquée, nous fournit les bibliothèques comme celle de μC/OS et nous permet de faire interagir notre système avec du matériel en VHDL. Ensuite, il nous suffit de le compiler et de le télécharger sur le FPGA à travers la mémoire flash pour tester notre application.

Le code de la partie logicielle du niveau 3 a été copié aux outils appropriés d'Altera qui compilent le code pour l'architecture NIOS. Le code reste presque identique au niveau 3 (voir Code 13) excepté pour l'interaction entre le matériel et le logiciel où des bibliothèques offertes par Altera ont été employées. Le concept reste fondamentalement le même, avec le logiciel qui capture une interruption du matériel. Également plutôt que d'écrire les résultats au GUI, le logiciel envoie les résultats directement à l'écran d'affichage à cristaux liquides sur la plateforme FPGA. Les entrées et sorties offertes sur la plateforme sont employées pour la description matérielle du système.

EXPÉRIMENTATION

Des résultats expérimentaux sont obtenus en utilisant un banc d'essai composé d'un ensemble d'événements (c.-à-d., EngineOn, EngineOff, Accelerate, Decelerate...) envoyés à l'application décrite dans le chapitre 4. La simulation du modèle entier a été faite sur un ordinateur portable ayant comme CPU un Pentium M (1.8GHz) et roulant sous Windows XP.

7.1 Résultats sur temps d'exécution et Taille

	Niveau 1	Niveau 2	Niveau 3
Temps d'exécution total	20.78s	12.94s	15.85s
Temps d'exécution de la fonction CalculateSpeed	0.002s	0.003s	N/A
Taille	48KB	160KB	192KB

Tableau 3: Temps d'exécution et taille

Certains résultats du Tableau 1 ont été obtenus avec la commande *time* de l'environnement Cygwin. Les résultats du Tableau 1 représentent la taille de chaque exécutable, le temps d'exécution total et le temps d'exécution d'une fonction spécifique appelée *CalculateSpeed(),* qui est la fonction la plus représentative dans la partie du logiciel.

7.1.1 Temps d'exécution total

Les résultats obtenus sur le temps d'exécution total nous démontrent que le niveau 1 est le plus lent. Selon l'implémentation, ces résultats peuvent varier, mais dans la

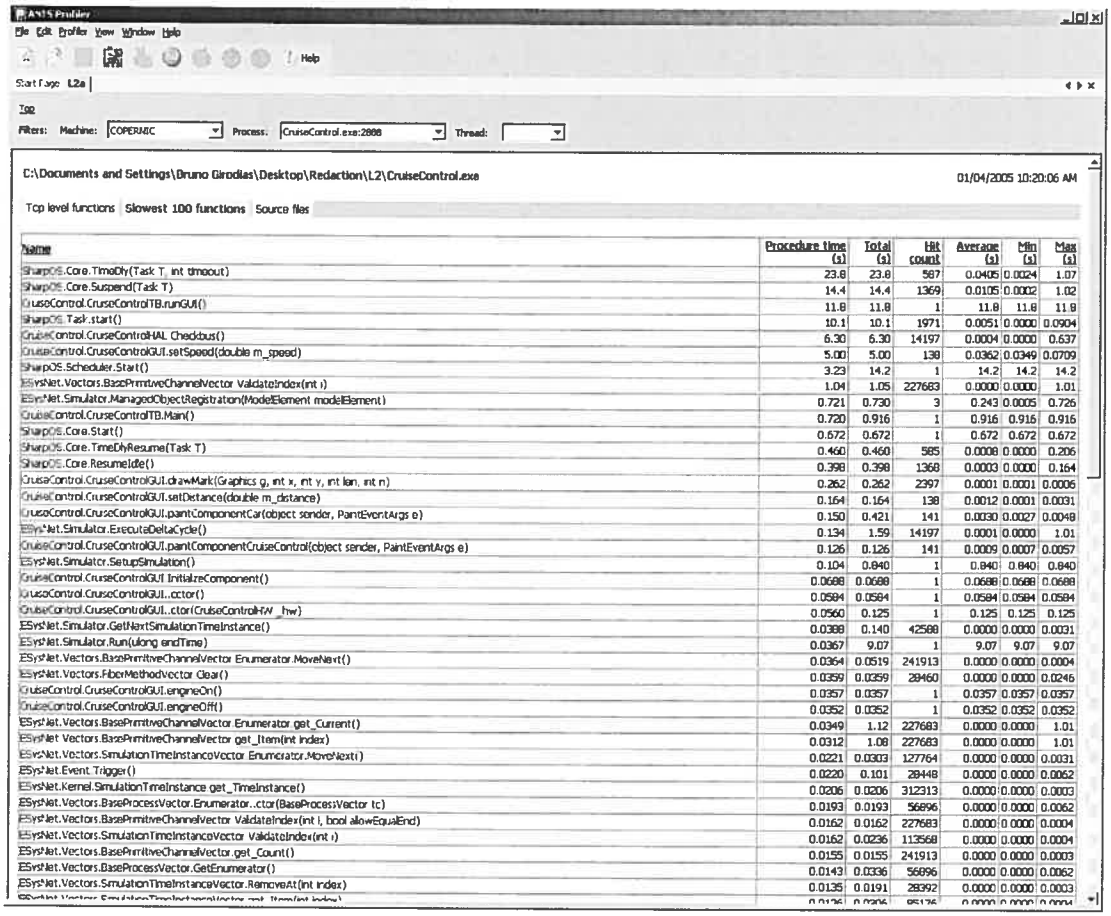
plupart des cas, le temps d'exécution totale du niveau sera plus lent puisque le concepteur devra employer de nombreux fils d'exécution pour modéliser le système. La grande différence dans le temps exécution entre les niveaux 2 et 3 se trouve dans le mécanisme de communication entre le matériel et le logiciel. Au niveau 2, des appels de fonctions simples sont employés pour communiquer au lieu des longues techniques d'interopérabilité au niveau 3.

7.1.2 Temps d'exécution de la fonction CalculateSpeed

Les résultats obtenus sur le temps d'exécution de la fonction de *CalculateSpeed()* montrent non seulement son temps exécution, mais également que ces tests de performance ne peuvent pas être obtenus au niveau 3 (indiqué par N/A dans Tableau 3 et Tableau 4). Ceci est dû au fait qu'on ne peut pas faire de l'introspection en C comme dans C#. Ceci ne pose aucun problème puisqu'au niveau 2 toutes les erreurs de fonctionnalité ou d'exécution sont détectées et quelques corrections et optimisations peuvent être faites. Les résultats pour le niveau 1 sont meilleurs que ceux du niveau 2 puisqu'il n'y a aucune communication entre le matériel et le logiciel.

7.1.3 Taille totale

Comme prévu, le niveau 1 est le plus petit en taille. Dans le niveau 1 le modèle est simple et n'inclut pas d'architecture. Le modèle est seulement compilé avec les APIs .NET nécessaires. On peut être étonné de voir que la taille du niveau 2 est plus petite que le niveau 3. Ce résultat est expliqué par le fait que la bibliothèque SharpOS est seulement un sous-ensemble des possibilités de µC/OS.



Name	Procedure time (s)	Total (s)	Hit count	Average (s)	Min (s)	Max (s)
SharpOS.Core.TimeDly(Task T, int timeout)	23.8	23.8	587	0.0405	0.0024	1.07
SharpOS.Core.Suspend(Task T)	14.4	14.4	1369	0.0105	0.0002	1.02
CruiseControl.CruiseControlTB.runGUI()	11.8	11.8	1	11.8	11.8	11.8
SharpOS.Task.start()	10.1	10.1	1971	0.0051	0.0000	0.0904
CruiseControl.CruiseControlHAL.Checkbus()	6.30	6.30	14197	0.0004	0.0000	0.637
CruiseControl.CruiseControlGUI.setSpeed(double m_speed)	5.00	5.00	138	0.0362	0.0349	0.0709
SharpOS.Scheduler.Start()	3.23	14.2	1	14.2	14.2	14.2
ESysNet.Vectors.BasePrmitiveChannelVector.ValidateIndex(int i)	1.04	1.05	227683	0.0000	0.0000	1.01
ESysNet.Simulator.ManagedObjectRegistration(ModelElement modelElement)	0.721	0.730	3	0.243	0.0005	0.726
CruiseControl.CruiseControlTB.Main()	0.720	0.916	1	0.916	0.916	0.916
SharpOS.Core.Start()	0.672	0.672	1	0.672	0.672	0.672
SharpOS.Core.TimeDlyResume(Task T)	0.460	0.460	585	0.0008	0.0000	0.206
SharpOS.Core.ResumeIdle()	0.398	0.398	1368	0.0003	0.0000	0.164
CruiseControl.CruiseControlGUI.drawMark(Graphics g, int x, int y, int len, int n)	0.262	0.262	2397	0.0001	0.0001	0.0006
CruiseControl.CruiseControlGUI.setDistance(double m_distance)	0.164	0.164	138	0.0012	0.0001	0.0031
CruiseControl.CruiseControlGUI.paintComponentCar(object sender, PaintEventArgs e)	0.150	0.421	141	0.0030	0.0027	0.0048
ESysNet.Simulator.ExecuteDeltaCycle()	0.134	1.59	14197	0.0001	0.0000	1.01
CruiseControl.CruiseControlGUI.paintComponentCruiseControl(object sender, PaintEventArgs e)	0.126	0.126	141	0.0009	0.0007	0.0057
ESysNet.Simulator.SetupSimulation()	0.104	0.840	1	0.840	0.840	0.840
CruiseControl.CruiseControlGUI.InitializeComponent()	0.0688	0.0688	1	0.0688	0.0688	0.0688
CruiseControl.CruiseControlGUI..cctor()	0.0584	0.0584	1	0.0584	0.0584	0.0584
CruiseControl.CruiseControlGUI..ctor(CruiseControlHW _hw)	0.0560	0.125	1	0.125	0.125	0.125
ESysNet.Simulator.GetNextSimulationTimeInstance()	0.0388	0.140	42588	0.0000	0.0000	0.0031
ESysNet.Simulator.Run(ulong endTime)	0.0367	9.07	1	9.07	9.07	9.07
ESysNet.Vectors.BasePrmitiveChannelVector.Enumerator.MoveNext()	0.0364	0.0519	241913	0.0000	0.0000	0.0004
ESysNet.Vectors.FiberMethodVector.Clear()	0.0359	0.0359	28460	0.0000	0.0000	0.0246
CruiseControl.CruiseControlGUI.engineOn()	0.0357	0.0357	1	0.0357	0.0357	0.0357
CruiseControl.CruiseControlGUI.engineOff()	0.0352	0.0352	1	0.0352	0.0352	0.0352
ESysNet.Vectors.BasePrmitiveChannelVector.Enumerator.get_Current()	0.0349	1.12	227683	0.0000	0.0000	1.01
ESysNet.Vectors.BasePrmitiveChannelVector.get_Item(int index)	0.0312	1.08	227683	0.0000	0.0000	1.01
ESysNet.Vectors.SimulationTimeInstanceVector.Enumerator.MoveNext()	0.0221	0.0303	127764	0.0000	0.0000	0.0031
ESysNet.Event.Trigger()	0.0220	0.101	28448	0.0000	0.0000	0.0062
ESysNet.Kernel.SimulationTimeInstance.get_TimeInstance()	0.0206	0.0206	312313	0.0000	0.0000	0.0003
ESysNet.Vectors.BaseProcessVector.Enumerator..ctor(BaseProcessVector tc)	0.0193	0.0193	56896	0.0000	0.0000	0.0062
ESysNet.Vectors.BasePrmitiveChannelVector.ValidateIndex(int i, bool allowEqualEnd)	0.0162	0.0162	227683	0.0000	0.0000	0.0004
ESysNet.Vectors.SimulationTimeInstanceVector.ValidateIndex(int i)	0.0162	0.0236	113568	0.0000	0.0000	0.0004
ESysNet.Vectors.BasePrmitiveChannelVector.get_Count()	0.0155	0.0155	241913	0.0000	0.0000	0.0003
ESysNet.Vectors.BaseProcessVector.GetEnumerator()	0.0143	0.0336	56896	0.0000	0.0000	0.0062
ESysNet.Vectors.SimulationTimeInstanceVector.RemoveAt(int index)	0.0135	0.0191	28392	0.0000	0.0000	0.0003

Figure 18: Capture d'écran de l'outil ANTS

7.2 Résultats de profilage

	Niveau 1	Niveau 2	Niveau 3
GUI	54.70s	5.00s	1.09s
SW	116.57s	39.20s	N/A
HW	-	14.75s	9.22s
OS	-	89.32s	N/A
SDSL	-	15.35s	14.7s
.NET	44.48s	0	0

Tableau 4: Résultats de profilage

Les résultats dans le Tableau 4 ont été obtenus en utilisant le logiciel ANTS [38] qui est un outil de profilage puissant pour les applications de .NET. Si les résultats des Tableau 3 et du Tableau 4 sont comparés, on peut noter que le temps d'exécution est augmenté approximativement d'un facteur de dix. Ceci est dû au fait que ANTS ajoute quelques en-têtes à l'application pour pouvoir faire le profilage. Le profileur ANTS est capable d'extraire toute l'information dans une application, au sujet des 100 fonctions les plus lentes (voir Figure 18). Ces fonctions ont été prises et regroupées dans différentes catégories représentant leur fonctionnalité. Les catégories sont: GUI, représentant des fonctions liées à l'interface graphique de l'utilisateur; SW, représentant des fonctions liées au côté logiciel de l'application embarquée; HW, concernant l'aspect de matériel de l'application incluse; OS, pour les fonctions internes du logiciel d'exploitation; SDSL pour les fonctions internes du SDSL et du .NET pour les fonctions internes du cadre d'applications .NET.

En utilisant le Tableau 4, un diagramme a été crée pour comparer les composantes à différents niveaux. Pour le niveau 1 sur la Figure 19, on peut voir que le temps de simulation est partagé entre le GUI, le SW et le .NET. Au niveau 1, le système embarqué est décrit complètement et seulement en logiciel. C'est pour cette raison que les composantes de SW et de GUI prennent la majeur partie du temps de simulation. De plus, puisque la bibliothèque de .NET est fréquemment employée, des résultats d'exécution de quelques fonctions internes de cette bibliothèque peuvent être observés.

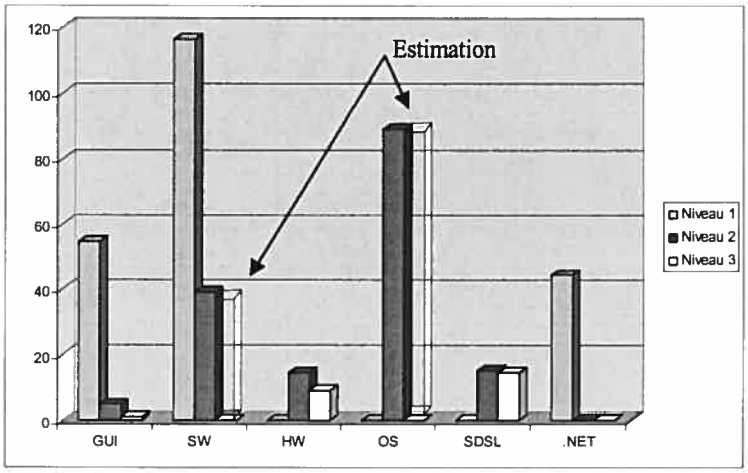


Figure 19: Graphique des résultats de profilage

Dans le niveau 2, quelques nouvelles composantes apparaissent et produisent de l'information utile qui ne peut pas être observée à d'autres niveaux. Ce niveau permet de chercher l'information sur les fonctions internes du logiciel d'exploitation utilisé dans le système embarqué. Ceci signifie que les développeurs de systèmes embarqués peuvent choisir d'établir leur propre OS et obtenir de l'information sur le temps d'exécution. Avec l'outil de profilage, il n'est pas possible de déterminer le temps d'exécution de Micro/OS au niveau 3. Les résultats du niveau 2 donnent une bonne estimation sur le pire cas d'exécution de l'OS puisqu'au niveau 3 le système d'exploitation est optimisé pour l'application. L'information d'exécution sur la fonction interne du SDSL peut être obtenue à chaque niveau puisque eSYS.net est employé et est basé sur .NET.

Comme mentionné précédemment, on ne peut pas déterminer de résultats d'exécution pour les composantes de SW et d'OS au troisième niveau. Ceci est dû au fait qu'au niveau 3 le SW et l'OS sont écrits en C qui ne peut pas être introspectable. Il manque, entre autre, dans le profilage les résultats d'exécution du mécanisme d'interopérabilité employé pour communiquer le logiciel et le matériel. À partir de l'évaluation d'exécution du niveau 2, il est possible d'avoir une bonne idée sur ce qu'il faut prévoir au niveau 3.

Cette expérimentation nous a non seulement donné des résultats d'exécution, mais nous a aussi démontré la puissance d'employer le cadre d'applications .NET comme structure de base de la plateforme pour le raffinement. Du fait même, beaucoup d'autres outils existants peuvent être utilisés pour aider à corriger et analyser le système.

CONCLUSIONS ET PERSPECTIVES

8.1 Conclusions

Dans ce mémoire, différentes plateformes ont été brièvement présentées avec une introduction à quelques approches pour la simulation de systèmes embarqués. Plusieurs restreignent l'exploration du développeur et d'autres apportent des techniques intéressantes pour la conception d'un système embarqué.

Cette recherche a porté une attention plus particulière sur la simulation de composantes logicielles avec un système d'exploitation d'un système embarqué. Il a été noté que, pour simuler, le développeur devait trouver un environnement pour la simulation native, apprendre les différents APIs de chaque système d'exploitation et trouver ou créer un port pour le système d'exploitation sur l'architecture de la cible sélectionnée.

Actuellement, l'abstraction d'OS représente une des principales caractéristiques manquantes dans les outils de développement et, si présente, satisferait un besoin grandissant dans l'industrie. Ce dispositif permettrait à des concepteurs de décrire et valider leur application avec plusieurs OS à différents niveaux d'abstraction et, par conséquence, explorer plusieurs services d'OS requis dans l'application. De la même manière, on peut explorer les divers OS sans devoir nécessairement apprendre comment programmer avec certains APIs.

Cette recherche a proposé une plateforme pour aider au raffinement des systèmes embarqués dans un environnement standard comme le cadre d'applications .NET. Plusieurs bibliothèques sont ajoutées et intégrées pour offrir aux développeurs la

possibilité de simuler la partie matérielle et la partie logicielle de leurs systèmes embarqués. Cet article se concentre sur l'aspect logiciel où la fonctionnalité est examinée avec des bibliothèques qui simulent le comportement d'un système d'exploitation ou quelques services spécifiques de système d'exploitation.

Cette plateforme a présenté trois niveaux d'abstraction. Ces niveaux permettent d'explorer l'application, l'exploration de différents services et systèmes d'exploitation et l'exploration détaillée d'un système d'exploitation sélectionné. La plateforme permet le profilage de l'évaluation de ressources pour les prochains niveaux. Tous ces dispositifs sont dans un environnement multilangage qui offre des mécanismes d'interopérabilité qui peuvent être employés pour la cosimulation.

Travailler avec le cadre d'applications .NET nous offre un environnement standard qui est portatif à d'autres plateformes autres que Windows. La combinaison de l'IDE éclipse [28] et du projet mono [34] nous permettrait de travailler sur d'autres plateformes telles que Linux, mais aussi d'interagir avec des ISS et des pluggins déjà intégrés dans cet IDE. L'introspection est un excellent dispositif pour le développement d'outils supplémentaires pour notre plateforme. Des outils existants, comme le profileur ANTS peuvent déjà être utilisés pour analyser cet environnement et avec l'annonce par Microsoft du produit Indigo, d'autres possibilités s'ouvrent à nous.

8.2 Perspectives

Il serait intéressant de compléter le modèle de la bibliothèque SharpOS et d'enrichir la plateforme avec d'autres systèmes d'exploitation ou par des bibliothèques standard de services de systèmes d'exploitation comme ITRON [31]. Avec l'aide du mécanisme d'annotation du cadre d'applications, un dispositif pour ajouter des contraintes de temps peut être ajouté. Ce mécanisme permettrait d'améliorer la conception du temps dans notre plateforme, un sujet qui n'a pas été abordé dans cette recherche. Ensuite, une attention pourrait être portée sur les mécanismes d'interopérabilité pour une cosimulation précise avec des modèles

matériels décrits soit avec VHDL, soit avec des SDSL ou même avec des outils comme ModelSim.

La création d'une spécification unique pour le premier niveau d'abstraction permettrait un partitionnement semi-automatique et faciliterait la génération automatique de HAL et l'automation du raffinement à chaque niveau. Des recherches prometteuses permettent d'envisager que le code intermédiaire soit porté et compilé vers d'autres architectures et processeurs. De même, avec la sortie du cadre d'applications .NET pour systèmes embarqués, on peut envisager dans le futur la notion de temps réel dans notre plateforme pour le raffinement.

BIBLIOGRAPHIE

[1] Bacivarov I., et al., *Time HW-SW Cosimulation Using Native Execution of OS and Applications SW*, Proceedings of the HLDVT'02, 2002

[2] Bhasker J., *A SystemC Primer*, Prentice Hall PTR, 2002

[3] Bhasker J., *A VHDL Primer*, Prentice Hall PTR, 2002

[4] Bradley M., Xie K., *Hardware/Software Co-Verification with RTOS Application Code*, White Paper from Mentor Graphics

[5] Chevalier J., et al., *SPACE: A Hardware/Software SystemC Modeling Platform Including an RTOS*, NASCUG, 2004

[6] David E. Simon, *An Embedded Software Primer*, Addison-Wesley, 2001

[7] David J-P., *Cours IFT6223*, DIRO, Université de Montréal, http://www.iro.umontreal.ca/~pift6223

[8] David J-P., *Architecture synchronisée par les données pour système re-configurable*, thèse de doctorat, Faculté des Sciences Appliquées, Université Catholique de Louvain, 2002

[9] Decotigny D., *Croisière au Cœur d'un OS*, http://sos.enix.org, 2004

[10] Drayton P., Alabahari B., Neward T., *C# in a Nutshell*, O'Reilly, 2003

[11] Grotker T., *Modeling software with SystemC 3.0*, http://www-ti.informatik.unituebingen.de/~systemc/Documents/Presentation-6-OSCI5_groetker.pdf, 2002

[12] Honda S., Wakabayashi T., Tomiyama H., Takada H., *RTOS-Centric Hardware/Software Cosimulator for Embedded System Design*, CODES + ISSS 2004

[13] Kadionik P., *Le noyau temps réel uC/OS*, http://www.enseirb.fr, 2004

[14] Lapalme J., et al., *.NET Framework – A solution for the next generation tools for system-level design?*, Automation and Test (DATE04), 2004

[15] Labrosse Jean J., *MicroC/OS-II The Real-Time Kernel*, CMPBooks, 2002

[16] Liberty J., *Programming C#*, O'Reilly 2003.

[17] Lions J. L., *Ariane 5 Flight 501 Failure and About the Mars Pathfinder*, http://vlsi.colorado.edu/~abel/pubs/anecdote.html, 2004

[18] Lutz M. H., Laplante P. A., *C# and the .NET framework: ready for real time?*, IEEE Software, vol. 20, pp. 74 -80, 2003

[19] Nicolescu G., *Spécification et validation des systèmes hétérogènes embarqués*, thèse de doctorat, TIMA, Institut National Polytechnique de Grenoble, 2002

[20] Pech P., *Simulation of uC/OS-II on Win32 platform*, http://wsim.pc.cz/abstract.php

[21] Troelsen A., *COM and .NET Interoperability*, Apress, 2002

[22] Volkan A., Nienaltowski P., *SCOOPLI: a library for concurrent object-oriented programming on .NET*, http://se.inf.ethz.ch/people/arslan/data/scoop/workshops/Plzen_2003/SCOOPLI_on_.NET.pdf

[23] Yoo S., et al., *Automatic Generation Including Fast Timed Simulation Models of Operating Systems in Multiprocessor SoC Communication Design*, DATE 2002

[24] Zerarka M. T., *Accélération de prédiction génétique par implémentation hautement parallèle sur un matériel re-configurable*, thèse de maîtrise, DIRO, Université de Montréal, 2004

[25] Zimmermann W., *uC/OS–II WIN32 32bit Windows Operating Systems V1.30*, http://www.it.fht-esslingen.de/~zimmerma/software/uCOS-II_WIN32.htm, 2004

[26] "Altera", http://www.altera.com/, 2005

[27] "Comparison of VHDL, Verilog and System Verilog", http://www.bitpipe.com

[28] "Eclipse", http://www.eclipse.org/, 2005

[29] "ECMA and ISO/IEC C# and Common Language Infrastructure Standards", http://msdn.microsoft.com/net/ecma/, 2005

[30] "eCos", http://sourceware.org/ecos/

[31] "ITRON", http://www.assoc.tron.org/eng/document.html

[32] "MicroC/OS-II", http://www.micrium.com/, 2005

[33] "Migrating Java Applications To The .NET Framework", http://www.financialdevelopers.com/assets/presentations/Infusion%20Development%20%20Migrating%20Java%20Applications.ppt

[34] "Mono", http://www.go-mono.com/, 2005

[35] ".NET Framework", http://www.microsoft.com/net, 2003

[36] "Nucleus", http://www.acceleratedtechnology.com/, 2005

[37] "ProActive", http://www-sop.inria.fr/oasis/ProActive/

[38] "Red-gates ANTS", http://www.red-gates.com/, 2005

[39] "SystemC", http://www.systemc.org/, 2003

[40] "WinDriver", http://www.windriver.com/, 2005

[41] "Carbonkernel", http://www.carbonkernel.org/, 2005

[42] Rowson J., *Hardware/Software Cosimulation co-simulation*, Proc. Design Automation Conference (DAC), 1994

[43] Zivojnovic V., Meyr H., *Compiled Hw/Sw Cosimulation*, Proc. Design Automation Conference (DAC), 1996

[44] "Altera Nios uC/OS-II ", http://www.altera.com/literature/hb/nios2/n2sw_nii52008.pdf , 2005

[45] Gauthier L., *OS generation for multitask software targeting on heterogeneous multiprocessor architectures for specific embedded systems*, Thèse de Doctorat INPG, 2001

ANNEXES A

Partie du code de chaque niveau

Niveau 1

```
01  package CruiseControl;
02
03  public class Interface
04  {
05
06   final static int ACTIVE = 1;
07
08   final static int INACTIVE = 0;
09
10   final static int CRUISING = 2;
11
12   int state = INACTIVE;
13
14   double desiredSpeed = 0;
15
16   double acc = 0;
17
18   double brake = 0;
19
20   double alpha = 0;
21
22   CarModel activeSpeed = null;
23
24   CruiseControl activeCruise = null;
25   int sync =0;
26
27   public CruiseControlForm form;
28   public System.Threading.Thread T1;
29   public System.Threading.Thread T2;
30   public System.Threading.Thread T3;
31   public System.Threading.Thread L1;
32   public System.Threading.Thread L2;
33   public System.Threading.Thread L3;
34   public System.Collections.Queue queue =
    System.Collections.Queue.Synchronized(new System.Collections.Queue());
35
36   public System.Threading.Thread TB;
```

```
37
38   public Interface(){}
39
40   public Interface(CruiseControlForm m_form)
41   {
42      this.form = m_form;
43   }
44
45   public void initialize()
46   {
47      activeSpeed = new CarModel(this);
48      activeCruise = new CruiseControl(this);
49      T1 = new System.Threading.Thread(new
     System.Threading.ThreadStart(activeSpeed.run));
50      T2 = new System.Threading.Thread(new
     System.Threading.ThreadStart(activeCruise.run));
51      TB = new System.Threading.Thread(new
     System.Threading.ThreadStart(this.testbench));
52
53
54      T1.set_Priority(System.Threading.ThreadPriority.AboveNormal);
55      T2.set_Priority(System.Threading.ThreadPriority.AboveNormal);
56      T1.Start();
57      T2.Start();
58
59      TB.set_Priority(System.Threading.ThreadPriority.Lowest);
60      System.Threading.Thread.Sleep(100);
61      TB.Start();
62
63
64
65   }
66
67   public void engineOn()
68   {
69      if (this.state == INACTIVE)
70      {
71         this.state = ACTIVE;
72         activeSpeed.engineOn();
73         form.CarPanel.engineOn();
74
75      }
76   }
77
78
79
80   public void engineOff()
81   {
82
83         activeSpeed.engineOff();
84         activeCruise.engineOff();
85
86         this.state = INACTIVE;
87         this.acc = 0;
88         this.desiredSpeed = 0;
```

```
89          setAcceleration(0);
90          form.engineOff();
91
92    }
93    public void controlOn()
94    {
95       synchronized(this)
96       {
97          if (this.state == ACTIVE)
98          {
99             this.form.ControlPanel.controlOn();
100            this.state = CRUISING;
101            desiredSpeed = getSpeed().doubleValue();
102            activeCruise.controlOn(desiredSpeed, this.acc);
103
104         }
105      }
106
107   }
108
109   public void controlOff()
110   {
111      synchronized(this)
112      {
113         if (this.state == CRUISING)
114         {
115            this.state = ACTIVE;
116            acc = 0;
117            setAcceleration(0);
118            activeCruise.controlOff();
119
120         }
121      }
122   }
123
124   public void deactiveControl()
125   {
126      synchronized(this)
127      {
128         if (this.state == CRUISING)
129         {
130            this.state = ACTIVE;
131            acc = 0;
132            activeSpeed.brake();
133            form.ControlPanelOff();
134
135         }
136      }
137   }
138
139   public void threaded_accelerate()
140   {
141      T3 = new System.Threading.Thread(new
      System.Threading.ThreadStart(this.accelerate));
142      T3.set_Priority(System.Threading.ThreadPriority.Lowest);
```

```
143     T3.Start();
144     this.queue.Enqueue(T3);
145     L2 = L1;
146     L1 = T3;
147  }
148
149  public void accelerate()
150  {
151     if (L2 != null)
152     {
153        L2.Join();
154     }
155     synchronized(this)
156     {
157        try
158        {
159           Thread.sleep(1000);
160           this.form.MessageTextBox.AppendText("accelerate" +
    System.Environment.get_NewLine());
161           if (this.state == ACTIVE)
162           {
163              if (this.acc < 50)
164              {
165                 this.acc += 1;
166                 activeSpeed.setAcceleration(this.acc);
167                 form.setAcceleration(this.acc);
168              }
169           }
170
171        }
172        catch(InterruptedException e){}
173        L3= (System.Threading.Thread)this.queue.Dequeue();
174     }
175  }
176
177  public void threaded_decelerate()
178  {
179     T3 = new System.Threading.Thread(new
    System.Threading.ThreadStart(this.decelerate));
180     T3.set_Priority(System.Threading.ThreadPriority.Lowest);
181     T3.Start();
182     this.queue.Enqueue(T3);
183     L2 = L1;
184     L1 = T3;
185  }
186
187  public void decelerate()
188  {
189     if (L2 != null)
190     {
191        L2.Join();
192     }
193     synchronized(this)
194     {
195        try
```

```
196         {
197
198             Thread.sleep(0);
199             this.form.MessageTextBox.AppendText("decelerate" +
    System.Environment.get_NewLine());
200             if (this.state == ACTIVE)
201             {
202                 if (this.acc > 0)
203                 {
204                     this.acc -= 1;
205                     activeSpeed.setAcceleration(this.acc);
206                     form.setAcceleration(this.acc);
207                 }
208             }
209
210         }
211         catch(InterruptedException e){}
212         if(this.queue.get_Count() != 0)
213         {
214             L3= (System.Threading.Thread)this.queue.Dequeue();
215         }
216
217     }
218 }
219
220 public void threaded_accelerateCruise()
221 {
222     T3 = new System.Threading.Thread(new
    System.Threading.ThreadStart(this.accelerateCruise));
223     T3.set_Priority(System.Threading.ThreadPriority.Lowest);
224     T3.Start();
225     this.queue.Enqueue(T3);
226     L2 = L1;
227     L1 = T3;
228 }
229
230 public void accelerateCruise()
231 {
232     if (L2 != null)
233     {
234         L2.Join();
235     }
236     synchronized(this)
237     {
238
239         try
240         {
241             Thread.sleep(0);
242             this.form.MessageTextBox.AppendText("accelerateCruise" +
    System.Environment.get_NewLine());
243             if (this.state == CRUISING)
244             {
245                 desiredSpeed += 1;
246                 activeCruise.setDesiredSpeed(desiredSpeed);
247                 form.setDesiredSpeed(desiredSpeed);
```

```
248
249             }
250
251         }
252         catch(InterruptedException e){}
253         L3= (System.Threading.Thread)this.queue.Dequeue();
254     }
255 }
256
257 public void threaded_decelerateCruise()
258 {
259     T3 = new System.Threading.Thread(new
    System.Threading.ThreadStart(this.decelerateCruise));
260     T3.set_Priority(System.Threading.ThreadPriority.Lowest);
261     T3.Start();
262     this.queue.Enqueue(T3);
263     L2 = L1;
264     L1 = T3;
265 }
266
267 public void decelerateCruise()
268 {
269     if (L2 != null)
270     {
271         L2.Join();
272     }
273     synchronized(this)
274     {
275         try
276         {
277
278             Thread.sleep(0);
279             this.form.MessageTextBox.AppendText("decelerateCruise" +
    System.Environment.get_NewLine());
280             if (this.state == CRUISING)
281             {
282                 desiredSpeed -= 1;
283                 activeCruise.setDesiredSpeed(desiredSpeed);
284                 form.setDesiredSpeed(desiredSpeed);
285
286             }
287
288         }
289         catch(InterruptedException e){}
290         L3= (System.Threading.Thread)this.queue.Dequeue();
291     }
292
293 }
294
295 public void threaded_incAplha()
296 {
297     T3 = new System.Threading.Thread(new
    System.Threading.ThreadStart(this.incAlpha));
298     T3.set_Priority(System.Threading.ThreadPriority.Normal);
299     T3.Start();
```

```
300  }
301
302  public void incAlpha()
303  {
304     synchronized(this)
305     {
306        try
307        {
308
309           Thread.sleep(0);
310           this.form.MessageTextBox.AppendText("incAlpha" +
     System.Environment.get_NewLine());
311           if ((this.state == ACTIVE) || (this.state == CRUISING))
312           {
313              if (alpha < 0.6)
314                 alpha += 0.09;
315              else
316                 alpha = 0.6;
317              setAlpha(alpha);
318           }
319        }
320        catch(InterruptedException e){}
321     }
322  }
323
324  public void threaded_decAlpha()
325  {
326     T3 = new System.Threading.Thread(new
     System.Threading.ThreadStart(this.decAlpha));
327     T3.set_Priority(System.Threading.ThreadPriority.Normal);
328     T3.Start();
329  }
330
331  public void decAlpha()
332  {
333     synchronized(this)
334     {
335        try
336        {
337
338           Thread.sleep(0);
339           this.form.MessageTextBox.AppendText("decAlpha" +
     System.Environment.get_NewLine());
340           if ((this.state == ACTIVE) || (this.state == CRUISING))
341           {
342              if (alpha > -0.6)
343                 alpha -= 0.09;
344              else
345                 alpha = -0.6;
346              setAlpha(alpha);
347
348           }
349           sync =0;
350           System.Threading.Monitor.Pulse(this);
351           System.Threading.Monitor.Exit(this);
```

```
352         }
353         catch(InterruptedException e){}
354      }
355   }
356
357   public void threaded_brake()
358   {
359      T3 = new System.Threading.Thread(new
      System.Threading.ThreadStart(this.brake));
360      T3.set_Priority(System.Threading.ThreadPriority.Highest);
361      T3.Start();
362
363   }
364
365   public void brake()
366   {
367      this.form.MessageTextBox.AppendText("brake" +
      System.Environment.get_NewLine() );
368      if ((this.state == ACTIVE) || (this.state == CRUISING))
369      {
370         activeCruise.controlOff();
371         while (this.queue.get_Count() != 0)
372         {
373            L3=(System.Threading.Thread) this.queue.Dequeue();
374            L3.Abort();
375         }
376         activeSpeed.brake();
377         form.brake();
378         acc = 0;
379         this.state = ACTIVE;
380      }
381   }
382
383    public void setSpeed(double speed) {
384    form.setSpeed(speed);
385    }
386
387    public void setDistance(double distance) {
388    form.setDistance(distance);
389    }
390
391    public void setDesiredSpeed(double m_speed) {
392     synchronized(this)
393     {
394        form.setDesiredSpeed(m_speed);
395     }
396    }
397
398   public Double getSpeed()
399   {
400      return activeSpeed.getSpeed();
401   }
402
403    public void setAcceleration(double m_acc){
404
```

```
405     this.acc = m_acc;
406     activeSpeed.setAcceleration(m_acc);
407     form.setAcceleration(m_acc);
408
409   }
410
411   public Double getAcceleration() {
412     return new Double(this.acc);
413   }
414
415   public void setAlpha(double m_alpha) {
416     if ((this.state == ACTIVE) || (this.state == CRUISING)) {
417       activeSpeed.setAlpha(m_alpha);
418       form.setAlpha(m_alpha);
419     }
420   }
421 }
422
423
424 package CruiseControl;
425
426 public class CruiseControl{
427
428
429   final static int ACTIVE = 1;
430   final static int INACTIVE = 0;
431   int state = INACTIVE;
432   double desiredSpeed = 0;
433   double acc = 0;
434   private Interface father;
435
436   public CruiseControl() {
437
438   }
439
440   public CruiseControl(Interface m_father) {
441     father = m_father;
442   }
443
444   public void controlOn(double m_desiredSpeed, double m_acc) {
445
446       if (this.state == INACTIVE)
447       {
448          desiredSpeed = m_desiredSpeed;
449          this.acc = m_acc;
450          father.setDesiredSpeed(desiredSpeed);
451          this.state = ACTIVE;
452       }
453
454   }
455
456   public void controlOff() {
457
458       if (state == ACTIVE)
459       {
```

```
460             state = INACTIVE;
461             desiredSpeed = 0;
462             father.setDesiredSpeed(0);
463             father.brake();
464         }
465
466     }
467
468     public void engineOff() {
469      synchronized(this)
470      {
471         state = INACTIVE;
472         desiredSpeed = 0;
473         acc = 0;
474         father.setDesiredSpeed(0);
475      }
476     }
477
478     public double getDesiredSpeed() {
479
480         return desiredSpeed;
481
482     }
483
484     public void setDesiredSpeed(double m_speed) {
485
486         if (this.state == ACTIVE)
487             this.desiredSpeed = m_speed;
488
489     }
490
491     public void accelerate() {
492
493         if (this.state == ACTIVE)
494         {
495             this.desiredSpeed += 1;
496             father.setDesiredSpeed(this.desiredSpeed);
497         }
498
499
500     }
501
502     public void decelerate() {
503
504         if (this.state == ACTIVE)
505         {
506             this.desiredSpeed -= 1;
507             father.setDesiredSpeed(this.desiredSpeed);
508         }
509
510     }
511
512     public void calculateAcceleration() {
513      synchronized(this)
514      {
```

```
515       double currentSpeed;
516       currentSpeed = father.getSpeed().doubleValue();
517       double delta = Math.abs(currentSpeed - desiredSpeed);
518       if (this.state == ACTIVE)
519       {
520          if (delta < 0.4)
521             ;
522          else if (currentSpeed < desiredSpeed)
523          {
524             if (delta < 2)
525                this.acc += 0.5;
526             else if (delta < 8)
527                this.acc += 1;
528             else
529                this.acc += 2.5;
530             father.setAcceleration(this.acc);
531          }
532          else
533          {
534             if (delta < 2)
535                this.acc -= 0.5;
536             else if (delta < 8)
537                this.acc -= 1;
538             else
539                this.acc -= 1.8;
540             if (this.acc < 0)
541             {
542                father.setAcceleration(0);
543                this.acc = 0;
544                this.state = INACTIVE;
545                father.deactiveControl();
546             }
547             else
548                father.setAcceleration(this.acc);
549          }
550       }
551    }
552    }
553
554    public void setAcceleration(double m_acc) {
555
556       if (this.state == ACTIVE)
557          father.setAcceleration(m_acc);
558    }
559
560    public void run() {
561      while (true) {
562      try
563      {
564         Thread.sleep(5);
565         // serve Requests before calculating Acceleration
566         if (this.state == ACTIVE)
567            this.calculateAcceleration();
568      }
569      catch (InterruptedException e){}
```

```
570      }
571    }
572  }
573
574
575  package CruiseControl;
576
577  public class CarModel {
578
579
580    final static int ACTIVE = 1;
581    final static int INACTIVE = 0;
582    final static double deltaT = 1;
583    final static double g = 9.8;
584    final static int m = 1000;
585    public int state = INACTIVE;
586    double acc = 0;
587    double speed = 0;
588    double distance = 0;
589    private static double coeff = 0.31;
590    private double alpha = 0;
591    private Interface father;
592
593    public CarModel() {}
594
595    public CarModel(Interface m_father) {
596      father = m_father;
597    }
598
599    public void engineOn() {
600
601        if (state == INACTIVE)
602        {
603           state = ACTIVE;
604        }
605
606    }
607
608  public void engineOff()
609  {
610     synchronized(this)
611     {
612        //state = INACTIVE;
613        //father.setSpeed(0);
614        //speed = 0;
615        acc = 0;
616     }
617
618
619  }
620
621    public void setSpeed(double m_speed) {
622     synchronized(this)
623     {
624        this.speed = m_speed;
```

```
625     }
626
627    }
628
629    public Double getSpeed() {
630     synchronized(this)
631     {
632        return new Double(this.speed);
633     }
634    }
635
636    public void calculateSpeed(double m_newAcc) {
637        synchronized(this)
638        {
639           if (this.state == ACTIVE)
640           {
641              if ((m_newAcc <= 50) && (m_newAcc >= 0))
642              {
643                 speed = speed + m_newAcc * deltaT - coeff * speed * deltaT -
    Math.sin(alpha) * 4 * g * deltaT;
644                 if (this.speed < 0.005)
645                    this.speed = 0;
646
647              }
648              distance += speed * deltaT / 3600;
649              father.setSpeed(speed);
650              father.setDistance(distance);
651
652           }
653        }
654
655    }
656
657    public void incAcceleration(double m_acc) {
658     synchronized(this)
659     {
660        if (this.state == ACTIVE)
661        {
662           if (((this.acc < 50) && (m_acc > 0)) || ((this.acc > 0) && (m_acc < 0)))
663              this.acc += m_acc;
664        }
665     }
666    }
667
668    public void setAcceleration(double m_acc) {
669     synchronized(this)
670     {
671        this.acc = m_acc;
672     }
673    }
674
675    public void brake() {
676
677        acc = 0;
678        if (speed > 0.01)
```

```
679        {
680        }
681        else
682        {
683           speed = 0;
684        }
685
686     }
687
688
689     public void setAlpha(double m_alpha) {
690
691        alpha = m_alpha;
692
693     }
694
695   public void run(){
696   while (true)
697   {
698      try
699      {
700         Thread.sleep(10);
701         if (this.state == ACTIVE)
702         {
703            this.calculateSpeed(acc);
704         }
705
706
707      }
708      catch (InterruptedException e){}
709   }
710     }
711  }
```

Niveau 2 Logiciel

```
01  using System;
02  using SharpOS;
03  using System.Security.Permissions;
04
05  namespace CruiseControl
06  {
07
08   public class CruiseControlSW
09   {
10
11      public static int SUSPEND_RESUME =1;
12      public static int CRUISING = 2;
13      public static int ACTIVE = 1;
14      public static int INACTIVE = 0;
15      public static int FINISH = 3;
16
17      public SharpOS.Core OS_Core = new SharpOS.Core();
```

```
18      public SharpOS.Task T1;
19      public SharpOS.Task T2;
20
21      static double coeff = 0.31;
22      static double deltaT = 1;
23      static double g = 9.8;
24
25      double acc = 0;
26      double speed = 0;
27      double distance = 0;
28      double desiredSpeed = 0;
29      double alpha = 0;
30      static int state = INACTIVE;
31
32      public  CruiseControlSW()
33      {
34
35      }
36
37      public void Acceleration()
38      {
39         acc = CruiseControlHAL.Load("acceleration");
40      }
41
42      public void DesiredSpeed()
43      {
44         desiredSpeed = CruiseControlHAL.Load("desiredSpeed");
45      }
46
47      public void Alpha()
48      {
49         alpha = CruiseControlHAL.Load("alpha");
50      }
51
52      public void EngineOn()
53      {
54         state = (int) CruiseControlHAL.Load("state");
55      }
56
57      public void EngineOff()
58      {
59         acc = 0;
60         desiredSpeed = 0;
61         state = (int) CruiseControlHAL.Load("state");
62      }
63
64      public void ControlOn()
65      {
66         desiredSpeed = CruiseControlHAL.Load("desiredSpeed");
67         acc = CruiseControlHAL.Load("acceleration");
68         state = (int) CruiseControlHAL.Load("state");
69      }
70
71      public void ControlOff()
72      {
```

```
73          desiredSpeed =  CruiseControlHAL.Load("desiredSpeed");
74          state = (int) CruiseControlHAL.Load("state");
75       }
76
77       public void Brake()
78       {
79          acc = 0;
80          if (speed > 0.01)
81          {
82          }
83          else
84          {
85             speed = 0;
86          }
87       }
88
89       public void Run()
90       {
91
92          System.Console.Write("##### SharpOS V1.0 Port 1 for WIN32.NET #####\n");
93          OS_Core.Init();
94          T1=OS_Core.CreateTask("T1",new
     System.Threading.ThreadStart(CarModelRun),11);
95          T2=OS_Core.CreateTask("T2",new
     System.Threading.ThreadStart(CruiseControlRun),22);
96          System.Console.Write("*****Start Software *****\n");
97          OS_Core.Start();
98
99       }
100
101      private void calculateSpeed(double _acc)
102      {
103         if(state == ACTIVE || state == CRUISING || speed > 0)
104         {
105            if ((_acc <= 50) && (_acc >= 0))
106            {
107
108               speed = speed + _acc * deltaT - coeff * speed * deltaT -
     System.Math.Sin(alpha) * 4 * g * deltaT;
109
110               if (speed < 0.005)
111               {
112                  speed = 0;
113               }
114               distance += speed * deltaT / 3600;
115               lock(this)
116               {
117
118                  CruiseControlHAL.Speed(speed,"sw");
119                  System.Console.WriteLine(speed);
120                  CruiseControlHAL.Distance(distance,"sw");
121               }
122
123            }
124         }
```

```
125     }
126
127     public void CarModelRun()
128     {
129        while (state != FINISH)
130        {
131           OS_Core.TimeDly(T1,10);
132           if (state == ACTIVE || state == CRUISING || speed > 0)
133           {
134              System.Console.WriteLine("***** Calculate Speed *****");
135              calculateSpeed(acc);
136
137
138           }
139        }
140     }
141
142     private void calculateAcceleration()
143     {
144        double currentSpeed;
145        currentSpeed = speed;
146        double delta = System.Math.Abs(currentSpeed - (int)desiredSpeed);
147
148        if (state == CRUISING)
149        {
150           if(delta < 0.4)
151           {
152              System.Console.WriteLine(acc);
153           }
154           else if(currentSpeed < desiredSpeed)
155           {
156              if(delta < 2)
157                 acc += 0.5;
158              else if (delta < 8)
159                 acc += 1;
160              else
161                 acc += 2.5;
162              CruiseControlHAL.Acceleration(acc,"sw");
163              System.Console.WriteLine(acc);
164           }
165           else
166           {
167              if (delta < 2)
168              {
169                 acc -= 0.5;
170              }
171              else if (delta < 8)
172              {
173                 acc -= 1;
174              }
175              else
176              {
177                 acc -= 1.8;
178              }
179              if (acc < 0)
```

```
180             {
181                acc = 0;
182                CruiseControlHAL.Acceleration(acc,"sw");
183                System.Console.WriteLine(acc);
184                state = ACTIVE;
185                CruiseControlHAL.ControlOff("sw");
186             }
187             else
188             {
189                CruiseControlHAL.Acceleration(acc,"sw");
190                System.Console.WriteLine(acc);
191             }
192          }
193       }
194    }
195
196    public void CruiseControlRun()
197    {
198       while(state != FINISH)
199       {
200          OS_Core.TimeDly(T2,5);
201
202          if(state == CRUISING)
203          {
204             System.Console.WriteLine("***** Calculate Acceleration *****");
205             calculateAcceleration();
206          }
207       }
208    }
209
210 }
211 }
```

Niveau 3 Logciel

```
01  #include    "includes.h"
02
03  #define TASK_STK_SIZE   1024  // Stack size, in bytes
04
05  #define SUSPEND_RESUME   // Task 1 and 2 use suspend and resume for scheduling
06  // (if not defined: use time delays)
07
08  #define CRUISING 2
09  #define ACTIVE 1
10  #define INACTIVE  0
11
12  static double coeff = 0.31;
13  static double deltaT = 1;
14  static double g = 9.8;
15  static int m = 1000;
16
17  double acc = 0;
18  double speed = 0;
```

```
19  double distance = 0;
20  double desiredSpeed = 0;
21  double alpha = 0;
22  double temp1;
23  double temp2;
24  double temp3;
25
26  static int state = INACTIVE;
27
28  OS_STK  StartStk[TASK_STK_SIZE]; // Task stacks
29  OS_STK  CarModelRunStk[TASK_STK_SIZE];
30  OS_STK  CruiseControlRunStk[TASK_STK_SIZE];
31
32
33  void Start(void *pdata);  // The 3 tasks
34  void CarModelRun(void *pdata);
35  void CruiseControlRun(void *pdata);
36
37  void AccelerationISR(void);
38  void DesiredSpeedISR(void);
39  void AlphaISR(void);
40  void EngineOnISR(void);
41  void EngineOffISR(void);
42  void ControlOnISR(void);
43  void ControlOffISR(void);
44  void BrakeISR(void);
45
46  extern "C"  __declspec( dllexport ) void __stdcall main_app(void)
47  {
48   // Display a banner.
49   printf("##### uCOS-II V%4.2f Port V%4.2f for WIN32 #####\n",
    ((FP32)OSVersion())/100, ((FP32)OSPortVersion())/100);
50
51   // Initialize uCOS-II.
52   OSInit();
53
54   // Create the first task
55   OSTaskCreate(Start, (void *) 11, &StartStk[TASK_STK_SIZE], 11);
56
57   //Register Interrupt
58   PC_IntVectSet(1, AccelerationISR);
59   PC_IntVectSet(2, DesiredSpeedISR);
60   PC_IntVectSet(3, AlphaISR);
61   PC_IntVectSet(4, EngineOnISR);
62   PC_IntVectSet(5, EngineOffISR);
63   PC_IntVectSet(6, ControlOnISR);
64   PC_IntVectSet(7, ControlOffISR);
65   PC_IntVectSet(8, BrakeISR);
66
67   // Start multitasking.
68   OSStart();
69
70   /* NEVER EXECUTED */
71   printf("main(): We should never execute this line\n");
72  }
```

```
73
74
75  void Start(void *pdata)
76  {
77   printf("%4u: ***** Start Software *****\n", OSTime);
78
79   #if OS_TASK_STAT_EN > 0
80      OSStatInit();                              //Initialize the statistics task
81   #endif
82
83   OSTaskCreate(CarModelRun, (void *) 22, &CarModelRunStk[TASK_STK_SIZE], 22);
84   OSTaskCreate(CruiseControlRun, (void *) 33,
    &CruiseControlRunStk[TASK_STK_SIZE], 33);
85   while (1)
86   {
87      #ifdef SUSPEND_RESUME
88         OSTaskSuspend(OS_PRIO_SELF);
89      #else
90         OSTimeDly(10);
91      #endif
92   }
93  }
94
95  typedef bool (CALLBACK *CALLBACKFUNCTION)(double sp);
96  CALLBACKFUNCTION speed_function;
97  CALLBACKFUNCTION acceleration_function;
98  CALLBACKFUNCTION desiredSpeed_function;
99  CALLBACKFUNCTION distance_function;
100 CALLBACKFUNCTION deactivateControl_function;
101 CALLBACKFUNCTION brake_function;
102 typedef double (CALLBACK *CALLBACKFUNCTION1)(char *str);
103 CALLBACKFUNCTION1 load_function;
104
105 // Hardware CallBack
106
107 extern "C" __declspec( dllexport ) void __stdcall set_speed(CALLBACKFUNCTION pf,
    double sp)
108 {
109     speed_function = pf;
110 }
111 extern "C" __declspec( dllexport ) void __stdcall
    set_acceleration(CALLBACKFUNCTION pf, double sp)
112 {
113     acceleration_function = pf;
114 }
115 extern "C" __declspec( dllexport ) void __stdcall set_distance(CALLBACKFUNCTION
    pf, double sp)
116 {
117     distance_function = pf;
118 }
119 extern "C" __declspec( dllexport ) void __stdcall
    set_desiredSpeed(CALLBACKFUNCTION pf, double sp)
120 {
121     desiredSpeed_function = pf;
122 }
```

```
123 extern "C" __declspec( dllexport ) void __stdcall
    set_deactivateControl(CALLBACKFUNCTION pf, double sp)
124 {
125     deactivateControl_function = pf;
126 }
127 extern "C" __declspec( dllexport ) void __stdcall set_brake(CALLBACKFUNCTION pf,
    double sp)
128 {
129     brake_function = pf;
130 }
131 extern "C" __declspec( dllexport ) void __stdcall set_load(CALLBACKFUNCTION1 pf,
    char *sp)
132 {
133     load_function = pf;
134 }
135
136 // Interrupt function
137
138 void AccelerationISR(void)
139 {
140     OSIntEnter();
141     acc = (*load_function)("acceleration");
142     OSIntExit();
143 }
144 void DesiredSpeedISR(void)
145 {
146     OSIntEnter();
147     desiredSpeed = (*load_function)("desiredSpeed");
148     OSIntExit();
149 }
150 void AlphaISR(void)
151 {
152     OSIntEnter();
153     alpha = (*load_function)("alpha");
154     OSIntExit();
155 }
156 void EngineOnISR(void)
157 {
158     OSIntEnter();
159     state = (*load_function)("state");
160     OSIntExit();
161 }
162 void EngineOffISR(void)
163 {
164     OSIntEnter();
165     acc = 0;
166     desiredSpeed = 0;
167     state = (*load_function)("state");
168     OSIntExit();
169 }
170 void ControlOnISR(void)
171 {
172     OSIntEnter();
173     desiredSpeed = (*load_function)("desiredSpeed");
174     acc = (*load_function)("acceleration");
```

```
175     state = (*load_function)("state");
176     OSIntExit();
177 }
178 void ControlOffISR(void)
179 {
180     OSIntEnter();
181     desiredSpeed = 0;
182     state = (*load_function)("state");
183     OSIntExit();
184 }
185 void BrakeISR()
186 {
187     OSIntEnter();
188     acc = 0;
189     if (speed > 0.01)
190     {
191     }
192     else
193     {
194        speed = 0;
195     }
196     OSIntExit();
197 }
198
199 void calculateSpeed(double _acc)
200 {
201  if(state == ACTIVE || state == CRUISING || speed > 0)
202  {
203     if ((_acc <= 50) && (_acc >= 0))
204     {
205        speed = speed + _acc * deltaT - coeff * speed * deltaT - sin(alpha) * 4 *
    g * deltaT;
206
207        if (speed < 0.005)
208        {
209           speed = 0;
210
211        }
212        distance += speed * deltaT / 3600;
213        bool res = (*speed_function)(speed);
214        printf("%f\n",speed);
215        bool  res1 = (*distance_function)(distance);
216     }
217  }
218 }
219
220 void CarModelRun(void *pdata)
221 {
222
223  while (1)
224  {
225     #ifdef SUSPEND_RESUME
226        OSTaskSuspend(OS_PRIO_SELF);
227     #else
228        OSTimeDly(1);
```

```
229     #endif
230     if (state == ACTIVE || state == CRUISING || speed > 0)
231     {
232        printf("%4u: ***** Calculate Speed *****\n", OSTime);
233        calculateSpeed(acc);
234     }
235  }
236 }
237
238 void calculateAcceleration()
239 {
240  double currentSpeed;
241  currentSpeed = speed;
242  double delta = abs(currentSpeed - (int)desiredSpeed);
243
244  if (state == CRUISING)
245  {
246     if(delta < 0.4)
247     {
248        printf("%f\n",acc);
249     }
250     else if(currentSpeed < desiredSpeed)
251     {
252        if(delta < 2)
253           acc += 0.5;
254        else if (delta < 8)
255           acc += 1;
256        else
257           acc += 2.5;
258        bool res = (*acceleration_function)(acc);
259        printf("%f\n",acc);
260     }
261     else
262     {
263        if (delta < 2)
264        {
265           acc -= 0.5;
266        }
267        else if (delta < 8)
268        {
269           acc -= 1;
270        }
271        else
272        {
273           acc -= 1.8;
274        }
275        if (acc < 0)
276        {
277           acc = 0;
278           bool  res = (*acceleration_function)(acc);
279           printf("%f\n",acc);
280           state = ACTIVE;
281           bool  res1 = (*deactivateControl_function)(1);
282        }
283        else
```

```
284         {
285            bool  res = (*acceleration_function)(acc);
286            printf("%f\n",acc);
287         }
288      }
289   }
290  }
291
292
293  void CruiseControlRun(void *pdata)
294  {
295
296   while(1)
297   {
298      #ifdef SUSPEND_RESUME
299            OSTaskResume(11);
300         OSTaskResume(22);
301      #endif
302
303      OSTimeDly(1);
304      if (kbhit())
305               exit(0);
306
307      if(state == CRUISING)
308      {
309         printf("%4u: ***** Calculate Acceleration *****\n", OSTime);
310         calculateAcceleration();
311
312      }
313   }
314  }
```

ANNEXES B

Le code source de SharpOS

Classe Task

```
01  using System;
02  using System.Security.Permissions;
03
04  namespace SharpOS
05  {
06   /// <summary>
07   /// Structure of SharpOS Task
08   /// </summary>
09   public class Task
10   {
11      private string name;
12      private byte priority;
13      private byte OSX;
14      private byte OSY;
15      private byte OSBitX;
16      private byte OSBitY;
17      private string msg;
18      private int delay;
19      private System.Threading.Thread task_thread;
20      private int pid;
21      private int bit; // initialiser a 0 et = a 1 si c<est le semaphore qui a agit
    sur son thread
22      private System.Byte[] OSMapTbl= {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80};
23      public Task(string thread_name, System.Threading.ThreadStart ts, byte prio,
    int pid, int bit)
24      {
25         this.name=thread_name;
26         this.task_thread = new System.Threading.Thread(ts);
27         this.task_thread.Name=thread_name;
28         this.priority=prio;
29         this.pid=pid;
30         this.bit = 0;
31         OSY = System.Convert.ToByte(System.Convert.ToString( this.priority >> 3));
32         OSBitY = OSMapTbl[OSY];
33         OSX = System.Convert.ToByte(System.Convert.ToString(prio & 0x07));
34         OSBitX = OSMapTbl[OSX];
```

```
35          this.msg =null;
36          this.delay =0;
37          //this.option=0;
38       }
39       public void start()
40       {
41          lock(this)
42          {
43             switch(this.task_thread.ThreadState)
44             {
45                case(System.Threading.ThreadState.Unstarted):
46                this.task_thread.Start();
47                //this.option=2;
48                break;
49                case(System.Threading.ThreadState.Suspended):
50                if(this.bit == 0)
51                {
52                   this.task_thread.Resume();
53                }
54                break;
55                case(System.Threading.ThreadState.WaitSleepJoin):
56                if(this.bit == 0)
57                {
58                   this.task_thread.Interrupt();
59                }
60                break;
61                case(System.Threading.ThreadState.Aborted):
62                break;
63                case(System.Threading.ThreadState.Stopped):
64                break;
65             }
66          }
67       }
68       public void raisePriority()
69       {
70          this.task_thread.Priority = System.Threading.ThreadPriority.AboveNormal;
71       }
72       public void lowerPriority()
73       {
74          this.task_thread.Priority = System.Threading.ThreadPriority.Lowest;
75       }
76       public string get_name()
77       {
78          return name;
79       }
80       public int get_pid()
81       {
82          return pid;
83       }
84       public byte get_prio()
85       {
86          return priority;
87       }
88       public void set_prio(byte val)
89       {
```

```
90          priority = val;
91       }
92       public System.Threading.ThreadState STATUS
93       {
94          get
95          {
96             return this.task_thread.ThreadState;
97          }
98       }
99       public System.Threading.Thread get_thread()
100      {
101         return this.task_thread;
102      }
103      public int get_bit()
104      {
105         return this.bit;
106      }
107      public void set_bit(int i)
108      {
109         this.bit=i;
110      }
111      public byte getOSX()
112      {
113         return OSX;
114      }
115      public void setOSX(byte X)
116      {
117         OSX=X;
118      }
119      public byte getOSY()
120      {
121         return OSY;
122      }
123      public void setOSY(byte Y)
124      {
125         OSY = Y;
126      }
127      public byte getOSBitX()
128      {
129         return OSBitX;
130      }
131      public void setOSBitX(byte bitX)
132      {
133         OSBitX = bitX;
134      }
135      public byte getOSBitY()
136      {
137         return OSBitY;
138      }
139      public void setOSBitY(byte bitY)
140      {
141         OSBitY = bitY;
142      }
143      public void setmsg(string message)
144      {
```

```
145         this.msg = message;
146      }
147      public string getmsg()
148      {
149         return this.msg;
150      }
151      public void setDelay(int dly)
152      {
153         this.delay= dly;
154      }
155      public int getDelay()
156      {
157         return this.delay;
158      }
159   }
160   }
```

Classe Event

```
01   using System;
02   namespace SharpOS
03   {
04    /// <summary>
05    /// Summary description for OSEvent.
06    /// </summary>
07    public class OSEvent
08    {
09       public string OSEventType;
10       public int OSEventCnt;
11       public string OSEventMsg;
12       public byte OSEventGrp;
13       public byte[] OSEventTbl;
14       private System.Byte[] OSMapTbl= {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
     0x80};
15       private System.Byte[] OSUnMapTbl= {
16          0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
17          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
18          5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
19          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
20          6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
21          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
22          5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
23          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
24          7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
25          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
26          5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
27          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
28          6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
29          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
30          5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
31          4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
32       };
33       public OSEvent(string Type, int Cnt, string msg)
34       {
```

```
35          this.OSEventType = Type;
36          this.OSEventCnt = Cnt;
37          this.OSEventMsg = msg;
38          this.OSEventGrp = 0x00;
39          this.OSEventTbl = new byte[8];
40          for (int i=0; i < 8; i++)
41          {
42             this.OSEventTbl[i]=0x00;
43          }
44       }
45       public byte EventTaskRdy( )
46       {
47          byte X, Y , prio;
48          byte OSX;
49          byte OSY;
50          byte OSBitX;
51          byte OSBitY;
52          //trouver la priorite de la tache la plus prioritaire en attente de
    l<evenement
53          Y = OSUnMapTbl[OSEventGrp];
54          X = OSUnMapTbl[OSEventTbl[Y]];
55          prio = System.Convert.ToByte(System.Convert.ToString((Y <<3) + X));
56          //Retirer cette tache de la liste d<Attente de ce semaphore
57          OSY = System.Convert.ToByte(System.Convert.ToString( prio >> 3));
58          OSBitY = OSMapTbl[OSY];
59          OSX = System.Convert.ToByte(System.Convert.ToString(prio & 0x07));
60          OSBitX = OSMapTbl[OSX];
61          OSEventTbl[OSY]=
62          System.Convert.ToByte(System.Convert.ToString(OSEventTbl[OSY] & ~OSBitX
    ));
63          if(OSEventTbl[OSY] == 0x00)
64          {
65             OSEventGrp =
66             System.Convert.ToByte(System.Convert.ToString(OSEventGrp & ~OSBitY));
67          }
68          return prio;
69       }
70       public void EventTaskWait(byte X, byte BitX, byte Y, byte BitY)
71       {
72          this.OSEventGrp =
73          System.Convert.ToByte(System.Convert.ToString(this.OSEventGrp | BitY));
74          this.OSEventTbl[Y] =
75          System.Convert.ToByte(System.Convert.ToString(this.OSEventTbl[Y] | BitX));
76       }
77       public int getCnt()
78       {
79          return this.OSEventCnt;
80       }
81       public void setCnt(int count)
82       {
83          this.OSEventCnt =count;
84       }
85       public byte getEventGrp()
86       {
87          return this.OSEventGrp;
```

```
88      }
89      public string getMsg()
90      {
91         return this.OSEventMsg;
92      }
93      public void setMsg(string msg)
94      {
95         this.OSEventMsg = msg;
96      }
97   }
98  }
```

Classe Semaphore

```
01  using System;
02  using System.Security.Permissions;
03  namespace SharpOS
04  {
05   /// <summary>
06   /// Summary description for Semaphore.
07   /// </summary>
08   public class Semaphore
09   {
10      public int tampon;
11      public SharpOS.OSEvent SemEvent;
12      public Semaphore(int n)
13      {
14         if(n == 0)
15         {
16            System.Console.Write("\n erreur \n");
17         }
18         this.SemEvent = new OSEvent("SEM", n, null);
19         tampon = n;
20      }
21      public bool SemPend(SharpOS.Task T)
22      {
23         while(true)
24         {
25            lock(this)
26            {
27               if(this.SemEvent.getCnt() > 0)
28               {
29                  this.SemEvent.setCnt(this.SemEvent.getCnt()- 1);
30                  return true;
31               }
32               //dans ce cas bloquer la tache en mettant 1 dans
33               OSEventTbl[prio_tache]
34
    this.SemEvent.EventTaskWait(T.getOSX(),T.getOSBitX(),T.getOSY(),T.getOSBitY());
35               return false;
36            }
37         }
38      }
```

```
39      public byte SemPost()
40      {
41         byte prio;
42         lock(this)
43         {
44            if(this.SemEvent.getCnt() < tampon)
45            {
46               this.SemEvent.setCnt(this.SemEvent.getCnt()+1);
47               prio = this.SemEvent.EventTaskRdy();
48               return prio;
49            }
50            else
51            {
52               return 64;
53            }
54         }
55      }
56      public int SemAccept()
57      {
58         int cnt = this.SemEvent.getCnt();
59         if(cnt > 0)
60         {
61            this.SemEvent.setCnt(this.SemEvent.getCnt()- 1);
62         }
63         return cnt;
64      }
65   }
66   }
```

Classe Mailbox

```
01   using System;
02   namespace SharpOS
03   {
04    /// <summary>
05    /// Summary description for Mailbox.
06    /// </summary>
07    public class Mailbox
08    {
09       private SharpOS.OSEvent MboxEvent;
10       public Mailbox(string msg)
11       {
12          MboxEvent = new OSEvent("MBox", 0,msg);
13       }
14       public byte MBoxPost(string msg)
15       {
16          byte prio;
17          lock(this)
18          {
19             if(this.MboxEvent.getMsg() !=null)
20             {
21                return 70;
22             }
```

```
23            else
24            {
25               if(this.MboxEvent.getEventGrp() != 0x00) //une tache est en
26               attente du mbox
27               {
28                  prio = this.MboxEvent.EventTaskRdy();
29                  return prio;
30               }
31               this.MboxEvent.setMsg(msg);
32               return 80; //code 80 aucune tache en attente
33            }
34         }
35      }
36      public string MBoxPend(SharpOS.Task T)
37      {
38         string msg1, msg2;
39         while(true)
40         {
41            lock(this)
42            {
43               msg1 = this.MboxEvent.getMsg();
44               msg2= T.getmsg();
45               if(msg1!=null)
46               {
47                  this.MboxEvent.setMsg(null);
48                  return msg1;
49               }
50               else
51               {
52                  if(msg2!=null)
53                  {
54                     T.setmsg(null);
55                     return msg2;
56                  }
57               }
58
     this.MboxEvent.EventTaskWait(T.getOSX(),T.getOSBitX(),T.getOSY(),T.getOSBitY())
    ;
59               return "0";
60            }
61         }
62      }
63      public string MBoxAccept()
64      {
65         string msg;
66         msg =this.MboxEvent.getMsg();
67         this.MboxEvent.setMsg(null);
68         return msg;
69      }
70   }
71  }
```

Classe Scheduler

```
01  using System;
02  namespace SharpOS
03  {
04   /// <summary>
05   /// Structure of the SharpOS Sceduler
06   /// </summary>
07   public class Scheduler
08   {
09      private SharpOS.Core OSCore;
10      private SharpOS.Task OSRunningTask ;
11      private byte OSHighPrio;
12      public Scheduler(SharpOS.Core OSCore)
13      {
14         this.OSCore = OSCore;
15         this.OSRunningTask = this.OSCore.get_OSArray()[63];
16         this.OSHighPrio = 63;
17      }
18      //System.Collections.ArrayList OSArray
19      /*public void Init( SharpOS.Task[] OSArray , byte[] OSRdyTbl, byte OSRdyGrp)
20      {
21      this.OSArray = OSArray;
22      this.OSRdyTbl=OSRdyTbl;
23      this.OSRdyGrp=OSRdyGrp;
24      }*/
25      public void Start()
26      {
27         byte Y, tampon;
28         while (true)
29         {
30            if (OSRunningTask != null)
31            {
32               TaskDly();
33               if(OSRunningTask.get_bit() == 0 &&
34               OSRunningTask.get_thread().ThreadState ==
    System.Threading.ThreadState.WaitSleepJoin)
35               {
36                  OSRunningTask.lowerPriority();
37               }
38               else
39               {
40                  OSRunningTask.set_bit(1);
41                  OSRunningTask.lowerPriority();
42                  OSRunningTask.get_thread().Suspend();
43               }
44            }
45            //verifier s<il y a une tache en delay, si oui, decrementer son delay
46            Y = OSCore.get_OSUnMapTbl()[OSCore.get_OSRdyGrp()];
47            tampon = OSCore.get_OSRdyTbl()[Y];
48            OSHighPrio = System.Convert.ToByte(System.Convert.ToString((Y
49            <<3) + OSCore.get_OSUnMapTbl()[tampon]));
50            OSRunningTask = GetTaskPrio(OSHighPrio);
51            if(OSRunningTask != null)
52            {
```

```
53              if(OSRunningTask.get_bit() == 1 )
54              {
55                 OSRunningTask.set_bit(0);
56              }
57              OSRunningTask.raisePriority();
58              OSRunningTask.start();
59              System.Threading.Thread.Sleep(10);
60           }
61           else // si aucune tache n<est dans OSArray, resumer la tache idle
62           {
63              OSCore.ResumeIdle();
64           }
65        }
66     }
67     public SharpOS.Task GetTaskPrio(byte ind)
68     {
69        SharpOS.Task OSHighPrioTask;
70        OSHighPrioTask = (SharpOS.Task)this.OSCore.get_OSArray()[ind];
71        return OSHighPrioTask;
72     }
73     public void TaskDly()
74     {
75        SharpOS.Task tsk;
76        for(int i =0; i<63; i++)
77        {
78           tsk =(SharpOS.Task)this.OSCore.get_OSArray()[i];
79           if(tsk !=null)
80           {
81              if(tsk.getDelay() >0)
82              {
83                 tsk.setDelay(tsk.getDelay() -1);
84                 if(tsk.getDelay()==0)
85                 {
86                    this.OSCore.TimeDlyResume(tsk);
87                 }
88              }
89           }
90        }
91     }
92     public SharpOS.Task OSRunTask
93     {
94        get
95        {
96           return OSRunningTask;
97        }
98     }
99  }
100 }
```

Classe Core

```
01  using System;
02  namespace SharpOS
```

```
03  {
04   /// <summary>
05   /// Main Structure of SharpOS
06   /// </summary>
07   public class Core
08   {
09      private Scheduler sched;
10      private System.Byte[] OSMapTbl= {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40,
    0x80};
11      private System.Byte[] OSUnMapTbl= {
12         0, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
13         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
14         5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
15         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
16         6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
17         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
18         5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
19         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
20         7, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
21         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
22         5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
23         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
24         6, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
25         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
26         5, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
27         4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0
28      };
29      private SharpOS.Task[] OSArray = new SharpOS.Task[64];
30      private byte OSRdyGrp ;
31      private byte[] OSRdyTbl = new byte[8];
32      private System.Threading.Thread SchedThread;
33      private int pid;
34      public Core()
35      {
36         this.OSRdyGrp=0x00;
37         for(int i = 0; i < 8 ; i++)
38         {
39            this.OSRdyTbl[i]=0x00;
40         }
41         sched = new Scheduler(this);
42      }
43      public void Init()
44      {
45         CreateTask("IDLE", new System.Threading.ThreadStart(idle),63);
46      }
47      public void Start()
48      {
49         //sched.Init(OSArray,OSRdyTbl,OSRdyGrp);
50         System.Console.WriteLine("Starting Simulation");
51         System.Threading.Thread.Sleep(5);
52         this.SchedThread = new System.Threading.Thread(new
53         System.Threading.ThreadStart(sched.Start));
54         this.SchedThread.Name = "Scheduler";
55         this.SchedThread.Priority = System.Threading.ThreadPriority.Highest;
56         SchedThread.Start();
```

```
57      }
58      public SharpOS.Task CreateTask(string thread_name,
    System.Threading.ThreadStart ts,
59      byte prio)
60      {
61         SharpOS.Task T = new SharpOS.Task(thread_name, ts, prio, get_pid(),0);
62         OSRdyGrp = System.Convert.ToByte(System.Convert.ToString(OSRdyGrp |
63         T.getOSBitY()));
64         OSRdyTbl[T.getOSY()] =
65         System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[T.getOSY()] |
    T.getOSBitX()));
66         OSArray[prio] = T;
67         return T;
68      }
69      public void Suspend(SharpOS.Task T)
70      {
71         this.SchedThread.Priority=System.Threading.ThreadPriority.Lowest;
72         int prio = T.get_prio();
73         //enlever la tache de OSRdyTbl et mise a jour de OSRdyGrp
74         OSRdyTbl[T.getOSY()]=
75         System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[T.getOSY()] &
    ~(T.getOSBitX()));
76         if(OSRdyTbl[T.getOSY()] == 0x00)
77         {
78            OSRdyGrp = System.Convert.ToByte(System.Convert.ToString(
79            OSRdyGrp & ~(T.getOSBitY()));
80         }
81         T.set_bit(1);
82         //Suspendre son thread
83         OSArray[prio].get_thread().Suspend();
84         this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
85      }
86      public bool Resume(SharpOS.Task T1, SharpOS.Task T2)
87      {
88         byte prio = T2.get_prio();
89         //Vérifier que la tache est suspendue
90         if(OSArray[prio] != null) //verifier que cette tache existe
91         {
92            if(T2.get_thread().ThreadState ==
93            System.Threading.ThreadState.Suspended)
94            {
95               //Ajouter la tache a la liste des taches pretes a s<ex/cuter
96               OSRdyGrp =
97               System.Convert.ToByte(System.Convert.ToString( OSRdyGrp |
    T2.getOSBitY()));
98               OSRdyTbl[T2.getOSY()] =
99               System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[T2.getOSY()]
    | T2.getOSBitX()));
100              OSArray[prio].set_bit(0);
101              T1.get_thread().Suspend();
102              return true;
103           }
104           return false;
105        }
106        return false;
```

```
107     }
108     public void ResumeIdle()
109     {
110        this.SchedThread.Priority=System.Threading.ThreadPriority.Lowest;
111        if(OSArray[63].get_thread().ThreadState ==
112        System.Threading.ThreadState.Suspended)
113        {
114           OSRdyGrp = 0x80;
115           OSRdyTbl[0x7] =0x80;
116           OSArray[63].set_bit(0);
117        }
118        this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
119     }
120     public void SemPost(SharpOS.Semaphore s, SharpOS.Task T)
121     {
122        this.SchedThread.Priority=System.Threading.ThreadPriority.Lowest;
123        byte prio;
124        prio = s.SemPost();
125        if(prio == 64)
126        {
127           System.Console.WriteLine("ce semaphore ne peut etre poster");
128        }
129        else
130        {
131           //Ajouter la tache a la liste des taches pretes a s<ex/cuter
132           if(OSArray[prio] != null)
133           {
134              OSRdyGrp =
135              System.Convert.ToByte(System.Convert.ToString( OSRdyGrp |
    OSArray[prio].getOSBitY()));
136              OSRdyTbl[OSArray[prio].getOSY()] =
137
     System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[OSArray[prio].getOSY()]
    |
138              OSArray[prio].getOSBitX()));
139              if(OSArray[prio].get_bit() == 1)
140              {
141                 OSArray[prio].set_bit(0);
142              }
143           }
144        }
145        T.get_thread().Suspend();
146        this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
147        //this.SchedThread.Resume();
148     }
149     public void SemPend(SharpOS.Semaphore s , SharpOS.Task T)
150     {
151        this.SchedThread.Priority = System.Threading.ThreadPriority.Lowest;
152        //int prio = T.get_prio();
153        if(s.SemPend(T)==false) // tache en attente du semaphore
154        {
155           //enlever la tache de OSRdyTbl et mise a jour de OSRdyGrp
156           OSRdyTbl[T.getOSY()]=
157           System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[T.getOSY()] &
    ~(T.getOSBitX())));
```

```
158           if(OSRdyTbl[T.getOSY()] == 0x00)
159           {
160              OSRdyGrp =
161              System.Convert.ToByte(System.Convert.ToString( OSRdyGrp &
    ~(T.getOSBitY())));
162           }
163           T.set_bit(1);
164           T.get_thread().Suspend();
165        }
166        this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
167     }
168     public void MboxPost(SharpOS.Mailbox mb,SharpOS.Task T, string msg)
169     {
170        this.SchedThread.Priority=System.Threading.ThreadPriority.Lowest;
171        byte prio;
172        prio = mb.MBoxPost(msg);
173        if(prio == 70)
174        {
175           System.Console.WriteLine("Erreur;Mailbox occupe");
176        }
177        else
178        {
179           if(prio !=80) //tache en attente du messge
180           {
181              if(OSArray[prio] != null)
182              {
183                 //Ajouter la tache a la liste des taches pretes a
184                 s<ex/cuter
185                 OSRdyGrp =
186                 System.Convert.ToByte(System.Convert.ToString( OSRdyGrp |
    OSArray[prio].getOSBitY()));
187                 OSRdyTbl[OSArray[prio].getOSY()] =
188
     System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[OSArray[prio].getOSY()]
    |
189                 OSArray[prio].getOSBitX()));
190                 OSArray[prio].setmsg(msg);
191                 if(OSArray[prio].get_bit() == 1)
192                 {
193                    OSArray[prio].set_bit(0);
194                 }
195              }
196           }
197        }
198        T.get_thread().Suspend();
199        this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
200     }
201     public string MboxPend(SharpOS.Mailbox mb,SharpOS.Task T, int timeout) //voir
    pout
202     lr timeout
203     {
204        this.SchedThread.Priority = System.Threading.ThreadPriority.Lowest;
205        string msg=mb.MBoxPend(T);
206        if(msg=="0") // tache en attente d<un message
207        {
```

```
208           //enlever la tache de OSRdyTbl et mise a jour de OSRdyGrp
209           OSRdyTbl[T.getOSY()]=
210           System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[T.getOSY()] &
    ~(T.getOSBitX()));
211           if(OSRdyTbl[T.getOSY()] == 0x00)
212           {
213              OSRdyGrp =
214              System.Convert.ToByte(System.Convert.ToString( OSRdyGrp &
    ~(T.getOSBitY()));
215           }
216           T.set_bit(1);
217           T.get_thread().Suspend();
218        }
219        this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
220        return msg;
221     }
222     public void TimeDly(SharpOS.Task T, int timeout)
223     {
224        this.SchedThread.Priority=System.Threading.ThreadPriority.Lowest;
225        byte prio = T.get_prio();
226        if(timeout > 0)
227        {
228           //enlever la tache de OSRdyTbl et mise a jour de OSRdyGrp
229           OSRdyTbl[T.getOSY()]=
230           System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[T.getOSY()] &
    ~(T.getOSBitX()));
231           if(OSRdyTbl[T.getOSY()] == 0x00)
232           {
233              OSRdyGrp =
234              System.Convert.ToByte(System.Convert.ToString( OSRdyGrp &
    ~(T.getOSBitY()));
235           }
236           T.setDelay(timeout);
237           //Faire endormir le thread correspondant
238           T.set_bit(1);
239           T.get_thread().Suspend();
240        }
241        this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
242     }
243     public void TimeDlyResume(SharpOS.Task T)
244     {
245        //ajouter la tache a la liste des taches pretes
246        this.SchedThread.Priority=System.Threading.ThreadPriority.Lowest;
247        OSRdyGrp = System.Convert.ToByte(System.Convert.ToString( OSRdyGrp |
248        T.getOSBitY()));
249        OSRdyTbl[T.getOSY()] =
250        System.Convert.ToByte(System.Convert.ToString(OSRdyTbl[T.getOSY()] |
    T.getOSBitX()));
251        OSArray[T.get_prio()].set_bit(0);
252        this.SchedThread.Priority=System.Threading.ThreadPriority.Highest;
253     }
254     public int get_pid()
255     {
256        pid++;
257        return pid-1;
```

```
258     }
259     public void idle()
260     {
261        for(;;)
262        {
263           int i =0;
264           while (i < 20)
265           {
266              System.Console.Write("i");
267              i++;
268           }
269           i = 0;
270           Suspend(OSArray[63]);
271        }
272     }
273     public System.Byte[] get_OSUnMapTbl()
274     {
275        return this.OSUnMapTbl;
276     }
277     public System.Byte[] get_OSMapTbl()
278     {
279        return this.OSMapTbl;
280     }
281     public SharpOS.Task[] get_OSArray()
282     {
283        return this.OSArray;
284     }
285     public System.Byte get_OSRdyGrp()
286     {
287        return this.OSRdyGrp;
288     }
289     public void set_OSRdyGrp( byte Grp)
290     {
291        this.OSRdyGrp = Grp;
292     }
293     public System.Byte[] get_OSRdyTbl()
294     {
295        return this.OSRdyTbl;
296     }
297     public void set_OSRdyTbl(int i, byte Tbl)
298     {
299        this.OSRdyTbl[i]=Tbl;
300     }
301  }
302  }
```