Tuteurs : Yann-Gaël Guéhéneuc et Foutse Khomh
Elève : Benedicto Pierre

Université de Montréal

Projet Ift3150

Analyseur syntaxique C# pour PADL

Table des matières

Introduction

La rétro conception est un domaine capital en informatique car elle permet d'analyser un programme en construisant des modèles génériques. Le projet PADL, faisant partie de la suite d'outils Ptidej, implémente un de ces modèles, et possède déjà plusieurs analyseurs syntaxiques permettant de traiter quelques langages de programmation.

Le but de ce projet est de créer le modèle PADL d'un programme C# en utilisant et modifiant un analyseur syntaxique pour ce langage.

Ce rapport expose l'énoncé du projet dans un premier temps, puis la présentation de la solution. On continuera par la présentation de la base existante pour aller vers l'implémentation. Enfin je présenterai les résultats obtenus et les possibilités d'évolution.

I. Énnoncé

Ce projet consiste en la création d'un analyseur syntaxique dans le but de construire le méta modèle PADL d'un programme C#. Cette partie présente l'énoncé du projet.

A. Bénéfices attendus

En choisissant ce projet, les étudiants apprendront à utiliser le dernier langage en date de Microsoft, C# et approfondiront leurs connaissances du langage de programmation Java et des techniques d'analyses syntaxiques et de méta modélisation. De plus, ils apprendront à utiliser de manière approfondie l'environnement Eclipse.
Ces connaissances apporteront aux étudiants un avantage indéniable sur le marché du travail ou dans la poursuite de leurs études car la demande pour des développeurs C# est de plus en plus grande et une bonne connaissance des langages C# et Java est très souvent demandé. De plus, Eclipse est devenue la plate-forme de développement de référence dans l'industrie (IBM WebSphere) et dans le milieu universitaire.
Du point de vue des responsables du projet, le travail réalisé permettra d'étendre les possibilités de la suite d'outils Ptidej et des modèles de systèmes logiciels rétro-conçus par son intermédiaire.

B. Contexte

La rétro-conception de systèmes logiciels consiste en l'analyse du code source et des autres sources d'information sur les systèmes et en leur modélisation sous une forme intéressante pour les développeurs ou mainteneurs. La suite d'outils Ptidej offre des capacités de rétro-conception avancées par l'intermédiaire de modèles abstraits des systèmes logiciels, facilement analysables et manipulables. Ptidej inclue pour l'instant des analyseurs syntaxiques pour Java, un sous-ensemble de C++, et le format de fichier intermédiaire AOL.

C. Travail demandé

Le travail consiste d'abord à étudier le langage de programmation C# et à se familiariser avec les analyseurs syntaxiques existants pour ce langage. Il s'agit ensuite de comprendre superficiellement le modèle de données de la suite d'outils Ptidej et d'étudier avec le responsable du projet sa création.

I. Énnoncé

Il s'agit enfin de proposer et d'implanter un analyseur syntaxique C# pour la suite d'outils Ptidej d'intégrer l'implantation avec l'interface usager de Ptidej.

D. Environnement de travail

Le travail sera exécuté par un groupe d'au plus quatre étudiants motivés et bons programmeurs, en Java, avec l'environnement de développement pour Java fourni avec la plate-forme Eclipse. Le travail sera interfacé avec la suite d'outils de rétro-conception Ptidej.

II. Présentation de la solution

Cette partie présente l'idée générale du projet permettant de développer une solution répondant au problème posé.

A. Rappel du problème

L'objectif du projet est de développer un logiciel permettant de créer le méta modèle PADL d'un programme écrit en C#.

B. Idée générale

L'idée est d'analyser syntaxiquement le code C# pour construire son arbre syntaxique. Ensuite, il suffit de parcourir cet arbre et de créer les objets PADL correspondants.

C. Les étapes du projet

Pour commencer, il faut se familiariser avec le langage C# et comprendre son fonctionnement global. Ensuite, on analyse la grammaire C# dans le but de repérer les différentes structures de programmation.

La seconde étape consiste en la découverte et l'étude du méta modèle PADL dans le but de repérer et d'analyser les concepts de programmation du modèle.

Ensuite, il suffit de faire la correspondance entre C# et PADL afin de trouver la représentation des concepts de programmation C# dans le modèle PADL.

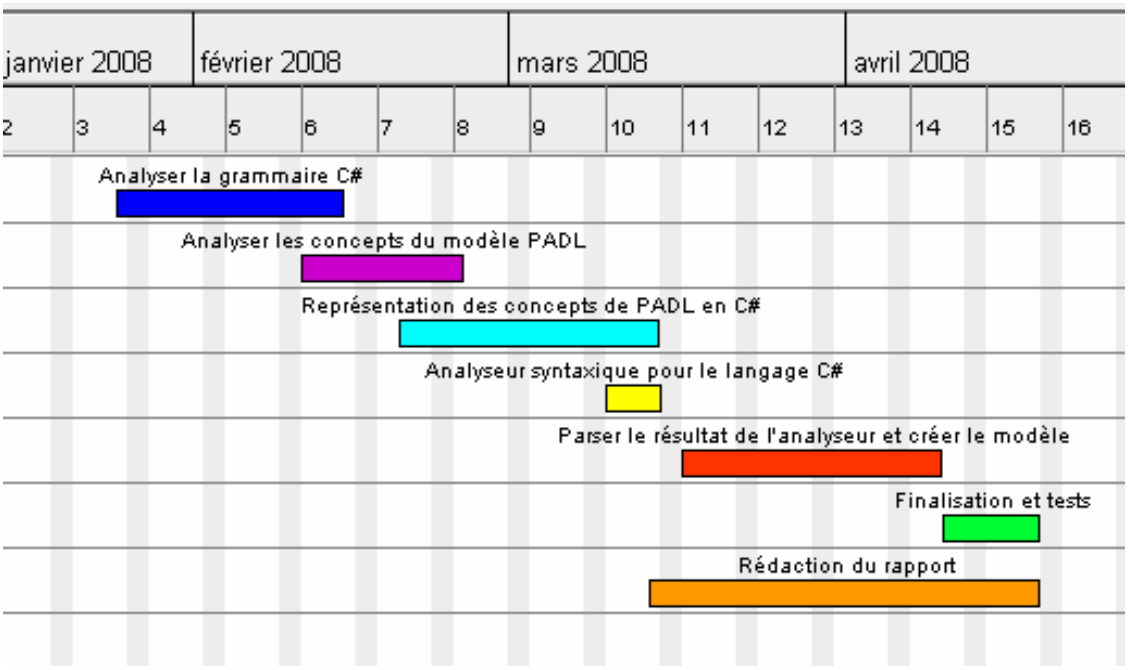
L'étape suivante comprend la recherche d'un analyseur syntaxique pour le langage C#. L'analyse de la sortie du programme et de son code sont les principaux critères de sélection.

Enfin, on parse le résultat de l'analyseur syntaxique en vue de créer les objets correspondants dans le méta modèle PADL.

D. Diagramme de Gantt

Le diagramme de Gantt ci-contre présente l'agencement des ces étapes dans le temps.

II. Présentation de la solution



III. Base existante

Cette partie a pour but d'exposer l'environnement de développement de ce projet. Dans un premier temps, je présenterai le méta modèle PADL, ensuite, je m'intéresserai au parser sélectionné et enfin aux logiciels utilisés pour mettre en place ma solution.

A. Le modèle PADL

Le méta-modèle PADL, faisant partie de la suite d'outils Ptidej, permet de donner une représentation générique d'un programme, quelque soit le langage dans lequel il a été écrit. Ce modèle permet par la suite de faire de la rétro conception, d'analyser un programme, de calculer des métriques, d'évaluer le couplage interclasse,…

Le projet PADL fournie les outils nécessaires à la création d'un modèle représenté par un objet *ICodeLevelModel*. Ensuite un objet *Factory* nous permet de créer les éléments dont ont a besoin tel que des *IClass, IAbstractMethod, IField*,…

B. Le Parser

Le rôle du parser est la construction de l'arbre syntaxique d'un programme C#. Nous avons décidé pour se projet d'utiliser un parser déjà existant.
Après quelques recherches, CSParser s'est démarqué grâce à son code clair qui facilite sa compréhension et sa modification.
Il a été développé à la base par Debreuil Digital Works en décembre 2006. Puis Denis Erchoff a ajouté le support de C# 2.0. Ce logiciel possède une licence BSD (Berkeley Software Distribution) qui est une licence libre permettant d'utiliser tout ou une partie du logiciel sans restriction.

L'exécution du parser se déroule en deux étapes. Tout d'abord, le programme parse le code et l'analyse afin de produire l'arbre syntaxique du programme.
Ensuite, un visiteur est appelé pour parcourir cet arbre. C'est cette partie du code qui va être modifiée dans ce projet.

C. Outil de développement

L'analyseur syntaxique CSParser étant écrit en C#, j'ai décidé de développer mon projet à l'aide de Microsoft Visual C# 2008 Express, logiciel

III. Base existante

gratuit dans le cadre de projets non commerciaux. Son utilisation nécessite une simple inscription auprès de Microsoft.

L'environnement Eclipse est aussi utilisé pour parcourir le projet PADL et pour tester le code de création du méta modèle produit par le programme C#. Enfin, le projet final sera écrit en Java, sous Eclipse.

IV. Implémentation

Cette partie présente l'implémentation détaillée du logiciel en s'intéressant d'abord aux liens entre PADL et C#, puis à la création d'objets simples, ensuite celle d'objets plus complexe, et enfin, nous établirons un tableau récapitulatif des correspondances entre les objets C# générés par l'analyseur et les objets du méta modèle PADL.

Ce projet est réalisé par modification du code de l'analyseur syntaxique C# CSParser. Cet analyseur parse le fichier C# et génère l'arbre syntaxique correspondant. Ensuite, un visiteur est mis à disposition pour parcourir cet arbre. C'est ce visiteur qui a été modifié pour arriver à produire le résultat attendu.
En effet, l'idée est de visiter l'arbre syntaxique, de créer les objets PADL correspondants et de les ajouter au modèle au fur et à mesure. Il faut générer le code Java permettant de créer ces objets dans le visiteur et l'écrire dans un fichier sur disque.

A. Liens entre PADL et C#

La première étape consiste à effectuer la correspondance entre les concepts de programmation C# et les objets PADL. Cette analyse a été faite par niveau d'abstraction, du plus élevé au moins élevé. Je me suis d'abord intéressé aux classes, puis à leurs attributs, aux méthodes, aux paramètres des méthodes, à l'héritage, aux invocations de méthode…

Exemple pour une classe :

Code C#
public class Voiture {}

Code Java correspondant
IEntity Voiture = Factory.getInstance().createClass(«Voiture »);

B. Objets simples

Pour une classe la correspondance est assez simple car l'analyse crée un objet ClassDeclaration. Il suffit donc d'écrire la ligne de code Java ci-dessus dans la méthode visitClassDeclaration().

Exemple pour un classe :

Visiteur C#

IV. Implémentation

```
public virtual object VisitClassDeclaration(ClassNode classDeclaration, object
data)
{
        writeLine_InFile("IEntity " + classDeclaration.Name.Identifier + " =
        Factory.getInstance().createClass(\"" + classDeclaration.Name.Identifier +
        "\");");
}
```

C. Objets complexes

Par contre, certains objets s'avèrent plus difficiles à créer car ils correspondent à plusieurs structures issues de l'analyse syntaxique. Leur construction nécessite donc la visite de plusieurs nœuds de l'arbre syntaxique. Le problème qui se pose alors est la sauvegarde des informations nécessaires à la création de l'objet en question tout au long de la visite des nœuds correspondants.
La solution choisie pour palier à ce problème est de passer un objet en paramètre aux sous-méthodes contenant les informations nécessaires.

Prenons un exemple, pour un attribut, il faut créer l'objet PADL *IField*. Pour créer cet objet, on a besoin du nom de l'attribut et de son type. Cependant on ne peut pas récupérer le type de l'attribut dans la méthode *visitFieldDeclaration()*. En effet, le type est visité dans une autre méthode, soit dans la méthode *VisitPredefinedTypeReference()*, soit dans la méthode *VisitTypeReference ()*.
L'opération consiste donc à créer un objet que l'on va passer en paramètre à la méthode charger de visiter le type.
Cet objet contient, le nom de la méthode d'origine, et des informations pertinentes pour la construction.
Dans ce cas, il contiendra « visitFieldDeclaration » ainsi que le nom de l'attribut que l'on est entrain de visiter.
Alors, dans la méthode *VisitPredefinedTypeReference ()* ou *VisitTypeReference ()*, on récupère l'objet passé en paramètre. Le premier champ nous permet de savoir dans quel contexte on se trouve, dans ce cas, la visite d'un attribut, puis on récupère dans le deuxième champ le nom de l'attribut. Maintenant que l'on a toutes les informations nécessaires, on peut créer l'objet IField correspondant à l'attribut.

Exemple d'IField :

```
public virtual object VisitFieldDeclaration(FieldNode fieldDeclaration, object data)
```

IV. Implémentation

```
{….
        string[] x = { "VisitFieldDeclaration", fieldDeclaration.Modifiers.ToString(),
       fieldDeclaration.Names[0].QualifiedIdentifier.ToString() };

        fieldDeclaration.Type.AcceptVisitor(this, x);

    …. }
```

Ici, le tableau de chaîne de caractère est l'objet passé en paramètre. Le premier champ contient le nom de la méthode appelante qui représente le contexte d'appel, à savoir, la visite d'un attribut. Le second champ correspond à l'accessibilité (*public*, *protected* ou *private*). Enfin le troisième champ contient le nom de l'attribut.

On peut également devoir faire référence à certain objets, par exemple lors de la création d'une invocation de méthode. Comme on ne peut pas sauvegarder les objets Java en C#, on doit aller les chercher directement dans le méta-modèle que l'on vient de créer, grâce notamment aux méthodes *getActorFromName()* et *getActorFromID().*

D. Tableau des correspondances

Concepts	**Objets visités**	**Objets PADL**
classe	ClassNode	IEntity, IClass, IInterface
attribut	FieldNode, PredefinedTypeNode, TypeNode	IField
methode	MethodNode, PredefinedTypeNode, TypeNode ParamDeclNode	IAbstractMethod, IParameter
Héritage	QualifiedIdentifierExpression	
Invocation de méthode	MemberAccessExpression, IdentifierExpression, InvocationExpression	IMethodInvocation

V. Résultats et évolutions

Cette partie présente en premier lieu l'état final du projet en listant l'ensemble des structures gérées par le logiciel, puis l'architecture de la solution, ensuite, on s'intéresse aux tests unitaires et aux possibilités d'évolutions pour finir.

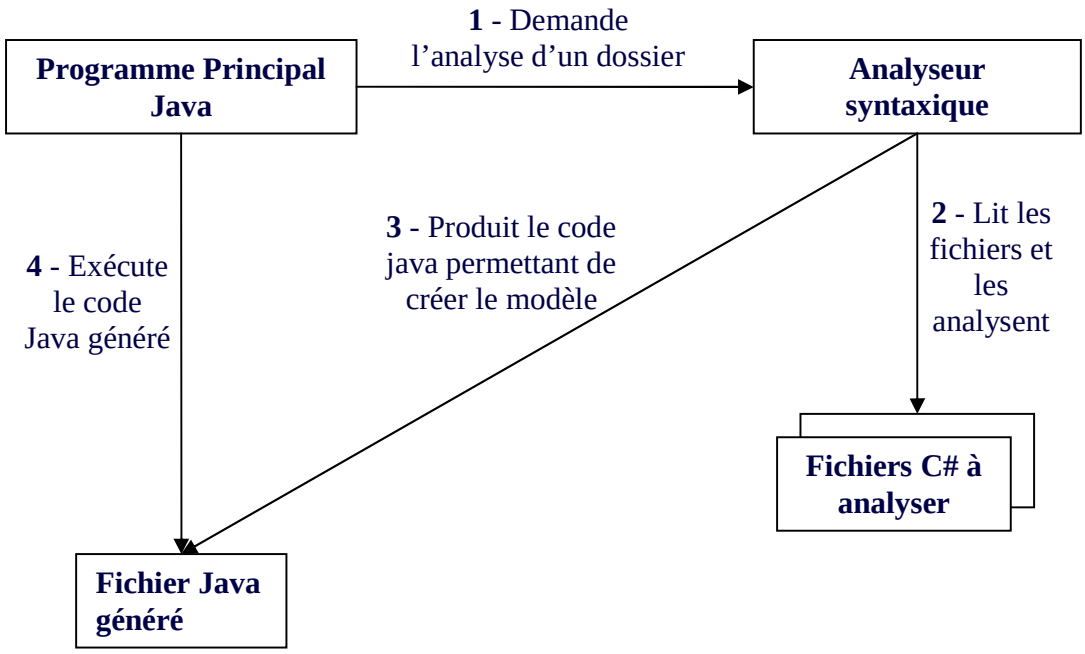
A. État d'avancement final

L'analyseur créé construit un méta modèle d'un niveau d'abstraction assez élevé, dans le sens où seule la structure du programme, les classes, et les liens entre ces classes sont représentés.

Liste des concepts de programmation traités :

- Les classes et interfaces
- Les attributs de classe
- Les méthodes
- L'héritage
- Les appels de méthode (ou invocation de méthode)

B. Architecture de la solution



V. Résultats et évolutions

Programme principal :

```
public static void main (String[] args)
{
if (args.length>0)
{
String directory = args[0];
String fileResult = "src\\main\\Test.java";
String exe = "src\\cmc.exe";
String classpath = "src\\;..\\PADL\\src;..\\CPL\\src;..\\CPL\\cfparse.jar;.";
String cmd = exe + " --test " + directory + " "+ fileResult;
Utils.StartCommand(cmd);
Utils.StartCommand("javac -classpath "+classpath+" "+fileResult);
Utils.StartCommand("java -classpath "+classpath+" main.Test ");
}
else
{
        System.out.println("miss directory parameter!!");
}
}
```

C. Tests unitaires

Quelques tests unitaires ont été mis en place afin de vérifier la correction du programme. Voici ci-dessous la liste des tests implémentés :

Tests compteurs

- nombre de classes et d'interfaces
- nombre de méthodes par classe ou interface
- nombre d'attributs par classe ou interface

Tests listage

- liste des attributs de chaque classe
- liste des méthodes de chaque classe

V. Résultats et évolutions

D. Évolutions possibles

Par manque de temps, certaines fonctionnalités n'ont pas pu être traitées, notamment l'analyse de projet complet. En effet, le programme obtenu permet d'analyser plusieurs fichiers, mais les priorités de création entre entité n'ont pas été gérées. En effet pour pouvoir créer une classe B qui hérite dune classe A, il faut que cette dernière est été crée au préalable.
Pour palier à ce problème, des indices sous forme de lettres ont été ajouté aux débuts des noms de fichier. Par exemple A_fichierx.cs, B_fichiery.cs,….

De plus, les invocations de méthodes ne sont gérées que pour des attributs, c'est-à-dire, que dans le cas ou un attribut appelle une certaine méthode d'un certain objet.
En effet, comme les variables n'ont pas été traitées, on ne pourra pas récupérer le type de la variable pour créer l'invocation.

De plus de nombreux autres tests pourrai être mis en place afin d'assurer la robustesse du programme.

Conclusion

Le bilan que je fais de ce projet est très positif. En effet, il m'a beaucoup apporté : j'ai découvert un nouveau langage de programmation, le C#. Le projet PADL m'a appris énormément sur la rétro-conception et l'analyse de programme qui est un domaine d'étude qui m'a beaucoup intéressé. De plus, l'étude du projet PADL m'a permis de me familiariser avec les techniques de développement de gros projets, comme l'utilisation des patrons de conception.

Le fait de travailler sur un véritable projet d'envergure est très motivant et permet de se rapprocher de situations plus professionnelles que scolaire. Même si le résultat obtenu n'est qu'une sous solution dans le sens ou elle nécessite quelques améliorations pour être effective, les objectifs fixés en début de projet ont été atteints.

Pour conclure, ce fut un plaisir de travailler sur ce projet car le domaine d'application est très intéressant car il regorge de possibilités, et l'équipe qui m'a accompagné fut fort sympathique et attentionnée.

Résumé

Ce rapport présente le projet développé dans le cadre du cours « projet informatique » de l'Université de Montréal. L'objectif est de créer un modèle générique, dans ce cas, le méta modèle PADL, d'un programme écrit en C#.
Ce modèle permet par la suite de faire de la rétro conception, d'analyser un programme, de calculer des métriques, d'évaluer le couplage interclasse,…
Pour construire ce modèle, plusieurs étapes sont nécessaires. Tout d'abord, il faut parser les fichiers C# afin d'en extraire la structure. Ensuite, on peut créer le modèle PADL correspondant.
Ce rapport présente les différentes étapes du projet ainsi que la solution obtenue.

Mots-clés : **Méta-modèle, PADL, Analyseur syntaxique, C#**

This report is about the project I have to develop for the computer science project course of University of Montreal. The objective is to create a generic model, in this case, the PADL Meta model, of a program written in C#.
This model enables do reverse engineering, analyze a program, calculate metrics, evaluate connections between classes,…
For build this model, several stages are necessary. First, we need to parse the C# files to extract their structures. After that, we can create the PADL model witch match.
This report exposes the differents project's stapes and the suggested solution.

Keywords : **Meta model, PADL, Syntax analyzer, C#**