Département Informatique
Année 2011/2012

Rapport de Synthèse

Création d'un Générateur de modèle PADL pour le langage C++

Sébastien Colladon

Laboratoire Ptidej de L'Ecole Polytechnique de Montréal
2900 Boulevard Edouard-Montpetit
Montréal, QC H3T 1J4,
Canada

Guy CAPLAT

Résumé
Etude de faisabilité, conception, développement, test et déploiement d'un générateur de modèle PADL pour le langage C++ avec la langage Java.

Mots clefs
Java, C++, Modèle objet, Patron de conception, Génie Logiciel, PADL.

Abstract
Analysis, design, development, test and deployment of a PADL generator for the C++ language with the Java language.

Key words
Java, C++, Object Model, Design Pattern, Software Engineering, PADL

Introduction

Ce PFE a été effectué du 4 janvier au 4 mai de l'année 2012 (4 mois pleins, ce qui fais 85 jours ouvrés) au sein du laboratoire Ptidej de l'Ecole Polytechnique de Montréal au Canada. Le projet qui m'a été confié était un projet qui avait été entrepris à quatre reprises auparavant dans ce laboratoire mais qui n'avait jusqu'à présent pas abouti. L'objectif de ce projet était de réaliser un générateur de modèle PADL pour le langage C++ avec le langage Java. Dans la suite du document nous allons voir un rapide descriptif de mon environnement de travail, puis nous verrons comment le projet a été réalisé.

Organisation

La planification du stage a été la suivante :

1. Choix du projet
2. Mise en place de l'environnement de développement, initialisation du projet
3. Spécification
4. Etude de faisabilité
5. Conception
6. Développement du projet
7. Test du projet
8. Deploiement du projet

La phase de développement s'est déroulée selon la méthode incrémental. Il était prévu qu'a la fin du PFE le projet soit encore en phase de développement mais qu'il soit à un stade suffisamment avancé pour pouvoir l'utiliser avec des programmes simples en ayant une représentation assez juste du programme analysé.

Ce projet était supervisé par Mr Yann Gaël GUEHENEUC (maître de stage), le développeur de PADL et responsable du laboratoire. J'ai été amené à avoir plusieurs entretiens avec d'autres personnes du laboratoire afin d'affiner mes connaissances sur le fonctionnement de PADL. Mon maître de stage et moi essayions de nous voir une fois par semaine au minimum afin de contrôler l'avancement et les orientations du projet. Ces réunions avaient lieu généralement le mardi.

Projet

I Contexte

Le laboratoire Ptidej analyse les programmes afin de trouver des principes ou des lois qui se répètent et ainsi poser les fondements de la compréhension d'un programme. Pour ce faire, Yann-Gaël GUEHENEUC a créé un méta modèle permettant de représenter un programme développé avec un langage objet. Ce méta modèle a été développé en Java et fourni une représentation statique du code (comme les patrons de conception du GoF). Au début, celui-ci était fais principalement pour le langage java (et ses conventions de nommages). Il existait à mon arrivée un seul parser qui était pour le langage Java, à partir du byte code généré. Il y a actuellement un parser en développement pour le Java à partir des fichiers sources utilisant l'AST de Eclipse JDT et pour le C# à partir des sources.

Pour pouvoir analyser toujours plus de programmes et faire des comparaisons entre les langages, les développeurs, le laboratoire a besoin de pouvoir analyser d'autres langages. C'est pourquoi il a été décidé que je travaille sur le langage C++. En effet, le langage C++ représente une mine de programmes open source analysables et donc un énorme potentiel de découverte pour le laboratoire.

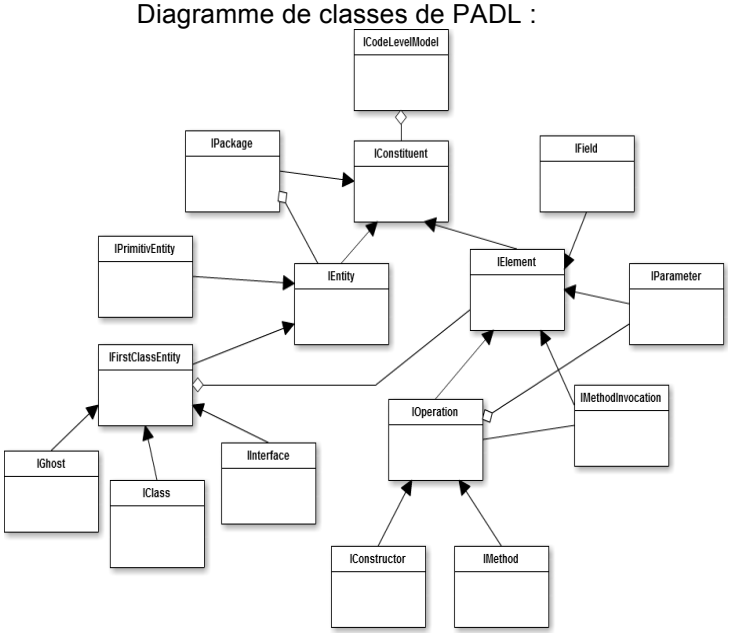
De plus, générer le modèle PADL pour les programmes écris en C++ représente un projet commencé en 2005. Ce projet a déjà été tenté par 4 équipes de travail différentes (dont mon maître de stage) sans jamais aboutir à une version exploitable. Etant donné que j'avais le choix entre effectuer un générateur de modèle PADL pour le langage C++ et un autre pour le langage C# (langage proche du java), j'ai opté pour ce qui selon moi représentait un challenge et ait choisi d'effectuer le générateur pour le langage C++.

De plus, créer un générateur de modèle PADL pour le langage C++ était en fait la continuité d'un cours que j'avais pris au premier semestre intitulé « Patron de conception pour la compréhension de programme ». En effet, nous devions lors des travaux pratiques, réaliser un générateur de modèle PADL à partir d'un fichier XML généré avec outil (chaque groupe de travail utilisait un outil différent) qui prend en entrée un fichier source C++. La partie « Etude de faisabilité » de ce document explique pourquoi je n'ai pas continué l'un de ces projets.

II PADL

PADL est un méta modèle de représentation d'un logiciel programmé avec un langage orienté objet. PADL est utilisé pour calculer certaines métriques sur un logiciel, détecter les motifs de patron de conception. PADL est à la base de beaucoup d'articles de génie logiciel.

Diagramme de classes de PADL :



Le ICodeLevelModel est la classe conteneur de tout un programme, il contient les IPrimitiveEntity qui représentent les types primitifs d'un langage (ils sont stocké directement dans le code level model pour éviter de les dupliquer dans les packages où ils sont rencontrés en paramètres de méthode ou en attribut d'une classe) et les IPackage (représente les package en java, les namespace en C++) qui contiennent les IClass (classe en langage objet), les IInterface (les interfaces en java, les classes virtuelles pures en C++) et les IGhost. Un IGhost est une classe dont nous n'avons pas eu la déclaration, elle provient généralement d'une librairie pré-compilé. Ensuite les IField représentent les attributs d'une classe, les IParameter représentent les paramètres de méthodes et les IOperations représentent les méthodes (et constructeurs) d'un programme. Ensuite viennent les invocations de méthodes qui représentent les différents appels de méthode effectuée dans une méthode (essentiel pour la détection de patron de conception).

III Expression du besoin

Cette application a pour but de créer le modèle PADL correspondant à n'importe quel logiciel écrit en C++. L'application doit permettre de récupérer à la fin du traitement le modèle PADL créé. Les utilisateurs seront essentiellement les personnes travaillant dans le laboratoire Ptidej.

La portée du projet est assez grande car elle concerne le laboratoire d'accueil. Le développement doit être fait par le réalisateur de PFE.

Trois critères d'appréciations majeurs découlent directement de l'expression des besoins :

- Portabilité
- Maintenabilité
- Facilité de déploiement

L'application ne doit pas disposer d'interface graphique, l'application doit juste permettre de récupérer le modèle PADL.

IV Etude de faisabilité

Dans cette étude de faisabilité, nous nous sommes intéressés à peu près à tous les outils permettant de représenter un programme C++. Nous nous sommes donc intéressés aux générateurs XML et aux parser fournissant un AST. L'objectif est de trouver une solution facilement déployable sur n'importe quel type d'ordinateur, sans contraintes de système d'exploitation, ayant une compatibilité ascendante et étant suffisament souple, documenté pour permettre à quelqu'un d'autre de reprendre le développement.

Tableau de comparaison des outils par rapport aux critères d'appréciations découverts lors de l'expression des besoins :

	Portabilité	Maintenabilité	Déploiement
CKDM	0	- (beaucoup de code juste pour le parsing. Communauté de développeur active)	- (nécessite GCC)
CML	+	- (beaucoup de code juste pour le parsing. Communauté de développeur peu active)	+ (nécessite un script d'appel de l'outil en fonction de la plateforme)
CXML	0	- (beaucoup de code juste pour le parsing. Communauté de développeur peu active)	- (nécessite GCC)
xt	? (outil non approprié)	? (outil non approprié)	? (outil non approprié)
TLR	+	- (nécessite de comprendre la grammaire et le parseur. Communauté de développeur active)	++ (code Java)
vaCC	++	- (nécessite de comprendre la grammaire et le parseur. Communauté de développeur active)	++ (code Java)
tbeans	? (inutilisable)	? (inutilisable)	? (inutilisable)
ipse T	++	+ (pas de paramétrage, bonne documentation mais peu d'exemple d'utilisation de l'AST. Communauté de développeur active)	+ (code Java mais indécouplable de la plateforme Eclipse)

Suite à cette étude, nous avons décidé d'utiliser l'outil Eclipse avec le plugin CDT.

V Spécification

L'application est complètement autonome, elle n'intéragis qu'avec l'utilisateur pour son lancement et avec le système.

L'application utilise Java pour s'exécuter, Java doit être installée dans l'environnement de déploiement.

L'application utilise en entré des fichiers sources et header C++.

L'application fournie en sorti un modèle PADL (un objet ICodeLevelModel).

L'application ne tiendra pas compte des erreurs présentes dans les fichiers fournis en entré.

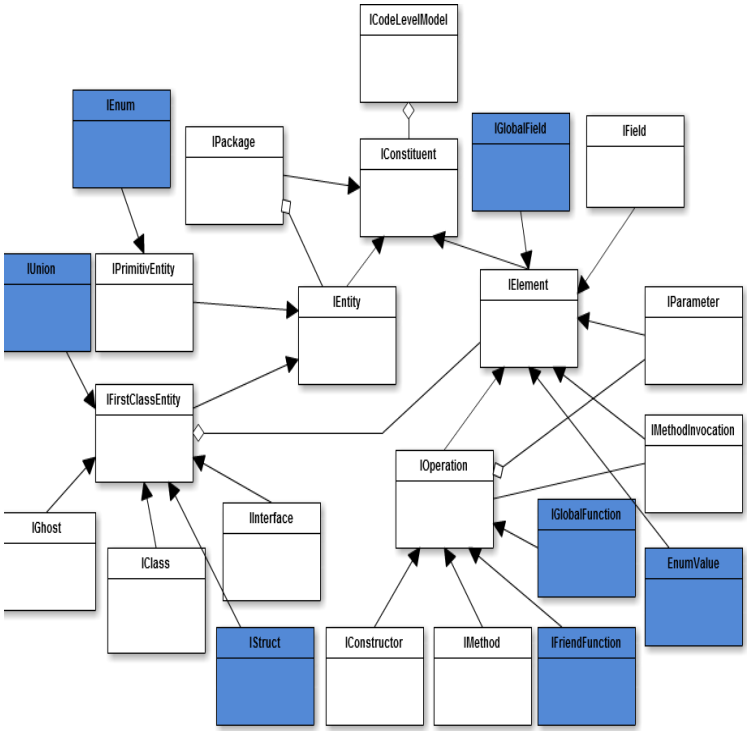
L'application utilise Eclipse CDT pour parser les fichiers sources. L'application utilise l'API de PADL pour créer le modèle PADL.

Le développement doit s'effectuer avec Eclipse. La gestion des versions est effectué avec SVN sur le serveur du laboratoire.

VI Conception

Afin de pouvoir représenter le C++, il a fallu ajouter certaines classes au méta modèle PADL. En effet, le C++ est beaucoup plus souple que le langage java et permet une plus grande représentation des données. Les classes ajoutées héritent des classes de base ce qui n'ajoute pas d'erreur lors de l'analyse du modèle PADL vue que ces classes sont traitées comme leur classe mère. Par contre, les classes permettent une évolution, un sous-classement des analyseurs existants, afin de les rendres plus performant pour le C++.

Diagramme de classes de PADL modifié :

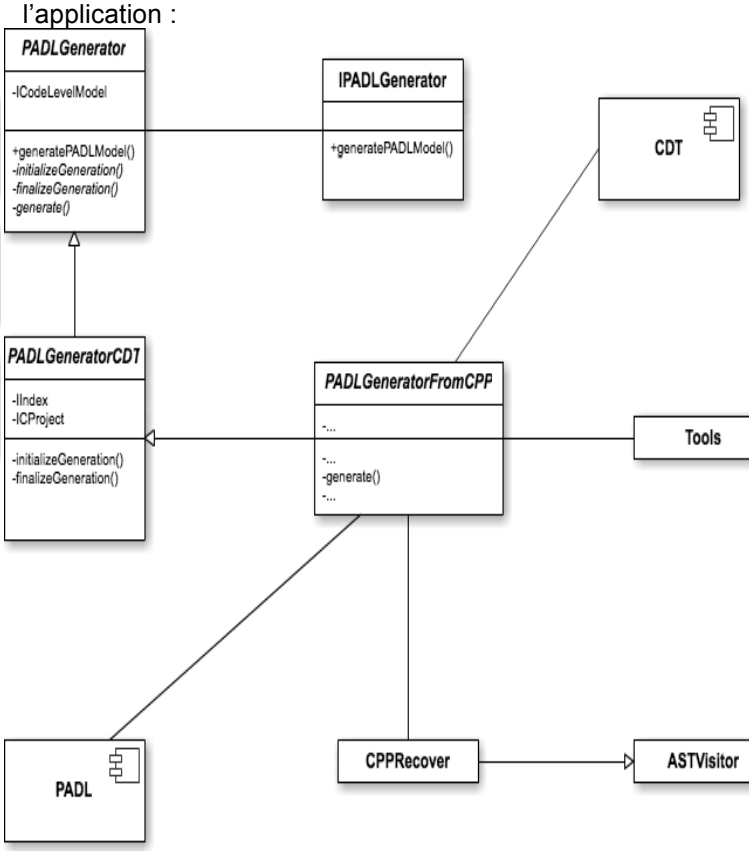


Une fois ces modifications apportées, il faut récupérer les informations qui nous intéressent avec Eclipse CDT pour pouvoir instancier le modèle PADL représentant le programme. Pour ce faire, l'architecture plugin d'Eclipse a été choisie. Ce choix offre une grande souplesse et assure une inclusion des dépendances efficaces. De plus, il n'est pas possible de découpler le parser C++ du CDT de la plateforme d'Eclipse car ce parser est très couplé à la notion de projet d'eclipse. Le parser doit être lancé sur les sources d'un projet Eclipse de type C++. Or, il n'est pas possible de créer ou de récupérer un projet Eclipse de type C++ sans lancer la plateforme Eclipse.

Parmi la liste des extensions réalisables, une extension qui n'utilise pas de système graphique a été choisie car l'application ne nécessite pas d'interface graphique. Nous avons opté pour une extension de type Application. Cette extension a l'avantage de lancer une nouvelle application Eclipse paramétrée selon nos besoins. Ce qui permet de lancer le traitement directement au démarrage de l'application Eclipse. Cela permet également de lancer Eclipse en mode Headless ce qui signifie que Eclipse ne charge pas l'interface graphique. Cela permet de gagner énormément de temps au lancement de l'application et de faciliter les tests ainsi que le déploiement.

L'application ne possède qu'un seul cas d'utilisation, c'est celui du lancement de l'application par l'utilisateur afin de récupérer le modèle PADL correspondant au programme à analyser. L'application ne s'occupe pas de la récupération des données à analyser. C'est l'utilisateur qui doit déposer les fichiers dans le projet C++ du workspace utilisé par l'application.

Voici le modèle structurel de l'application :



Toutes les classes hormis les composants PADL et CDT ainsi que la classe ASTVisitor ont été créé pour la réalisation de ce PFE. Un signe plus devant une méthode signifie que cette méthode est publique. Un signe moins devant une méthode ou un attribut signifie une portée protégée (transmise à l'héritage). Les classes ou les méthodes qui ont leur nom en italique signifie que c'est un élément abstrait. IPADLGenerator est une interface qui décrit le protocole pour créer un modèle PADL de manière générale. Pour créer un modèle il est obligatoire d'utilise un objet de type IPADLGenerator.

La classe abstraite PADLGenerator spécifie le comportement de la méthode generatePADLModel qui consiste en l'initialisation de la génération, la génération du modèle PADL et la finalisation de la génération.
La classe abstraite PADLGeneratorCDT spécifie l'initialisation et la finalisation pour une généralisation à partir du plugin CDT. L'initialisation s'occupe de rafraichir le contenu du projet cible, de réserver l'accès à l'index de l'AST et de rafraichir l'index de l'AST. La

finalisation s'occupe de rendre l'accès à l'index possible pour les autres processus.

La classe concrète PADLGeneratorFromCpp spécifie le comportement la fonction de génération dont l'algorithme est décris plus bas. Cette classe utilise une classe statique nommée Tools permettant de faire certaine opération sur des objets de l'AST afin de récupérer des informations. Elle utilise également un visitor pour l'AST afin de récupérer les déclarations de classe, union et structures. Elle utilise également directement l'API de CDT pour pouvoir parcourir les autres déclarations et connaître tous les types nécessaires à la récupération des informations pour créer le modèle PADL. Enfin, PADL est utilisé pour créer le modèle PADL du logiciel C++ cible.

VII Developpement

Le développement de ce projet s'est réalisé de manière incrémentale. C'est à dire que chaque fonctionnalités ont été ajoutées séparément au projet afin d'être sur de ne rien casser.

Afin de faciliter le développement, nous avons décidé de commencer le développement par tous ce qui attrait à la gestion des projets dans Eclipse. L'application possède un projet C++ par défaut dans lequel il faut mettre les sources du projet dont l'on souhaite obtenir le modèle PADL. Au lancement de l'application, le programme rafraichi ce projet afin de pouvoir tratier le programme voulu car un projet C++ Eclipse garde en mémoire des références vers les fichiers sources qui étaient précédemment présents à la précédente fermeture du projet Eclipse.

Le programme construit ensuite l'AST de chaque fichier source et l'index correspondant à chaque déclaration. Le programme parcours ensuite pour chaque Translation Unit, les définitions de classes, de structure, d'union, d'énumération, de fonctions globales et de variable globale.

Une fois cela effectué, nous récupérons pour chaque classe les attributs, les méthodes, les constructeurs et les destructeurs. Il faut faire cela après les déclarations de classes car si une classe contient un attribut ou champ de méthode qui est une classe que nous n'avons pas encore parsé, cela devient difficile d'obtenir les informations car il faut maintenir un cache des références à renseigner.

Ensuite nous renseignons les héritages et les relations d'amitiés aux niveaux des classes et aux niveaux des fonctions.

Une fois cela effectué, nous pouvons renseigner les invocations de méthodes dans chaque méthode. Pour faire ceci, nous maintenons un dictionnaire des méthodes PADL avec leur équivalence dans l'AST. Pour chaque méthode nous récupérons les appels de méthodes et nous vérifions si nous ne possédons pas cette méthode dans PADL. Si c'est le cas, nous ajoutons cette invocation de méthode dans la méthode courante.

VIII Test

Afin de tester l'application, des tests de non régression ont été réalisés au niveau des fonctionnalités permettant de s'assurer que lors d'un ajout de fonctionnalité, cet ajout n'avait pas cassé le fonctionnement validé à l'ajout de fonctionnalité précédent. Ces tests de non régressions consistaient en la création d'une classe C++ et de créer le modèle PADL correspondant (à la main) pour comparer ensuite le résultat de la génération à partir des sources.

Pour tester les fonctions d'outils permettant de récupérer des informations de l'AST ou de l'index, nous créions un test permettant de s'assurer que la fonctionnalité était bien remplie. Nous avons également effectué des tests de recouvrement (boite blanche) afin de s'assurer que certaines parties du code étaient bien atteintes lorsqu'elles le devaient.

Enfin, pour tester le modèle PADL créer, j'ai du créé un projet C++ contenant un programme pour chaque patron de conception du GoF. Une fois le modèle PADL généré, je passais celui-ci aux détecteurs de patron de conception afin de m'assurer que mon programme fonctionne correctement. Cela m'a beaucoup aidé pour détecter les fonctionnalités manquantes. Cette partie n'était pas facile. Il aurait été plus simple de générer le modèle PADL puis de générer à partir de celui-ci le code source et comparer les deux fichiers. Seulement PADL ne permet pas de créer de code source à partir d'un modèle PADL.

IX Déploiement

Pour faciliter le déploiement, nous avons créé une petite procédure. Cette procédure consiste en l'export du plugin application en incluant toutes ses dépendances (ce qui étonnamment n'étais pas si lourd que ça : 500ko

environs). Ensuite il faut placer ce plugin dans le répertoire plugin de l'installation eclipse cible. Ensuite il faut créer un projet C++ vide avec comme nom « C++ » dans le workspace par défaut de l'installation d'Eclipse cible. Ensuite il faut de lancer une commande java qui lance le lanceur de Eclipse avec comme application de base le plugin développé. Cette commande dépend du laucher. Si le plugin se trouve dans le bon répertoire (plugin de Eclipse) et que Eclipse possède une installation de CDT, le programme est alors efficacement déployé.

X Poursuite

L'application est stable et utilisable. Elle permet d'avoir un modèle PADL permettant de retrouver la plupart des patrons de conception contenu dans l'application source. Il faut maintenant adapter un peu plus PADL au langage C++ afin de permettre à PADL de tirer des informations des relations d'amitié, de l'utilisation de fonctions globales et de variables globales.

XI Difficultés rencontrées

Enormément de difficultés ont été rencontrées pour comprendre comment fonctionne l'API de Eclipse CDT car il n'y aucune documentation claire et précise expliquant comment utiliser cette API. L'API a subi d'énormes évolutions, ce qui fait qu'il est possible de trouver des tutoriels sur internet mais il n'y en a aucun qui est viable pour la dernière version de CDT.

Beaucoup de difficultés ont été rencontrées avec PADL, il a fallu modifier son code source à plusieurs reprises afin de rendre son architecture plus souple et modulaire.

Conclusions

Ce projet de fin d'étude a été pour moi une expérience très intéressante. Cela m'a permis d'allier plusieurs aspirations parmi lesquelles travailler en milieux anglophone, travailler dans le milieu de la recherche.

Le projet est actuellement réalisé, fonctionnel. Il permet de créer un modèle PADL à partir de source C++ en utilisant l'AST fourni par la plateforme Eclipse CDT. De plus, les fonctionnalités ajoutées à PADL pour représenter correctement le C++ permettent une plus grande souplesse du modèle et une évolutivité des outils du laboratoire utilisant un modèle PADL.

Le fait d'avoir réussi à mener à bien un projet qui n'avait pas abouti autant de fois auparavant est pour moi quelque chose dont je suis fier, gage de savoir faire et me permet par la même occasion de représenter correctement l'INSA de Lyon, mon école, à l'étranger.

Grâce à ce projet, J'ai énormément appris sur les patrons de conception ce qui m'a permis de perfectionner mes connaissances en architecture logiciel.

Ce projet permet actuellement au laboratoire de pouvoir élargir ses horizons de recherche en accédant aux nombreux logiciels réalisés en C++.

Références Bibliographiques

CDT, E. (2012, Mai 20). *Eclipse CDT documentation*. Consulté le Mai 20, 2012, sur Eclipse CDT documentation: http://www.eclipse.org/cdt/documentation.php

Erich Gamma, R. H. *Design Pattern.* Boston: Vuibert.

GUEHENEUC, Y.-G. (2012, mai 20). *Ptidej*. Consulté le mai 20, 2012, sur Web of the Ptidej Team: http://www.ptidej.net/