2010

Guilhem BELLION
Christophe DUC
Jonathan GOBBO
Thomas GRAVINA
Laura VIKOR

Encadrant : Michel BUFFA
Société : ROBOSOFT

[RAPPORT PROJET LUA]

Ce document est une synthèse de nos travaux réalisés en Master 1 MIAGE à Nice dans le cadre de notre projet d'année.

1 SOMMAIRE

2 INTRODUCTION

À l'heure où les progrès technologiques en robotique se font de plus en plus pressants, il convient de bien les intégrer à l'environnement industriel et/ou à notre quotidien. Le principal domaine concerné ? La robotique mobile. Alors que les robots manipulateurs se sont généralisés dans l'industrie, les robots mobiles prennent véritablement leur essor depuis seulement quelques années. La principale raison de ce retard est la complexité des systèmes mobiles. Tandis que les robots manipulateurs évoluent dans des environnements connus et sont programmés pour effectuer des tâches répétitives, les robots mobiles expriment des besoins plus complexes. Ils doivent en effet pouvoir s'adapter à leur environnement et travailler de manière partiellement autonome.

Si quelques robots mobiles de pointe sont présents depuis des dizaines d'années dans certains domaines précis (militaire, aéroports, …), ils n'ont pas encore pris toute leur ampleur dans les mondes industriel et domestique. Les domaines d'application sont pourtant nombreux et à fort potentiel. Du transport de matières dangereuses à l'assistance aux personnes âgées, en passant par l'assainissement de surface, les robots mobiles proposent des solutions viables et sécurisées. Toutefois, même s'ils sont souvent capables d'évoluer dans des environnements qui leurs sont inconnus, il peut parfois être intéressant de personnaliser ces robots pour les adapter à des situations et leur faire adopter des comportements très précis.

Avoir un contrôle presque total sur un robot mobile ? Un objectif excitant. Afin de rendre ses solutions robotiques plus modulables, la société ROBOSOFT a souhaité le satisfaire en offrant aux utilisateurs la possibilité de contrôler les robots avec des scripts écrits dans le langage de programmation Lua, parfaitement adapté à ce type de situation.

Dans le cadre de notre projet d'année de Master 1 MIAGE, nous avons donc été chargés de développer un éditeur de scripts Lua destiné à être déployé dans les applications éditées par ROBOSOFT. Cet éditeur permettra donc d'écrire des scripts en Lua et de les exécuter pour interagir avec les robots. Pour cela, il doit disposer des fonctionnalités de base de tout éditeur de code (coloration syntaxique, indentation automatique, auto-complétion, etc.) ainsi que d'une rubrique d'aide et d'exemples.

Ce document a pour vocation de présenter le contexte, les solutions envisagées et implémentées, et de relater en détails les différentes étapes de la réalisation du projet. Seront également exposés les points forts et les points faibles de notre application, ainsi que les différents problèmes auxquels nous avons fait face.

3 PRESENTATION DU SUJET

3.1 APPLICATION A REALISER

Ce projet s'inscrit dans la mise en place du langage Lua en interaction avec les robots de la société Rosoft, à travers un éditeur de script avancé. Une premiere approche avait déjà été traitée il y a quelques années par Mr. Michel Buffa (enseignant-chercheur à l'Université de Nice Sophia-Antipolis) mais rapidement abandonnée par la société.

Suite à une visite du directeur de ROBOSOFT chez l'un de ses clients qui avait réussi à piloter l'un de ses robot en utilisant Lua, la société à décidé de relancer ce projet afin de permettre à tous leurs clients le pilotage complet de leurs robots.

Dans le cadre de notre projet d'année de Master 1 MIAGE (Méthodes Informatique Appliquées à la Gestion des Entreprises) à l'Université de Nice Sophia-Antipolis, nous avons eu à concrétiser ce projet. L'équipe de developpement est composée de cinq étudiants : Guilhem BELLION, Christophe DUC, Jonathan GOBBO, Thomas GRAVINA, Laura VIKOR.

L'application à developper est un editeur de scripts complet (auto-complétion, vérification syntaxique, fonctions de sauvegarde, ouverture de fichiers, aide, etc…), permettant à l'utilisateur de créer ses propres scripts en Lua et de les tester directement sur un robot connecté ou via une simulation.

Nous avons pour cela, utilisé le langage C# (du framework .NET), l'environnement de developpement Visual Studio 2008 ainsi que la plateforme Microsoft Robotics Developer Studio.

Le but final, sera de permettre aux clients de la société ROBOSOFT, de pouvoir piloter leurs robots en utilisant leurs propres scripts.

4 PRESENTATION DE LA SOCIETE ROBOSOFT

Créée en 1985 et basée à Bidart, ROBOSOFT est la première start-up issue des laboratoires de l'INRIA. Elle est aujourd'hui pionnière et leader sur le marché des solutions de robotique avancée. Elle possède de nombreux partenaires :

Microsoft	Microsoft® Robotics Studio
	INRIA
LaRA	LRP (Laboratoire de Robotique de Paris) et LARA projet de l'INRIA.
GSF NETTOYAGE INDUSTRIEL	GSF, entreprise de nettoyage industrielle
KARTO by SRI International	Karto, fournit un Kit de développement logiciel robotique pour intégrer la navigation de robots intelligents, la cartographie et l'exploration des capacités dans leur plate-forme mobile.

L'entreprise propose une gamme complète de robots intelligents et modulaires, destinés à l'automatisation des services comme :

<u>La Sécurité :</u>

Pour automatiser l'intérieur et de la surveillance extérieure des bâtiments et des terrains, ROBOSOFT fournit aux gestionnaires de sites et aux entrepreneurs de la sécurité des robots de surveillance au sol. Ceux-ci peuvent être personnalisés en particulier par l'intégration de tout type de capteur.

<u>La Santé :</u>

Il existe deux types de solutions de robotique pour la santé:

Celles pour réaliser des opérations à distance, comme la télé-échographie, permettent aux spécialistes de pratiquement n'importe où et à n'importe quel moment de diagnostiquer d'urgence.

Celles d'assistance aux personnes à la maison, en aidant les personnes âgées et handicapées de se déplacer et de communiquer.

<u>Le Transport :</u>

ROBOSOFT fournit divers robots de transports pour les personnes, les

marchandises et le ravitaillement.

La Propreté :

Cette société propose deux types de robots un pour la propreté des vitres et l'autre pour la propreté des sols.

Les Véhicules Robotisés :

ROBOSOFT produit aussi des véhicules robotisés réalisés au travers d'un processus de robotisation fondée sur robuBOX (voir ci-dessous).

Cette diversité dans les solutions proposées permet à ROBOSOFT d'avoir une Multitude de clients. En effet elle possède de nombreux clients dans les domaines suivants :

- Recherche et Développement de grands groupes publics et privés: ils ont tous besoin de composants génériques et de soutien lié à la construction de robots de service personnalisé par eux-mêmes.
- Enseignement de la robotique
- Entreprises de nettoyage
- Réseaux de transports publics
- Producteurs des chaînes de télévision ou de films : capture des images en mouvement ou filmer dans des conditions difficiles pour l'homme.
- Fabricants de véhicules industriels : introduction des fonctions robotiques dans les véhicules existants afin de les rendre autonomes.

5 ENVIRONNEMENT TECHNIQUE

5.1 LANGAGE DE PROGRAMMATION LUA

5.1.1 PRESENTATION

Lua est un langage informatique de script créé en 1993. Le langage est utilisé par de nombreux jeux commerciaux comme FarCry, Blue Mars ou encore le mondialement connu : World of Warcraft. Des sociétés reconnues comme Adobe, utilisent également ce langage dans certaines de leurs solutions commerciales.

Le fait d'utiliser un langage de script permet notamment à des personnes ne connaissant pas vraiment toute les subtilités d'un langage de programmation (allocation mémoire, pointeurs, etc.) de pouvoir modifier un programme (les langages de scripts ont en général une syntaxe plus simple et se limitent a des cas d'utilisations connus). Ainsi, des graphistes et « level designers » de chez Blizzard (le créateur de World of Warcraft) peuvent modifier le comportement du jeu simplement en utilisant un langage de script.

Du point de vue de la robotique, le langage Lua peut permettre à des personnes n'ayant pas de connaissances avancées en programmation, de contrôler très facilement leur robot pour leur faire adopter des comportements totalement personnalisés. Comme tout langage de script, Lua s'intègre parfaitement à de gros environnement et devient une brique logicielle presque indispensable.

Les caractéristiques de Lua sont diverses :

- Performance (rapidité d'exécution),
- Portabilité (le langage peut être utilisé sous Unix, Windows et dans des terminaux portable ou encore des micro-processeurs),
- Embarquable (utilisable en plus d'un autre langage de programmation comme JAVA/C#/C/ADA…),
- Leger & gratuit (l'interpréteur Lua ne pèse qu'aux alentours de 200Ko et le langage est peut être utilisé dans des solutions commerciales sans frais).

5.1.2 CARACTERISTIQUES

Afin de pourvoir écrire des scripts il faut connaitre un minimum les caractéristiques de ce langage. Lua utilise une structure de données flexible (les tables, à l'image de JavaScript), des variables globales peuvent être définis à travers les scripts et des librairies standards sont présentes (io, math, file, string, …).

Les types de variables sont divers et communs aux autres langages :

- Table,
- Nombre,
- Booléen,
- Chaines de caractères,
- Fonctions …

Les Tables peuvent être sous forme d'Array ou de List :

```
a = { 1,2,3,4 };
→ print(a[1])

A = { [0]=1,2,3,4 };
→ print(A[0])
```

Ou encore sous forme de Dictionnaire :

```
d = {i=1,j=2,k=3};
→ print(d["i"])

d = {i=4,j=5,k=6};
→ print(d.i)
```

Il est important de noter, que les éléments de ces Tables peuvent être de n'importe quelle sorte. Tout est basé sur les Tables dans le langage Lua.

Lua possède les principales fonctions algorithmiques des autres langages comme par exemple :

IF THEN ELSE

```
if type(a)== "table" then
    io.write("a is table\n")
else if type(a)=="number"
then
    io.write("a is number\n")
end
```

WHILE

```
a = { 1,2,3,4,5,6}
i = 1
while a[i] do
    print(a[i])
    i = i + 1
end
```

FOR

```
for a=1,10,2 do
    print(a)
end
```

Un autre avantage de ce langage est de pouvoir utiliser des scripts en mode texte ou des scripts compilés. Les scripts compilés seront chargés plus rapidement que les scripts lisibles par l'utilisateur (mais le temps d'exécution restera inchangé).

5.2 MICROSOFT VISUAL STUDIO ET C#

Le logiciel Microsoft Visual Studio 2008 a été utilisé pour le développement de l'éditeur Lua. Celui-ci permet de développer dans de nombreux langages de programmation propriétaires de Microsoft comme C#, ASP.NET, etc.

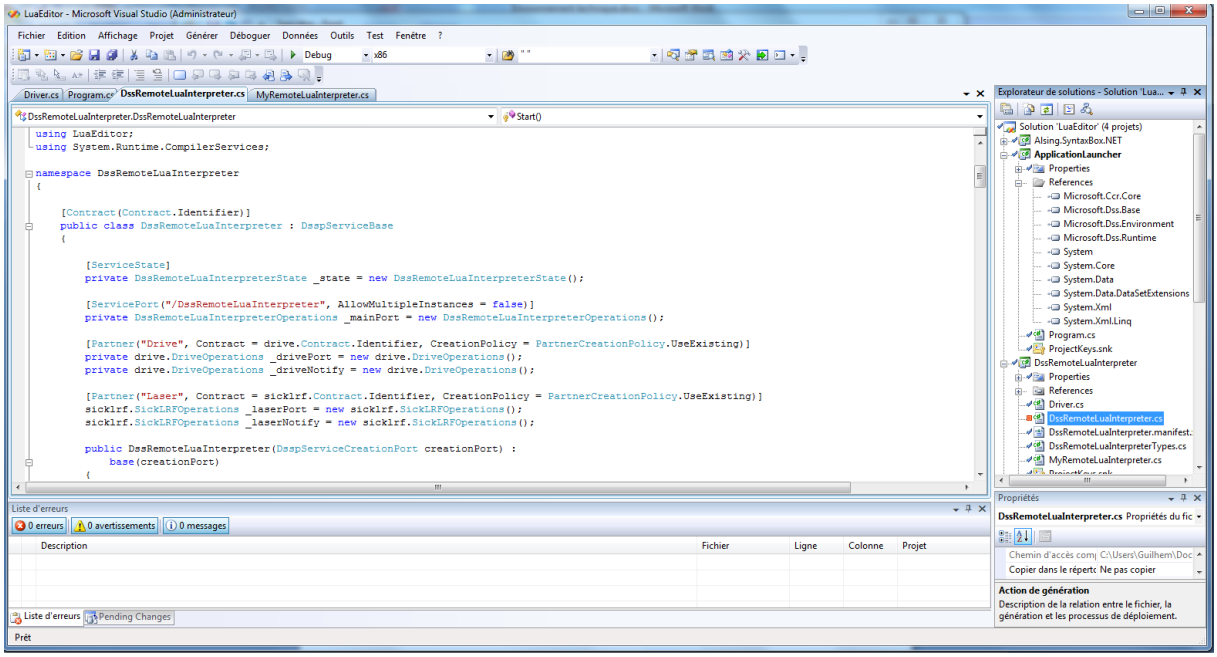

Figure 1 - Aperçu de l'environnement de développement Microsoft Visual Studio.

Afin de synchroniser notre travail au sein du groupe, nous avons également utilisé le plugin AnkhSVN qui permet de se connecter à un repository Subversion et d'y effectuer toutes les opérations nécessaires au partage de fichiers (Checkout, Update, Commit, etc.).

L'éditeur de scripts Lua a été développé avec le langage de programmation C#. Référence et principal langage de Microsoft, C# est orienté objet à l'image de son concurrent Java. L'utilisation de ce langage a été naturellement imposée par la société ROBOSOFT et plus généralement l'environnement robotique qui se base sur le framework .NET, dont C# fait partie.

5.3 MICROSOFT ROBOTICS DEVELOPER STUDIO

Notre application étant destinée à être utilisée pour la robotique, il nous a fallu travailler avec les outils sur lesquels s'appuie la société ROBOSOFT. Microsoft Robotics Developer Studio fait parti de ceux-ci et se pose comme la référence pour le développement robotique. Dans le cadre de notre projet, nous avons utilisé une version gratuite (licence dite *Express*) de cette solution. Cette dernière n'est pas restrictive par rapport à la solution complète et, proposée pour une utilisation non-commerciale, était adaptée à notre cas (application Open Source).

5.3.1 PRESENTATION DE LA PLATEFORME

Microsoft Robotics Developer Studio (MSRDS) est une plateforme de développement standardisée pour la robotique. Editée par Microsoft, cette plateforme est composée d'environnements d'exécution, de développement et de simulation.

Cette solution a vu le jour en 2006 après un désir de la part de Microsoft de démocratiser le développement robotique, et de proposer une solution complète aux industriels. En effet, jusqu'alors les sociétés impliquées dans la robotique devaient s'adapter à l'environnement technologique (système d'exploitation et langage de programmation) de chaque constructeur de robots programmables. La solution MSRDS permet de palier à ce problème.

En effet, cette plateforme propose un environnement standardisé, c'est-à-dire qu'elle permet de rendre indépendantes les parties matérielle (hardware) et logicielle (software) d'un robot. Il est donc possible d'exécuter à l'identique le même code source sur plusieurs plateformes matérielles différentes, ce qui est un avantage considérable pour les sociétés concernées. Cette portabilité du code, objectif majeur de Microsoft pour MSRDS, permet donc de s'abstraire totalement du choix du hardware sous-jacent.

S'appuyant sur le framework .NET, technologie propriétaire de Microsoft, MSRDS permet donc de développer, tester, simuler et débugger des applications robotiques. Il repose sur trois « briques » technologiques :

- Un environnement d'exécution : le *runtime*. Sur la base des modèles CCR et DSS, il permet de faire tourner des programmes sur n'importe quel robot.
- Un environnement de développement, qui met à disposition une sur-couche du framework .NET pour la robotique, accessible grâce à des outils spécialisés (Visual Programming Language, …) ou plus généralement de la gamme Microsoft (Visual Studio).
- Un environnement de simulation, qui permet de tester les programmes développés en faisant évoluer des robots dans des espaces virtuels en 3D, avec moteurs physiques pour restituer les chocs, la gravité, les forces, etc.

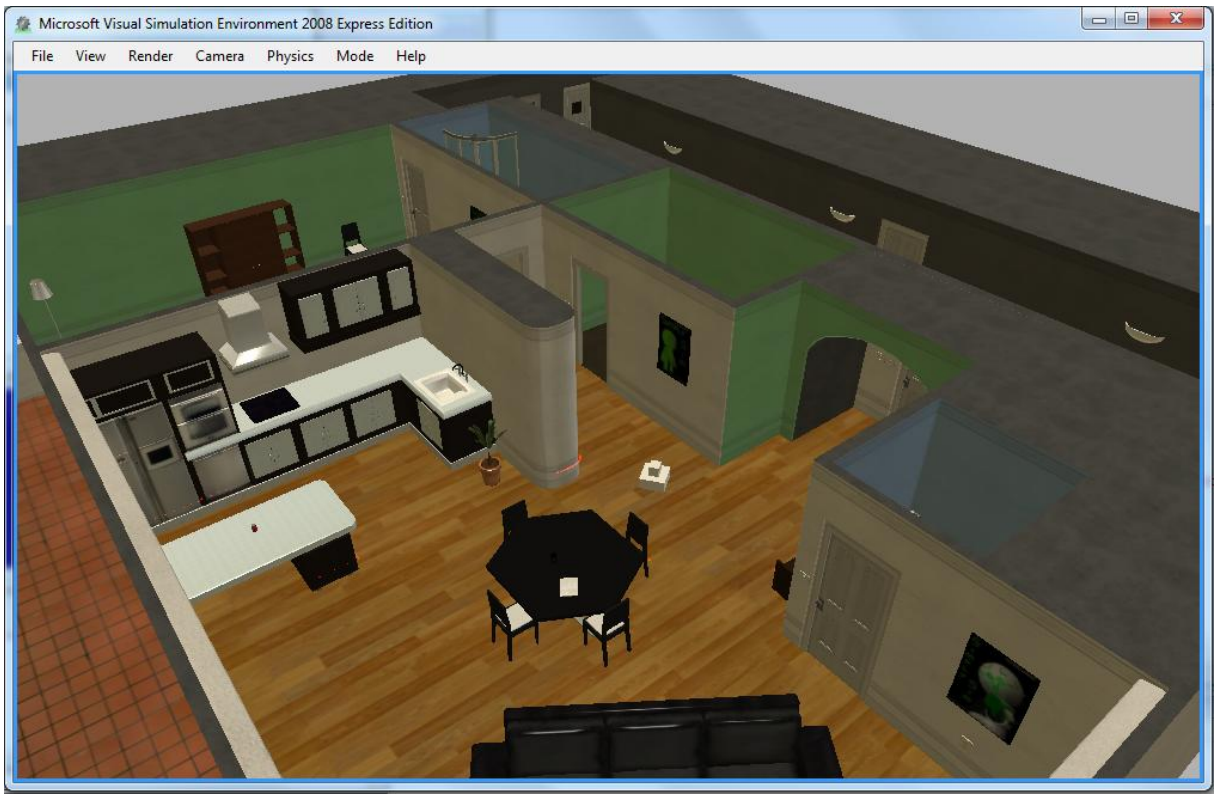

Figure 2 – Environnement de simulation de Microsoft Robotics Developer Studio : robot dans un appartement.

5.3.2 ARCHITECTURE LOGICIELLE

Microsoft Robotics Developer Studio adopte une architecture logicielle particulière pour faciliter le développement d'applications, face à la complexité de programmation en robotique. En effet, de nombreux problèmes se posent au niveau de la gestion des tâches en parallèle. Par exemple, un programme doit pouvoir interagir avec les composants du robot en même temps que celui-ci traite des informations relatives à ses capteurs. Ce problème se pose aux trois niveaux principaux d'un robot, qui sont les suivants :

- Un ensemble de capteurs permettant au robot d'analyser son environnement : laser, détecteur de mouvement, capteur de contact frontal, détecteur de chaleur, etc.
- Un centre de calcul (processeur) qui permet d'analyser les informations des capteurs, de gérer les entrées/sorties avec des programmes extérieurs, et de manipuler les mécanismes physiques.
- Des moteurs ou plus généralement des dispositifs mécaniques (actuateurs) permettant au robot d'effectuer des mouvements et d'agir sur l'environnement suivant les instructions reçues.

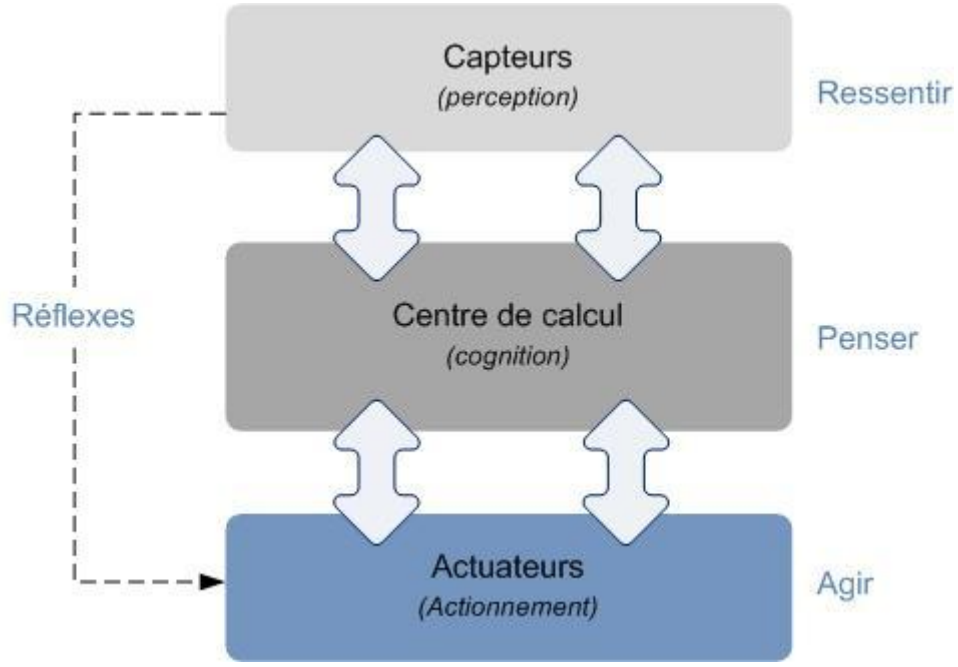

Figure 3 - Différentes composantes physiques et logicielle d'un robot mobile.

Pour assurer la cohérence de fonctionnement et la synchronisation de ces composantes du robot avec les applications les utilisant, MSRDS met en jeu 3 couches distinctes :

- Les services,
- L'environnement CCR,
- L'environnement DSS.

5.3.2.1 SERVICES

Les services sont des entités logicielles ou matérielles qui s'exécutent sur un PC, un serveur ou un robot. MSRDS permet à un utilisateur de créer des services personnalisés. Un service peut-être vu soit comme un composant matériel déployé sur un robot (comme un capteur, un laser, etc.), soit comme un composant logiciel de différentes natures (programme embarquable, interpréteur de scripts, interface graphique, etc.), soit comme un ensemble de composants logiciels et/ou matériels.

Chaque service a une structure bien précise afin de correspondre avec le standard mis en place par MSRDS. Le modèle de programmation d'un service met en jeu les éléments suivants :

- Un contrat,
- Un identifiant unique,
- Un état,
- Des ports de communication entrées/sorties avec « handlers »,
- Un gestionnaire de notifications,
- Des services partenaires,
- Un manifeste XML.

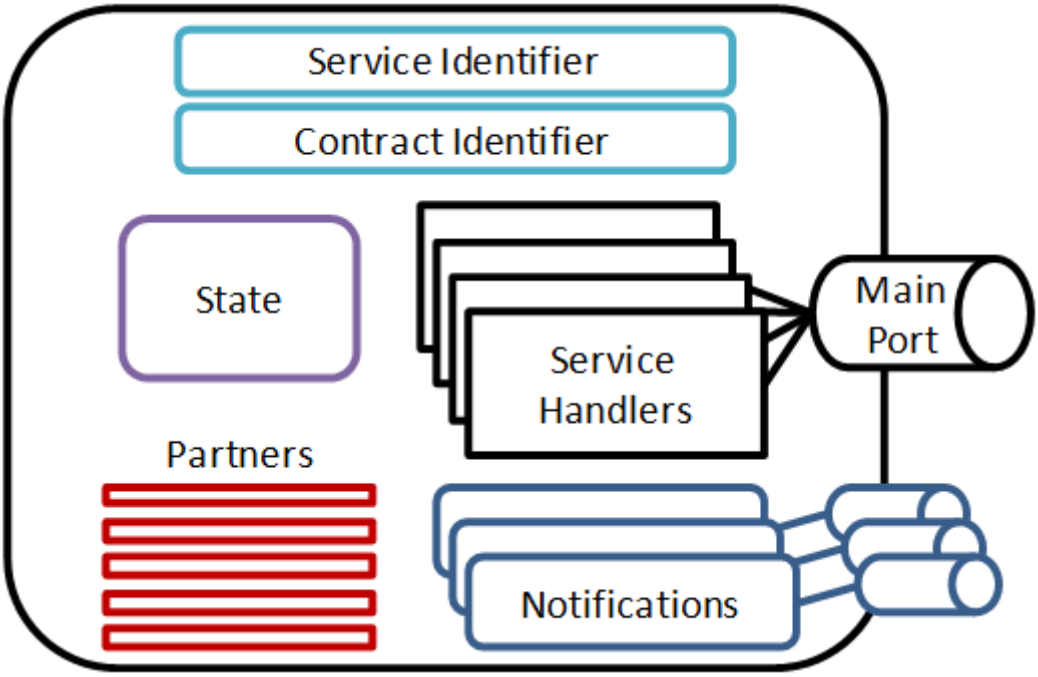

Figure 4 - Schéma représentant l'architecture d'un service.

5.3.2.1.1 CONTRAT DU SERVICE

Chaque service déployé dans l'environnement d'exécution doit avoir un contrat. Ce dernier permet de décrire le comportement du service et la manière dont on doit le consommer (i.e. l'utiliser). Accessibles grâce à une URI, les contrats permettent également aux services d'établir des relations automatiques entre eux.

5.3.2.1.2 IDENTIFIANT UNIQUE

Le contrat de chaque service comporte un identifiant unique sous forme d'URI qui sert, lors du démarrage du service, à identifier ses différentes instances de manière unique et sûre. L'identificateur de chaque instance est alloué dynamiquement au démarrage. Cet identifiant est utilisé par des services tiers pour les partenariats.

5.3.2.1.3 ETAT DU SERVICE

L'état d'un service peut être représenté à tout instant. Cela permet à des services partenaires d'obtenir les valeurs relatives au service à tout instant. Par exemple, pour un service de motorisation qui s'exécute sur un robot, l'état pourrait renseigner un service tiers de la puissance exercée, de la vitesse, etc.

5.3.2.1.4 PORT DE COMMUNICATION ET « HANDLERS »

Pour assurer leurs communications, les services déclarent un port principal permettant d'échanger des messages avec les autres services. Ce port définit également concrètement les opérations que le service propose. Ainsi, pour des raisons de sécurité et de compatibilité, le port de ne peut accepter que des messages de types qu'il définit. De plus, une méthode « handler » (i.e. : gestionnaire) est associée à chaque type d'opération. Ces méthodes définissent le comportement du service lors de la réception d'un message provenant d'un service tiers, et peuvent modifier son état en conséquence. Le code source suivant illustre l'ordre qu'un service, souhaitant diriger un robot, envoie au service gérant la motorisation du robot :

```
private ServiceEngineOperations _mainPort = new ServiceEngineOperations();
...
// Envoi du message "avance" au moteur
_mainPort.Post(new Forward());
```

Figure 5 - Section de code C# d'un service dont le but est de diriger un robot

```
[ServicePort]
public class ServiceEngineOperations : PortSet<...,Forward,...> {}
...
[ServiceHandler(...)]
public IEnumerator<ITask> ForwardHandler(Forward forward)
{
   // Réception du message "avance" venant d'un service tiers
   // Implémentation du comportement du moteur pour faire avancer le robot
   ...
}
```

Figure 6 – Section de code C# d'un service gérant la motorisation d'un robot

5.3.2.1.5 GESTIONNAIRE DE NOTIFICATIONS

Lorsqu'un service un autre service comme partenaire (voir la section « Partenariats »), il peut être alerté des modifications de son état grâce à un principe de notifications. Si un service A s'inscrit aux notifications d'un service partenaire B, alors il recevra automatiquement des alertes générées par B lorsque l'état de ce dernier sera modifié. Par exemple, cela peut être

particulièrement utile pour un service ayant comme partenaire un service de mesure de température et souhaitant effectuer des opérations à certaines températures.

5.3.2.1.6 SERVICES PARTENAIRES

Le but des services est de proposer des opérations afin d'accomplir des tâches bien précises. Les services doivent collaborent entre eux afin de réaliser l'objectif global de l'application. En ce sens, les services peuvent établir des partenariats, obligatoires ou optionnels (si le service n'est pas présent). Par exemple, une application souhaitant diriger un robot en le faisant éviter les obstacles devra avoir pour partenaires les services du moteur et du détecteur d'obstacle.

5.3.2.2 MODELE CCR

CCR (Concurrency and Coordination Runtime) assure la coordination et la synchronisation des services s'exécutant avec DSS. Il permet de s'abstraire des problèmes de la programmation multithread en robotique. Sous forme de librairie, CCR répond à ces besoins :

- Asynchronisme : les opérations appelées d'un service à un autre le sont grâce à des messages asynchrones. Cela permet qu'un service ne soit pas bloqué avant la réponse du service appelé, et ainsi d'assurer une exécution en parallèle.
- Concurrence : un service, bien qu'entité indépendante dans un nœud DSS, peut dialoguer avec plusieurs autres services pour s'échanger des ressources. En ce sens, il est important de gérer le fait que plusieurs services peuvent vouloir accéder à une même ressource en même temps.
- Coordination : CCR permet de coordonnées les différentes parties et sous-parties d'un système complexe et de gérer les éventuelles erreurs.

CCR propose donc des fonctions de haut niveau qui facilitent le développent de services en gérant les problèmes de threading à la place du programmeur.

5.3.2.3 MODELE DSS

Le modèle DSS (*Decentralized Software Services*) est l'environnement de support et d'exécution des services et gère la communication entre ces derniers via le réseau à travers le protocole ouvert DSSP (*Decentralized Software Services Protocol*). DSS expose de manière standardisée les services de son nœud d'exécution, qui peuvent alors être atteints par d'autres services qui sont soit du même nœud soit distants.

5.3.3 UTILISATION DE MSRDS DANS NOTRE PROJET

L'éditeur de scripts Lua à développer étant destiné à être utilisé dans un environnement robotique et couplé à des services, il nous a fallu appréhender les différentes problématiques et orienter notre architecture de façon à coupler au mieux notre solution avec Microsoft Robotics Developer Studio.

Notre travail a tout d'abord consisté à une prise en main de l'environnement. Pour cela, nous avons développé plusieurs services plus ou moins avancés afin de bien comprendre leur fonctionnement. Les nouvelles connaissances acquises, nous avons structuré notre application pour pouvoir l'intégrer facilement à un service.

L'éditeur de scripts s'est donc naturellement décomposé en 2 parties : l'interface graphique (et ses fonctions associées : auto-complétion, section d'aide, barre d'outils, etc.) et la section propre au langage Lua. En effet, nous avons fait en sorte de rendre le plus indépendant possible le composant « éditeur » et son utilisation de Lua, également dans le but que l'application soit évolutive une fois livrée à ROBOSOFT.

Dans ce cadre, nous avons adopté le design pattern « Façade » qui permet à une classe d'utiliser un autre ensemble de classes plus complexes à l'aide d'une classe (façade) simplifiée, jouant simplement le rôle de « tampon ». Toutes les parties de l'application ayant un lien quelconque avec le langage Lua (fonctions pour l'auto-complétion, vérification syntaxique des scripts, etc.) y accèdent grâce à cette façade.

Une interface C# a également été définie pour représenter un interpréteur Lua. Toute classe de l'interface graphique ayant besoin de ressources Lua fait donc appel à la façade, qui elle-même fait appel à une classe implémentant cette interface. En plus de rendre indépendants l'éditeur et une implémentation concrète de l'interpréteur (*LuaInterface* dans notre cas), cette interface permet, vis-à-vis de l'extérieur, de personnaliser le comportement et l'état de l'interpréteur Lua.

Quel rapport avec les services de Microsoft Robotics Developer Studio ? Un objectif : faciliter l'intégration de notre éditeur à toute application. Considérée comme un ensemble de librairies, notre solution pourra être importée dans de simples services (ou dans le middleware robuBOX). Le seul travail consistera à développer une implémentation de l'interface de l'interpréteur Lua, et en assurant le mapping des opérations de services partenaires sur des fonctions Lua.

Dans le cadre de notre application, nous avons développé un service ayant comme partenaires des services de motorisation (pour les déplacements) et de laser (pour détecter les alentours). Ce service utilise l'éditeur et permet efficacement de contrôler des robots avec des scripts écrits en Lua. Ce service a été utilisé dans un environnement de simulation par défaut de MSRDS, à défaut d'avoir un environnement viable de la part de ROBOSOFT.

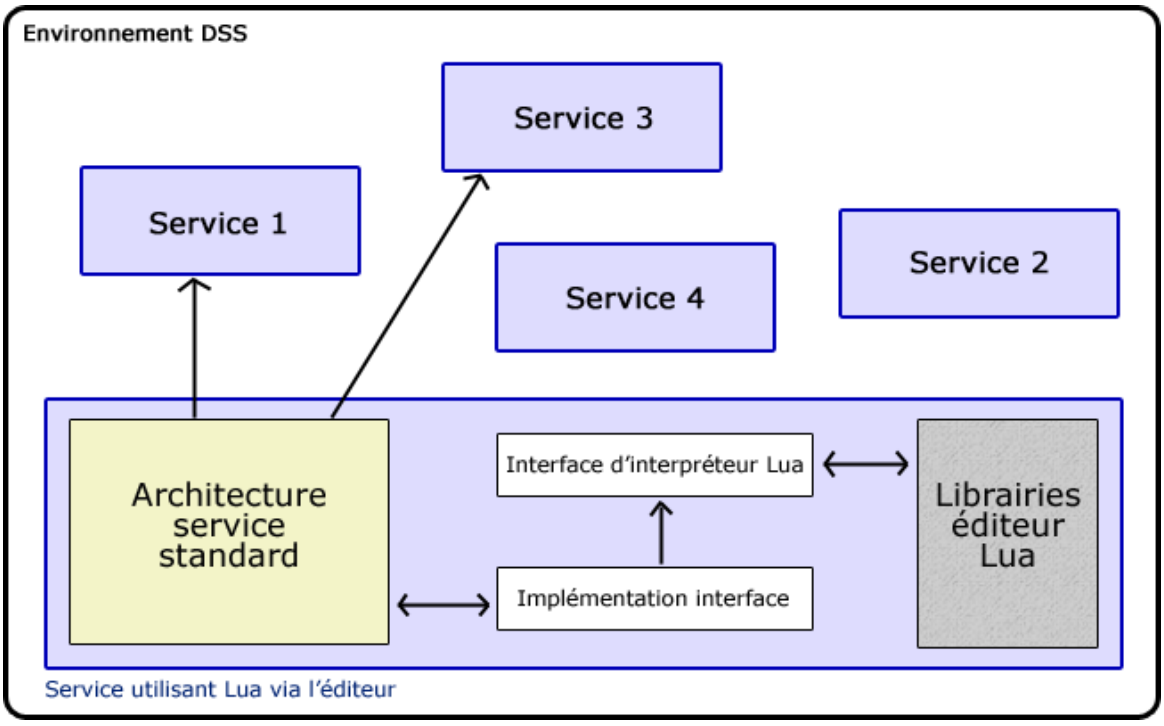

Figure 7 - Schéma simplifié d'un service utilisant l'éditeur de scripts Lua

5.4 ROBUBOX

La robuBOX est le middleware développé par ROBOSOFT qui équipe tous leurs robots et leur permet ainsi d'être pilotés. Lancée en 2006, la robuBOX est basée sur l'environnement Microsoft Robotics Studio (environnement de développement pour robotique sur Windows).

Figure 8 - Intégration de la robuBOX dans les solutions de ROBOSOFT

Cette solution permet des développements rapides et efficaces pour l'intégration de matériels et réseaux. De plus, grâce aux DSS (services) les robots et leurs logiciels liés peuvent être pilotés à distance. L'éditeur de scripts Lua sera intégré à cette solution.

Nous pouvons résumer au travers d'une citation du président de Rosobosoft (Mr. Vincent Dupourqué) l'interet de l'utilisation de cette Robubox :

« La démocratisation de la robotique nécessite à la base un environnement lui-même démocratisé et familier : Windows. En un an et en s'appuyant sur Visual Studio, .NET et Microsoft Robotics Studio, ROBOSOFT a créé la robuBOX™, permettant de contrôler aussi bien un robot simple que des flottes entières. La robuBOX™ intègre plus de 80 % de la complexité nécessaire à la mise en place des robots de services. En fournissant une plateforme commune de développement logiciel, Microsoft Robotics Studio permet à des sociétés comme ROBOSOFT de surmonter l'un des principaux obstacles sur le marché de la robotique : la fragmentation due à l'incompatibilité des plateformes disponibles. A l'occasion de RoboBusiness 2006, qui s'est tenu en juin à Pittsburgh, ROBOSOFT a présenté son robot tout terrain, le robuROC6, mettant en évidence tout le bien fondé de l'architecture construite à partir de la robuBOX™ et programmée avec Microsoft Robotics Studio ».

Vincent Dupourqué, Président et co-fondateur de ROBOSOFT.

5.5 LIBRAIRIES EXTERNES UTILISEES

Dans le cadre du développement de notre application, nous avons utilisé 2 librairies Open Source. La première est Alsing qui propose des fonctions avancées pour des éditeurs de code. Nous avons dû y apporter de nombreuses modifications pour le faire correspondre à nos besoins. Pour la mise

en place du langage Lua, nous avons utilisé la librairie LuaInterface qui est un interpréteur Lua destiné à des applications basées sur le framework .NET

Alsing : http://code.google.com/p/alsing/

LuaInterface : http://luaforge.net/projects/luainterface

6 FONCTIONNALITES DE L'APPLICATION REALISEE

6.1 MENUS & FONCTIONS ASSOCIEES

L'application propose plusieurs menus afin de permettre à l'Utilisateur de gerer ses scripts et l'exécution de ceux-ci.

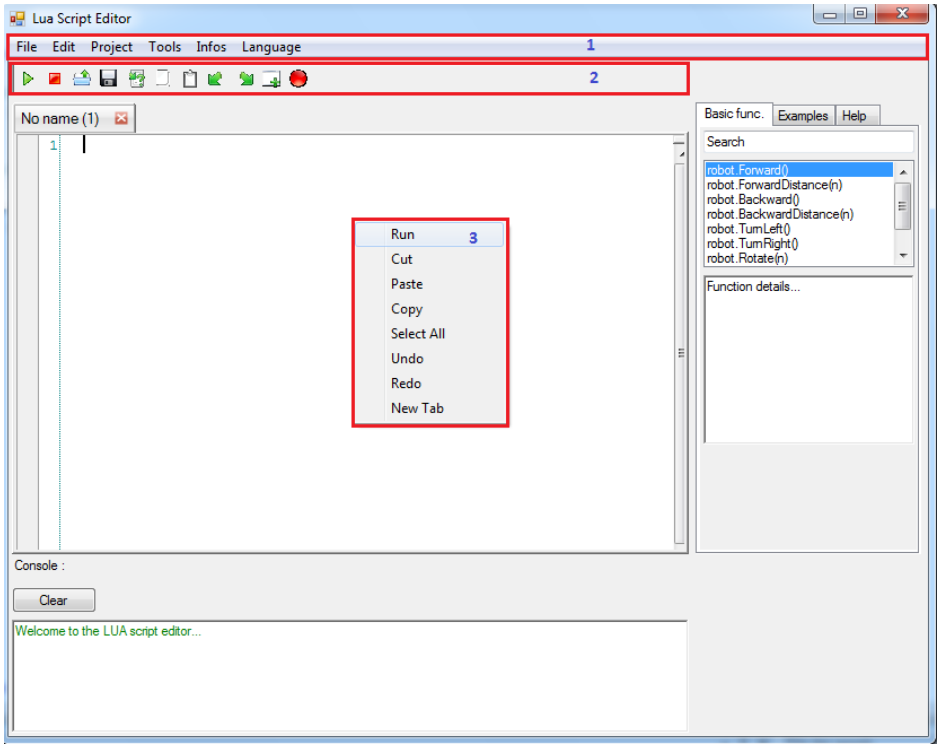

Figure 9- Menus de l'application

Il y a trois menus :

- Le menu General (numéro 1),
- La Toolbar (numéro 2),
- Le menu contextuel (numéro 3).

Les principales fonctions d'un éditeur de texte sont présentes :

- Copier / Coller / Couper,
- Ouvrir / Fermer un script,
- Nouvel Onglet,

- Enregistrer / Enregistrer Sous,
- Rechercher / Remplacer,
- Undo / Redo.

Des fonctions spécifiques à notre application ont également étaient mise en place :

- Exécuter le script,
- Stopper l'exécution,
- Vérifier le script (compiler),
- Créer une fonction.

Nous avons utilisé un pack d'icones pour la Toolbar afin d'obtenir un rendu visuel correcte pour l'utilisateur.

Les menus sont en interactions avec les Onglets, si aucun onglet n'est ouvert, certaines actions ne sont pas disponible comme par exemple :

- Exécuter le script,
- Vérifier le script,
- Fermer l'onglet,
- Enregistrer,
- …

Les fonctions des menus sont ainsi grisés et inutilisables.

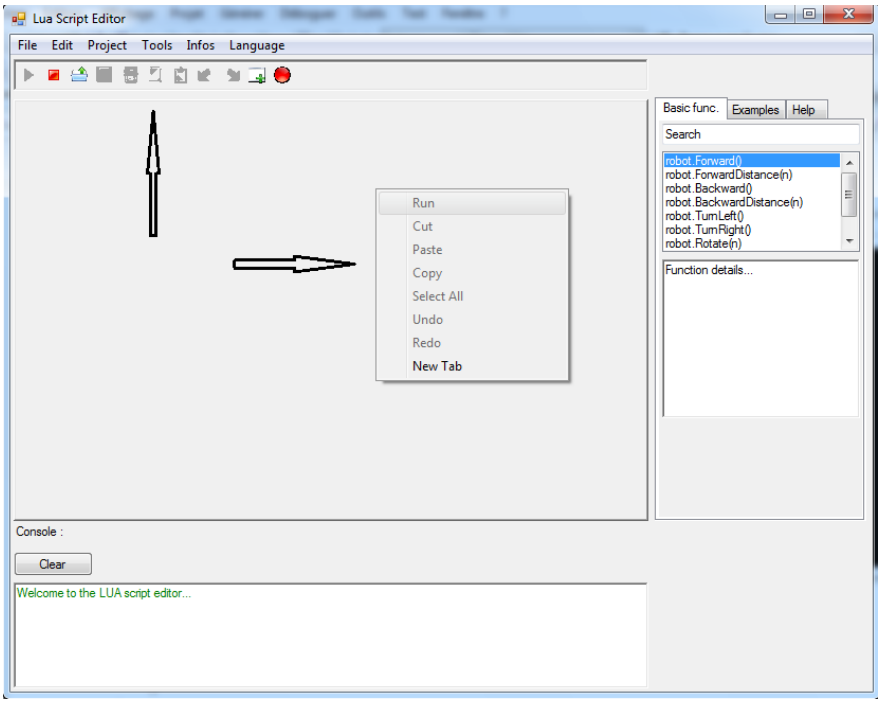

Figure 10- Fonctions inactives

6.2 MULTI FICHIERS

Afin de rendre notre éditeur plus fonctionnel, nous avons choisi d'implémenter la gestion multi-onglets. Cette fonctionnalité, présente dans tous les éditeurs récents permet à l'utilisateur une meilleure visualisation globale de son projet. Il est ainsi plus aisé de développer son programme.

L'éditeur permet de créer deux types d'onglets. Des onglets, interprétant le HTML, qui permettent d'afficher l'aide dans l'application et des onglets classiques qui ouvrent des éditeurs de code Lua. Pour les onglets de type éditeur nous avons du implémenter les fonctions concernées contenues dans le menu comme l'ouverture, l'enregistrement et la fermeture d'onglets.

Nous avons permit une bonne gestion des évènements liés à la souris avec l'intégration d'un bouton de fermeture sur chaque onglet ainsi que le clic mollette pour fermer un onglet.

Pour une meilleure ergonomie, chaque titre est rafraichit à chaque enregistrement. De plus, un astérisque apparaît après le titre lorsqu'un document est modifié et non enregistré. Lorsqu'un document est créé, il porte le nom « Untitle » suivit d'un numéro à la manière de notepad++.

6.3 AUTO-COMPLETION

Afin d'aider l'utilisateur dans l'écriture de ses scripts, l'éditeur propose la fonctionnalité d'auto-complétion. A l'aide du raccourci clavier Ctrl+Espace, une petite boîte de dialogue apparaît au niveau du curseur pour proposer à l'utilisateur les différentes fonctions et variables disponibles selon le contexte. Lorsque l'utilisateur sélectionne la fonction ou variable dans la liste proposée, la boîte de dialogue disparaît et le texte approprié est ajouté au script .

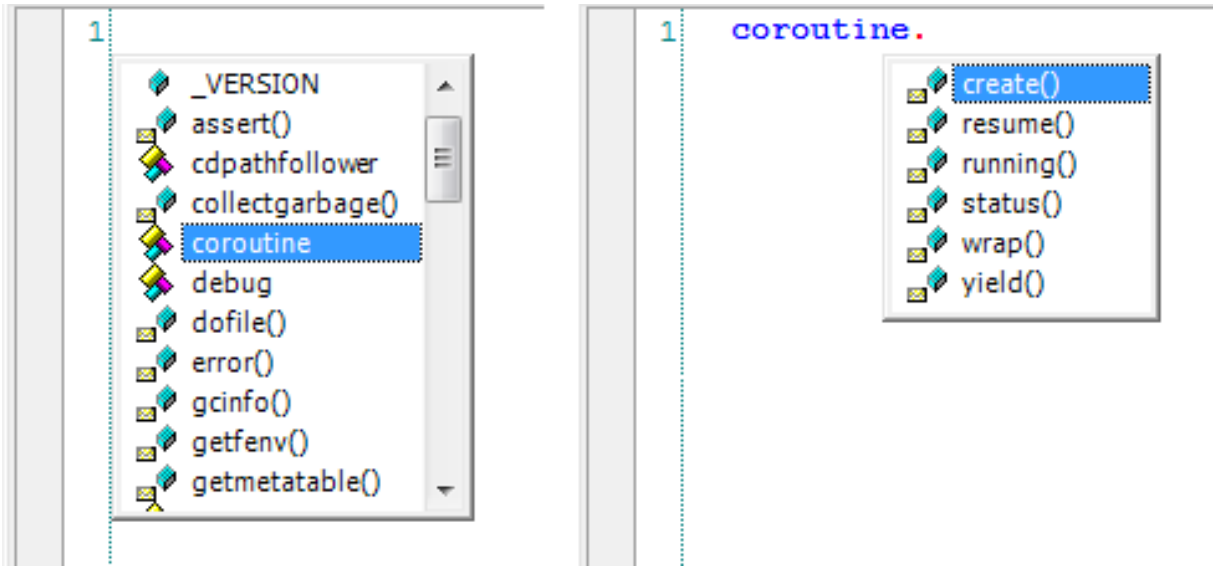

Figure 11 - Détection du contexte et hiérarchisation des fonctions

Le système gère les fonctions de façon dynamique en intégrant, aux fonctions de base de l'interpréteur Lua, les fonctions disponibles via le service lié au robot.

6.4 VERIFICATION SYNTAXIQUE

L'éditeur met en jeu une vérification syntaxique des scripts Lua, de façon automatique. Celle-ci se fait à la volée, 1 seconde après le dernier caractère écrit par l'utilisateur. La vérification se fait de façon totalement parallèle et transparente pour l'utilisateur, ce qui ne le bloque pas dans son développement. Cette fonctionnalité permet ainsi d'aider l'utilisateur dans l'écriture de ses scripts afin de vérifier qu'ils ne contiennent aucune erreur.

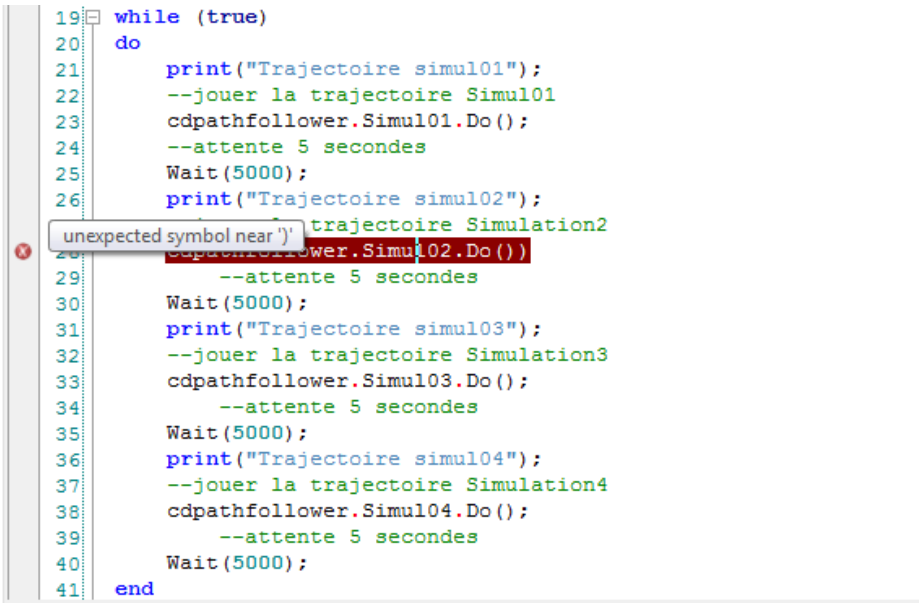

Figure 12 - Vérification syntaxique du code, avec explications des erreurs

En cas d'erreur repérée, l'éditeur le signale à travers une petite icône dans la marge, au niveau de la ligne posant problème et cette dernière est surlignée de rouge. Un survol de l'icône d'erreur affiche la raison exacte ayant provoqué cette erreur.

6.5 AIDE INTEGREE

Le logiciel développé possède une aide intégrée. Autrement-dit, une zone de l'interface permettant à l'utilisateur d'obtenir une aide à l'utilisation du logiciel.

Cette aide est divisée en deux sections :

- Une première qui est destinée à l'utilisation du logiciel : celle-ci permet à l'utilisateur de s'orienter dans l'application développée. Elle permet un guidage simple et rapide.
- Une seconde section est destinée à l'emploi du langage Lua au sein de l'application. Elle permet à l'utilisateur d'obtenir toutes les informations nécessaires quant à l'utilisation du langage Lua (conventions d'écriture, opérateurs, …).

Ces deux aides sont constituées de fichiers html qui sont parsées par le logiciel à la demande de l'utilisateur. Ainsi, des modifications peuvent y être apportées facilement par un administrateur afin de mettre à jour les aides fournis.

Lors des modifications, il est important de conserver la structure actuelle des fichiers HTML. Le parseur HTML se basant sur la structure native des fichiers d'aide.

Voici un aperçu de l'affichage d'une aide Lua :

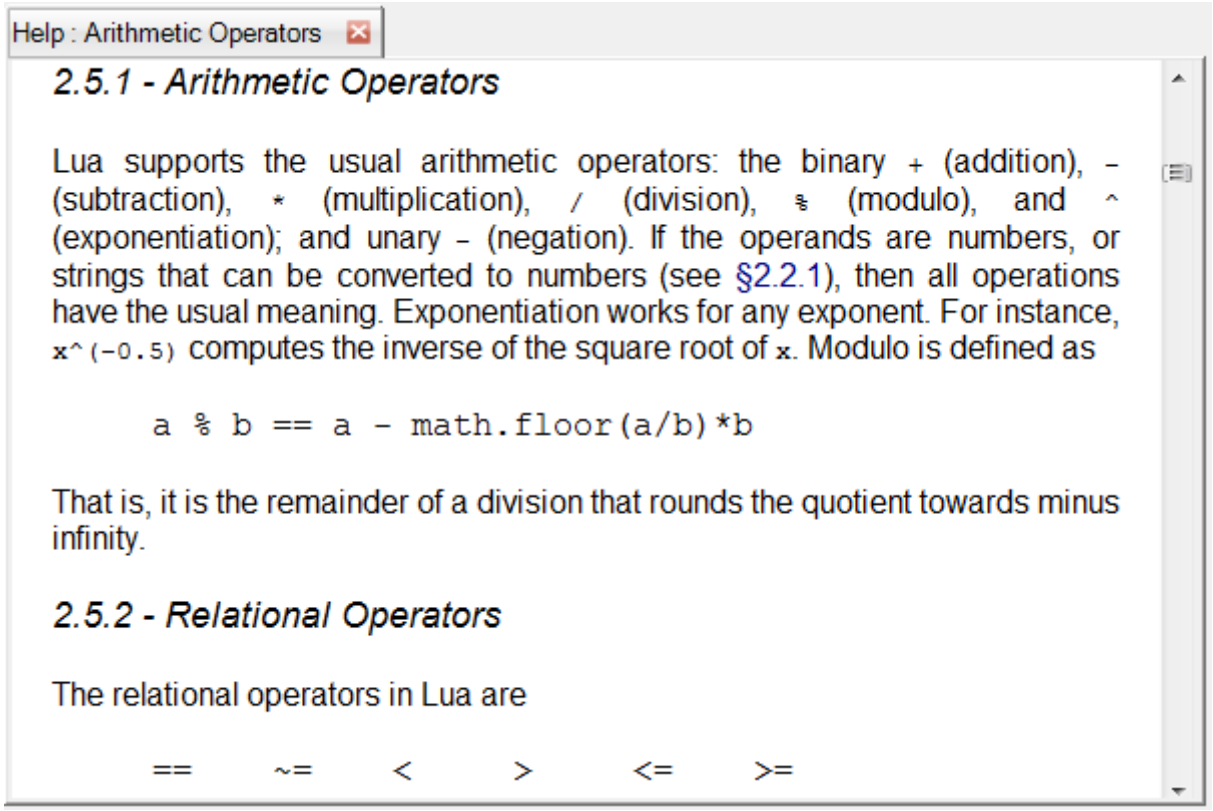

Figure 13 - Aide Lua

6.6 ASSISTANCE

En plus de l'aide normale fournie par l'application, une assistance est aussi disponible afin de rendre son utilisation encore plus simple et rapide.

Deux onglets permettent à l'utilisateur de choisir entre des fonctions de base qui seront spécifiées dans le code et des fonctions complètes que l'utilisateur aura la possibilité de tester directement dans le code source. En effet, un simple clique permettra d'inclure le code dans l'onglet courant ou encore d'ouvrir un nouvel onglet avec le code directement inclut. Ainsi l'utilisateur peut exécuter le code exemple, immédiatement.

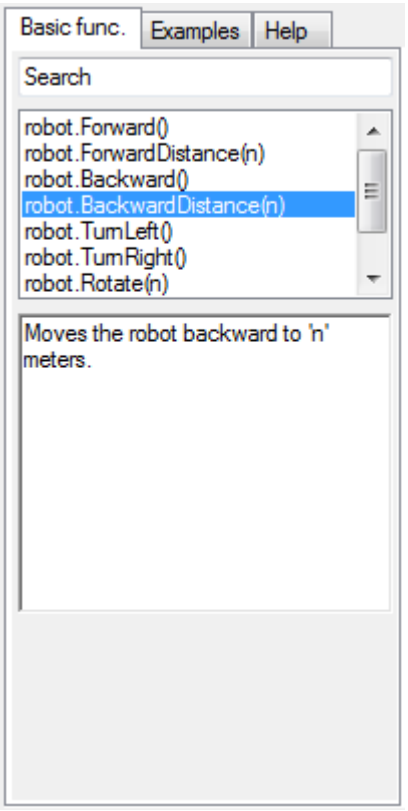

Figure 14 - Choix de fonction basique.

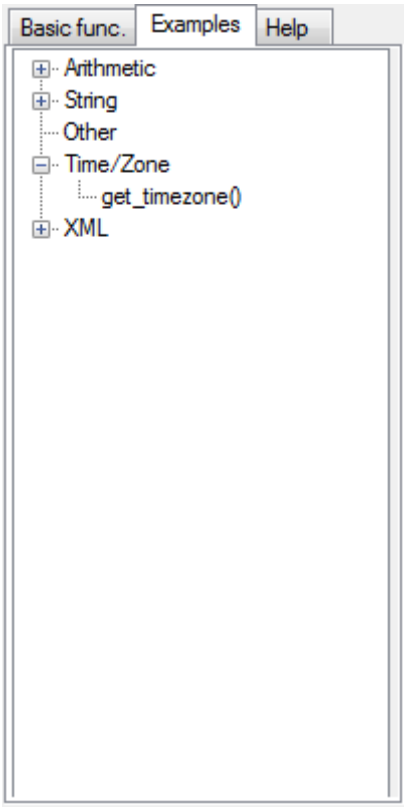

Figure 15 - Choix d'un exemple.

L'utilisateur à l'opportunité d'ajouter autant d'exemple par le biais d'un bouton dédié à cette fonction. Il lui suffira de spécifier le nom de la fonction, sa catégorie ainsi que le code.

Les catégories de fonctions sont éditables par le biais du menu Paramètres.

6.7 INTERNATIONALISATION

Il est important de distribuer un éditeur de texte sous plusieurs langues si l'on veut le rendre utilisable par le maximum de personnes possible. C'est pour cela que nous avons choisi de mettre en place l'internationalisation de l'éditeur de texte.

Pour mettre en place ce mécanisme ils nous a tout d'abord fallut créer des « Dictionnaires » (des fichiers de ressources utilisés par .Net) qui sont des fichiers au format XML.

Il nous a fallut créer un fichier par langue puis, dans chaque fichier, créer la correspondance entre une variable et un mot traduit.

Pour y voir plus clair, comparons un extrait des deux fichiers :

Dictionnaire.en-US.resx	Dictionnaire.fr-FR.resx

`<data name="MenuContextuel_Coller" xml:space="preserve">`
`  <value>Paste</value>`
`</data>`
`<data name="MenuContextuel_Copier" xml:space="preserve">`
`  <value>Copy</value>`
	`<data name="MenuContextuel_Coller" xml:space="preserve">`
	`  <value>Coller</value>`
	`</data>`
	`<data name="MenuContextuel_Copier" xml:space="preserve">`
	`  <value>Copier</value>`

```
  </data>
  <data    name="MenuContextuel_Couper"
xml:space="preserve">
    <value>Cut</value>
  </data>
```

```
  </data>
  <data    name="MenuContextuel_Couper"
xml:space="preserve">
    <value>Couper</value>
  </data>
```

Nous remarquons que les deux fichiers sont composés des mêmes variables (par exemple « **MenuContextuel_Coller** ») mais traduites dans deux langues differentes (par exemple « **Paste** » et « **Coller** »).

Ainsi, lorsque l'utilisateur choisi sa langue préférée, l'application va rechercher selon le nom du fichier les traductions à utiliser.

Pour effectuer ce processus, il faut tout d'abord définir une ResourceManager et deux CultureInfo.

Le ResourceManager sera en charge de récupérer les bonnes traductions selon la CultureInfo choisi (le choix de la langue par l'utilisateur définira la CultureInfo à sélectionner).

Nous avons donc deux CultureInfo (fr-FR et en-US). Si l'utilisateur sélectionne la langue Française, alors la CultureInfo fr-FR est sélectionnée et le ResourceManager va récupérer le dictionnaire ayant pour nom « Dictionnaire.fr-FR.resx ». Par la suite, les Labels, les Boutons etc… seront modifiés afin d'avoir un texte correspondant à la bonne langue sélectionnée.

Un exemple :

```
this.Text = m_ResourceManager.GetString("Application_Titre");
```

Ici, le titre de l'application sera défini en utilisant le ResourceManager qui ira lui-même sélectionner dans le bon Dictionnaire (soit le Français, soit l'Anglais) la variable ayant pour nom « Application_Titre »).

A l'heure actuelle, les langues Française et Anglaise sont disponibles mais l'on pourrait ajouter autant de langue que l'on souhaite, cela ne prendrai plus beaucoup de temps, le mécanisme étant déjà en place (il faudrait juste définir une nouvelle CultureInfo et décrire le nouveau Dictionnaire).

7 ORGANISATION DU PROJET

7.1 PHASES DU PROJET

Phase 1 : Prise de contact

- Constitution de l'équipe,
- Mise en place du SVN (Google Code),
- Mise en place Visual Studio,
- Discussions sur le wiki de ROBOSOFT,
- Prise en main du langage C#,
- Prise de connaissance détaillée du sujet,
- Elaboration d'un planning.

Phase 2 : Développement d'un prototype

- Adaptation d'une librairie open source pour l'éditeur (Alsing),
- Auto-indentation,

- Intégration d'un interpréteur Lua,
- Menus & Fonctions primaires,
- Prise en main du langage Lua,
- Exécution d'un script basique.

Phase 3 : Amélioration du prototype

- Menus : Utilisation d'icones / Correction de bugs,
- Menus : Ajout de fonctions avancées,
- Intégration de l'aide,
- Aide : Fonctions de bases & Exemples
- Ajout de la console de sortie,
- Auto-complétion,
- Gestion des onglets,
- Vérification de script à la volée,
- Mise en place de l'internationalisation.

Phase 4 : Intégration avec Microsoft Robotics Studio

- Prise en main de Microsoft Robotics Studio,
- Recherche de documentation sur l'architecture,
- Développement de services de bases,
- Interaction entre un service et l'éditeur,
- Intégration d'un prototype de robot de ROBOSOFT,
- Développement de services avancés,
- Contrôle de robots avec Lua.

Phase 5 : Finalisation

- Structuration de l'application avant livraison,
- Corrections de bugs,
- Derniers tests,
- Documentation.

7.2 CAHIER DES CHARGES

Lors du premier semestre notre mission était de s'imprégner du sujet, du contexte, des technologies à utiliser, de définir les fonctionnalités à implémenter et de réaliser un planning prévisionnel.

Ce document a donc constitué une base pour notre équipe lors de notre phase de développement de l'application au second semestre.

7.3 DISTRIBUTION DES TACHES

Les macro-tâches ont été distribuées lors de la réalisation du cahier des charges au premier semestre à partir des désirs de chacun.

Voici le tableau de répartition des tâches :

Menus & Fonctions associées	Christophe DUC
Barre d'onglets & Fonctions associées	Jonathan GOBBO et Laura VIKOR

Auto-complétion	Guilhem BELLION et Laura VIKOR
Validateur de scripts Lua	Guilhem BELLION
Intégration de l'interpréteur Lua	Guilhem BELLION
Gestionnaire d'aides	Thomas GRAVINA
Gestionnaire de fonctions de base	Thomas GRAVINA
Hiérarchie des fichiers et fonctions exemples	Thomas GRAVINA
Intégration de l'environnement de simulation	Ensemble du groupe
Internationalisation & Gestion des Dictionnaires	Christophe DUC

7.4 COMMUNICATION INTERNE

Dans le cadre de ce projet nous avons organisé des réunions au moins toutes les deux semaines afin de faire le point sur l'avancement du projet, définir les nouvelles tâches, résoudre les éventuels problèmes rencontrés par chacun et faire des choix en commun sur l'évolution du projet. L'horaire choisit étant celui proposé par la MIAGE.

Mis à part les réunions notre groupe a beaucoup communiqué par mails pour se tenir au courant de l'avancé de nos tâches respectives, des dates de réunions mais aussi pour nous alerter de différents bugs survenus ou des nouveaux besoins détectés.

7.5 COMMUNICATION AVEC L'ENTREPRISE

L'entreprise ROBOSOFT nous a mis à disposition un site de type wiki où l'on peut envoyer des messages, définir les tâches à accomplir mais aussi les assigner (TO-DOS) et déposer des fichiers. C'est par ce biais que le chef de projet a communiqué avec la société. Celui-ci a donc envoyé des mails pour poser des questions relatives au projet et pour tenir au courant la société de l'avancement de l'application. De plus il nous a été demandé par ROBOSOFT de poster des vidéos de démonstrations ainsi que des screenshots à plusieurs moments de la phase de développement.

8 BILAN DE L'APPLICATION

8.1 POINTS FORTS, POINTS FAIBLES

- Points forts :
  - Personnalisation du robot possible,
  - Interface utilisateur ergonomique par rapport à des éditeurs Lua existants,
  - Mini « Environnement de développement »,
  - Possibilité d'utiliser l'éditeur indépendamment ou de l'intégrer à une autre application.

- Points faibles :
  - Code source non optimisé (travail en parallèle nécessitant beaucoup de modifications et découverte de C#),
  - Pas assez paramétrable (choix de la coloration syntaxiques, etc…).

8.2 PERSPECTIVES D'AVENIR

L'application ayant été développée dans le cadre d'un projet de Master 1 MIAGE et en parallèle d'autres projets, il va de soit qu'elle n'est pas totalement aboutie et nécessiterait des modifications avant d'être distribuée.

De plus, de nombreuses améliorations et optimisations sont à prévoir pour rendre l'application encore plus fonctionnelle, robuste et sécurisée. Celle-ci étant sous licence LGPL, le développement devrait continuer au-delà de ce projet et pourra apporter une contribution à la communauté Lua. De nombreuses fonctionnalités supplémentaires pourraient être ajoutées comme la gestion d'import de librairies Lua directement dans l'éditeur, une gestion plus aboutie des projets, etc.

Cette application est bien sûr destinée à la société ROBOSOFT qui devrait l'intégrer dans ses prochaines versions de la robuBOX, avec l'espoir d'offrir à ses clients une interface alternative pour la gestion et la personnalisation des robots.

9 CONCLUSION

9.1 APPRENTISSAGE

Ce projet nous a permis de découvrir le langage de programmation C# et plus généralement d'avoir une première approche du framework .NET. Nous avons également dû nous familiariser avec l'environnement de travail Microsoft Visual Studio. Ce projet nous a également apporté une première expérience, certes minime, dans le monde de la robotique et de découvrir ses nombreuses possibilités.

Le résultat obtenue nous permet de nous rentre compte de l'importance et des avantages du langage de programmation Lua, et plus généralement des langages de scripts, dans des environnements de grande ampleur, de manière très adaptée et modulable.

9.2 PEDAGOGIQUE

Ce projet d'année nous a permis d'apprendre à gérer un projet de longue durée d'une part mais aussi un projet en collaboration avec une entreprise qui avait de réelles attentes d'autre part.

Nous avons donc réalisé l'importance d'une bonne attente et d'une bonne communication en interne, dans le groupe de développement, ainsi qu'en externe. De plus la rédaction de plusieurs documents (screenshot, vidéo, mails, mini-rapport), au fur et à mesure de l'avancement du projet, permettent de mieux gérer le planning des tâches à accomplir. Les réunions régulières ainsi qu'un travail personnel régulier nous a permis de finir dans les temps sans stresse final.

10 GLOSSAIRE

Auto-complétion : Un éditeur de texte fait de l'auto-complétion quand il remplace une abréviation par la forme complète d'une expression lorsque vous tapez cette abréviation suivie d'une combinaison de touches magique. Cela permet d'économiser énormément les frappes au clavier.

Automatisation : Ensemble de procédés qui rendent l'exécution d'une tâche automatique, sans intervention de l'homme.

CCR : Concurrency and Coordination Runtime. Permet de simplifier le developpement de services pour la plateforme Microsoft Robotics Developer Studio en faisant abstraction des problèmes de multithreading.

Checkout : Opération d'extraction d'une version d'un projet du repository vers un répertoire de travail local.

Commit : Opération de validation des modifications réalisées dans le répertoire de travail local et de propagation dans le repository d'où il est issu.

DSS : Decentralized Software Services. Environnement d'exécution des services de la plateforme Microsoft Robotics Developer Studio

EDI : Environnement de Développement Intégré. Ensemble de solutions logicielles pour faciliter

Fonction algorithmique : Fonction qui désigne l'ensemble des règles et des techniques qui sont impliquées dans la définition et la conception des objets.

Gestionnaire SVN : Logiciel de gestion de configuration permettant de stocker des informations pour une ou plusieurs ressources informatiques permettant de récupérer toutes les versions intermédiaires des ressources, ainsi que les différences entre les versions.

Hardware : Désigne tout ce qui fait partie du matériel, dans un système informatique. Opposé à software (logiciel).

Level design : c'est le processus dans la création de jeu vidéo qui s'occupe de la réalisation des niveaux (ou *cartes*) de jeu (*level* ou *map* en anglais).

Lua : langage de programmation de script, beaucoup utilisé de manière embarquée dans des environnements de grande ampleur.

Middleware : C'est une couche de logiciels qui crée un réseau d'échange d'informations entre différentes applications informatiques. Le réseau est mis en œuvre par l'utilisation d'une même technique d'échange d'informations dans toutes les applications impliquées1 à l'aide de composants logiciels.

Macro tache : Ensemble de taches.

MSRDS : Microsoft Robotics Developer Studio. Plateforme standardisée pour le développement de solutions liées à la robotique.

Open Source : s'applique aux logiciels dont la licence respecte des critères précisément établis par l'*Open Source Initiative*, c'est-à-dire la possibilité de libre redistribution, d'accès au code source et de travaux dérivés.

Repository : Dépôt de logiciels. Un répertoire dans lequel subversion stocke les informations nécessaires au contrôle de versions d'une série de projets.Robotisation : Désigne le fait de robotiser une tâche.

Robot : C'est est un dispositif mécatronique (alliant mécanique, électronique et informatique) accomplissant automatiquement soit des tâches qui sont généralement dangereuses, pénibles, répétitives ou impossibles pour les humains, soit des tâches plus simples mais en les réalisant mieux que ce que ferait un être humain.

Screenshot : Terme anglais correspondant à l'image obtenue suite à un enregistrement instantané de ce qui apparaît visuellement à l'écran, ceci à partir de la touche "Impr écran" accessible sur la majorité des claviers actuels.

Script : Programme informatique qui ne nécessite pas de compilation avant d'être exécuté. Pour fonctionner, les scripts doivent être interprétés par un programme ou un serveur dédié au langage dans lequel ils ont été écrits.

Wiki : Site web dont les pages sont modifiables par tout ou partie des visiteurs du site. Il permet ainsi l'écriture collaborative de documents.

11 ANNEXE

11.1 DIAGRAMME DE CLASSES