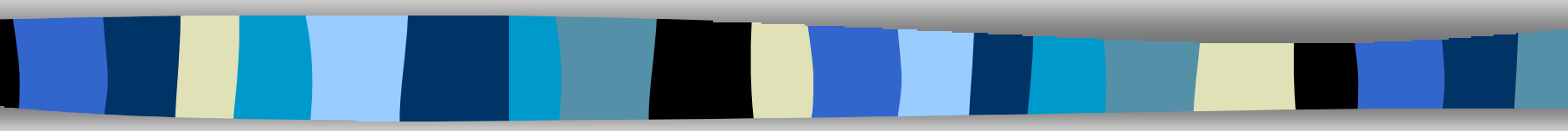
C++ Parser for PADL



Ward Flores

Sébastien Robidoux

IFT3051 – Projet été 2004

Département d'informatique et de recherche opérationnelle

Université de Montréal

Contenu

- Introduction
- Choix d’un analyseur syntaxique
  - Critères.
  - Problèmes rencontrés.
  - ANTLR et JavaCC.
- Connexion avec l’outil Ptidej UI
  - Ajouts des actions sémantiques.
  - Ajouts des structures au méta-modèle.
  - Ajouts des tests JUnit.
- Discussion (questions à volonté ! ☺)

Introduction

- **Objectifs**

  - Fait l’état de l’art des analyseurs syntaxiques

  - Raccorder l’analyseur le plus prometteur avec l’outil de rétro-conception Ptidej UI.

Introduction

- **Environnement de travail**
  - Eclipse avec CVS
  - Langage de programmation: Java
  - Système d’exploitation: Windows

Choix d’un analyseur syntaxique

- **Critères de sélection pour choisir notre analyseur syntaxique**

  - Analyse une version standard de C++.

  - Complexité du parser (pour le comprendre, l'utiliser).

  - Utilisable par d'autre programmes (Ptidej).

  - Status du parser (en développement, maintenue, abandonné).

Choix d’un analyseur syntaxique

- **Critères de sélection pour choisir notre analyseur syntaxique**

  - Temps d’analyse.

  - Outil implémenté en C++ ou Java.

  - Complexité de la grammaire (pour le compilateur de compilateurs).

  - Retourne un ASA facilement utilisable.

Choix d’un analyseur syntaxique

- **Problèmes rencontrés**

  - Création d’un arbre de syntaxe abstraite (GCC, PCCTS, etc.).

  - Utilisation avec des programmes externes.

  - Certains manuels peu documentés (PUMA).

Choix d’un analyseur syntaxique

- **Problèmes rencontrés**

  - Compatibilité avec le langage C++ pour les compilateur de compilateurs (JavaCUP).

  - Trouver une grammaire complète et fonctionnelle (Spirit).

  - Compatibilité avec Windows et Unix (CppCC).

Choix d’un analyseur syntaxique

- **ANTLR**

  - ***Fonctionnement***

```
class ExprParser extends Parser;
expr:   mexpr ((PLUS|MINUS) mexpr)*;
mexpr:   atom (STAR atom)*;
.... suite

class ExprLexer extends Lexer;
options {....}
LPAREN: '(' ;
RPAREN: ')' ;
.... suite
```

Choix d’un analyseur syntaxique

- **ANTLR**

  - *Classes générées…*

    * ExprLexer.java
    * ExprParser.java
    * ExprParserTokenTypes.java
    * ExprParserTokenTypes.txt

Choix d’un analyseur syntaxique

- **ANTLR**

  - Création d’une classe Main pour pouvoir utiliser le parseur

```
import antlr.*;
public class Main {
    public static void main(String[] args) throws Exception {
        ExprLexer lexer = new ExprLexer(System.in);
        ExprParser parser = new ExprParser(lexer);
        parser.expr();
    }
}
```

Choix d’un analyseur syntaxique

- **ANTLR**

  - Et pour générer des ASAs…

```
class ExprParser extends Parser;
options { buildAST=true; }
expr:   mexpr ((PLUS^|MINUS^) mexpr)*;
… suite
```

  - Il faut ajouter aussi dans la grammaire une subclasse qui hérite de TreeParser.

Choix d’un analyseur syntaxique

- **Utilisation d’ANTLR avec C++**

  - ANTLR est implémenté en Java mais si on travaille avec une grammaire C++, on génère de fichier C++

```
header { … }
options { language = "Cpp";}
.... suite
class ExprParser extends Parser;
.... suite
class ExprLexer extends Lexer;
.... suite
```

Choix d’un analyseur syntaxique

- **ANTLR**

  - On arrive à compiler l’outil et ainsi générer un analyseur syntaxique pour le langage C++ mais…

  - Une autre grammaire C++ pour ANTLR disponible sur le web mais…

Choix d’un analyseur syntaxique

- **JavaCC**

  - Fonctionnement

  * Compilateur de compilateurs LL(k)
  * Résultats en Java

Choix d’un analyseur syntaxique

- **JavaCC** Fonctionnement de la grammaire

```
void simple_type_specifier() :

{}
{
   (
      builtin_type_specifier()

   |
      qualified_type()

   )
}

void builtin_type_specifier() :

{}
{
      “void” | “char” | “short” | “int” | “long” | “float” |

      “double” | “signed” | “unsigned”

}
```

Choix d’un analyseur syntaxique

- **JavaCC** Fonctionnement de JJTree

```
void simple_type_specifier() :
{ /* Création d’un noeud */ }
{
   (
      builtin_type_specifier()
   |
      qualified_type()
   )
{ /* Ajout du noeud dans l’arbre */ }
}

void builtin_type_specifier() :
{ /* Création d’un noeud */ }
{
      “void” | “char” | “short” | “int” | “long” | “float”
      |
      “double” | “signed” | “unsigned”
{ /* Ajout du noeud dans l’arbre */ }
}
```

Choix d’un analyseur syntaxique

- **JavaCC** Utilisation avec une grammaire C++
  - PMD
  - Statique
  - Problèmes rencontrés
    - * SymtabManager.java

Choix d’un analyseur syntaxique

- **Comparaisons entre ANTLR et JavaCC**

Critères	ANTLR	JavaCC
Analyse une version standard de C++	Oui	Oui
Retourne un ASA facilement utilisable	ASA possible*	Oui
Utilisable par d'autre	Oui	Oui
Sans interface graphique	Oui	Oui
Outil implémenté en C++ ou Java	Java	Java
La grammaire C++produit des fichiers C++ ou Java	C++	Java
Grammaire facile à utiliser	Un peu	Oui

*ASA est possible, mais doit être implanté manuellement. Plus compliqué à faire.

Choix d’un analyseur syntaxique

- Et le grand gagnant est …

JavaCC

Connexion avec l’outil Ptidej UI

- **Construction du modèle avec l’aide du méta-modèle PADL**

  - Abstract Factory

  - IdiomLevelModel

  - Fonctions *create()* dans *CppCreator.java*

Connexion avec l'outil Ptidej UI

- **Les Classes**

  - Représentation d'une classe dans l'ASA

```
->external_declaration
-> declaration
->  declaration_specifiers
->   class_specifier
```

  - *createClass()*

  - Mais il faut tout définir avant de créer la classe…

Connexion avec l’outil Ptidej UI

- **L’accès** (public, protected, private…)

```
-> member_declaration
->  access_specifier
```

- *setVisibility()*

Connexion avec l’outil Ptidej UI

- **Les membres**

  - Représentation d’un membre dans l’ASA

```
-> member_declaration
->  declaration_specifiers
->   builtin_type_specifier
->  member_declarator_list
->   member_declarator
->    declarator
->     direct_declarator
->      qualified_id
```

Connexion avec l'outil Ptidej UI

- **Les membres**
  - plusieurs membres (`int a,b,c;`)
  - tableaux
  - pointeurs
  - *createMember()*

Connexion avec l’outil Ptidej UI

- **Les méthodes**

  - les paramètres

```
-> parameter_list
->  parameter_declaration_list
->   parameter_declaration
->    declaration_specifiers
->     builtin_type_specifier
->    declarator
->     direct_declarator
->      qualified_id
```

  - *createParameter()*

Connexion avec l’outil Ptidej UI

- **Les constructeurs**

```
-> member_declaration
->  ctor_definition
->   dtor_ctor_decl_spec
->   ctor_declarator
->    qualified_id
->   compound_statement
```

- createConstructor()

Connexion avec l’outil Ptidej UI

- **Les destructeurs**

```
-> member_declaration
->  dtor_definition
->   dtor_ctor_decl_spec
->   dtor_declarator
->    simple_dtor_declarator
->   compound_statement
```

- *createDestructor()**

Connexion avec l’outil Ptidej UI

- **Les méthodes « ordinaires »**

```
-> member_declaration
->  function_definition
->   declaration_specifiers
->    builtin_type_specifier
->   function_declarator
->    function_direct_declarator
->     qualified_id
->   func_decl_def
```

- createMethod()

Connexion avec l’outil Ptidej UI

- **<u>Les Structures et les Unions</u>**
  - Identique au classes
  - Seul le « jeton » est différent
  - Aucune méthode
  - *createStructure()** ou *createUnion()**

Connexion avec l’outil Ptidej UI

- **Les Enumerations**

```
-> external_declaration
->  enum_specifier
->   enumerator_list
->    enumerator
```

*- createEnum()**

Connexion avec l’outil Ptidej UI

- **Les Variables globales**

```
-> external_declaration
->  declaration
->   declaration_specifiers
->    builtin_type_specifier
->   init_declarator_list
->    init_declarator
->     declarator
->      direct_declarator
->       qualified_id
```

- *createGlobalField()**

Connexion avec l’outil Ptidej UI

- **Les lien d’héritage entre classes**

```
-> external_declaration
->  declaration
->   declaration_specifiers
->    class_specifier
->     base_clause
->      base_specifier
->       access_specifier
->      base_specifier
->       access_specifier
```

- Héritage multiple en C++ !

Connexion avec l’outil Ptidej UI

- **<u>Les autres liens… (*UseRelationship*)</u>**

```
-> declaration_specifiers
->  qualified_type
->   qualified_id
```

- createUseRelationship()

Connexion avec l’outil Ptidej UI

- **<u>Les Entités Fantôme (*Ghost*)</u>**
  - Représente une entité non présente dans le projet analysé.
  - Doit être remplacé si possible

  - *createGhost()*

Connexion avec l'outil Ptidej UI

- **<u>Ajout des structures manquants dans le Méta-modèle PADL</u>**

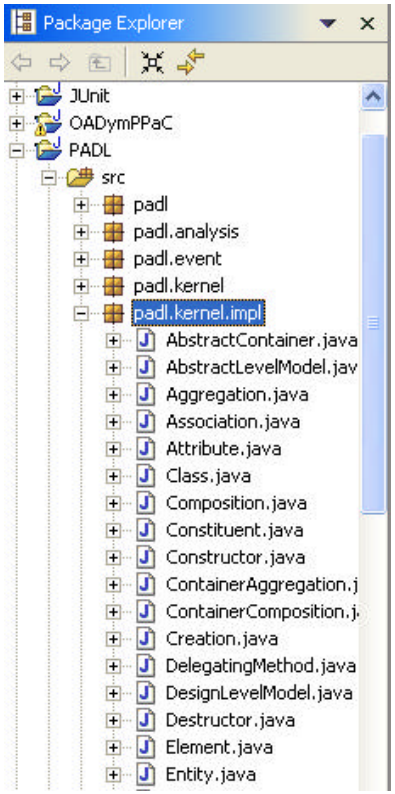
  - Méta-modèle complet pour Java
  - Java est semblable à C++ mais…
  - Ajout des structures: lesquels ?
    où ?
    comment ?

Connexion avec l’outil Ptidej UI

- **<u>Les nouveaux constituants à ajouter:</u>**
  - Les Destructeurs
  - Les Structures
  - Les Énumérations
  - Les Unions
  - Les GlobalField

Connexion avec l'outil Ptidej UI

- **La procédure à suivre…**



- Ajout des interfaces
- Ajout des classes
- Ajout des méthodes dans l'interface *IFactory* et la classe *Factory*

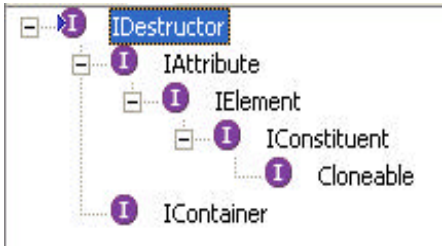
Connexion avec l’outil Ptidej UI

- **<u>À quel niveau dans la hiérarchie du PADL les ajouter ?</u>**

  Nous savons dans quel package les ajouter mais maintenant, à quel niveau dans la hiérarchie des classes du méta-modèle devrait-on les ajouter ?

Connexion avec l'outil Ptidej UI

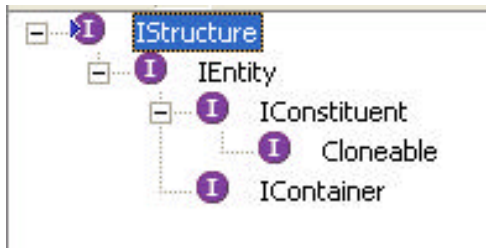
- **IDestructor**



- facile, car « comportement » très semblable au *constructeur*.

Connexion avec l’outil Ptidej UI

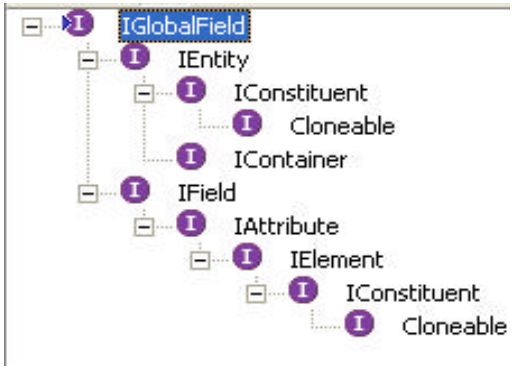
- **IStruct**



- au même niveau que l’entité class
- même chose pour **IEnum** et **IUnion**

Connexion avec l’outil Ptidej UI

- **IGlobalField**



- au même niveau que l’entité class, mais hérite de l’entité *Field* aussi

Connexion avec l’outil Ptidej UI

- Le méta-modèle peut supporter le langage C++ grâce aux nouveaux constituants.
- Qu’est-ce qu’on fait maintenant ?
- Comment ?

Connexion avec l’outil Ptidej UI

- Nous commençons en ajoutant les classes manquantes dans le package *ptidej.ui.kernel* du projet **Ptidej UI**.
- Ensuite, le procédé à suivre dépend s’il s’agit d’une entité ou d’un élément qu’on ajoute.

Connexion avec l'outil Ptidej UI

- **Pour les entités…**

Welcome | AbstractModelGraph.java

```
    }
    private Entity createEntity(final IEntity anEntity) {
        Entity entity = null;

        if (anEntity instanceof IClass) {
            entity =
                new Class(
                    this.getPrimitiveFactory(),
                    this,
                    (IClass) anEntity);
        }
        /* 2004/08/10: Sébastien Robidoux, Ward Flores */
        else if (anEntity instanceof IStructure) {
            entity =
                new Structure(
                    this.getPrimitiveFactory(),
                    this,
                    (IStructure) anEntity);
        }
        else if (anEntity instanceof IUnion) {
            entity =
                new Union(
                    this.getPrimitiveFactory(),
                    this,
                    (IUnion) anEntity);
        }
        else if (anEntity instanceof IEnum) {
            entity =
                new Enum(
                    this.getPrimitiveFactory(),
                    this,
                    (IEnum) anEntity);
        }
        /*END*/
        else if (anEntity instanceof IGhost) {
```

- On modifie la classe *AbstractModelGraph*

Connexion avec l’outil Ptidej UI

- **<u>Pour les éléments…</u>**

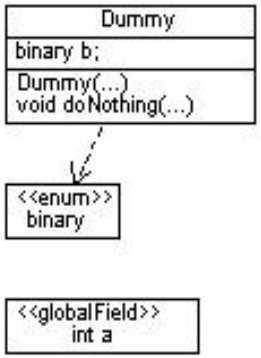
  - Modifier la méthode *addElement()* de la classe *Entity*

Connexion avec l'outil Ptidej UI

- **Représentation graphique**

Welcome | enum.cpp

```
enum binary {zero, un};
int a;
class Dummy {
  public:
    Dummy(){};
    binary b;
    void doNothing();
};
```



Connexion avec l’outil Ptidej UI

- **Ajout des tests JUnit**

  - Création du projet PADL C++ Creator Tests

  - Ajout de quatre suites de tests pour vérifier le bon fonctionnement du projet

Connexion avec l'outil Ptidej UI

- **Récupération et tri de l'information avant de faire les tests.**

Welcome | TestWorld.java

```
protected void setUp() {

    if (TestWorld.Entities == null
        || TestWorld.Elements1 == null
        || TestWorld.Elements2 == null) {
        final IIdiomLevelModel idiomLevelModel =
            Primitive.getFactory().createIdiomLevelModel("World.cpp");
        idiomLevelModel.create(
            new CppCreator(new String[] { "test_src/world.cpp" }));

        // All the entities of the idiomLevelModel
        TestWorld.Entities =
            new IEntity[idiomLevelModel.listOfActors().size()];
        idiomLevelModel.listOfActors().toArray(TestWorld.Entities);

        // All the element of the first class
        TestWorld.Elements1 =
            new IElement[TestWorld.Entities[0].listOfActors().size()];
        TestWorld.Entities[0].listOfActors().toArray(TestWorld.Elements1);

        // All the element of the second class
        TestWorld.Elements2 =
            new IElement[TestWorld.Entities[1].listOfActors().size()];
        TestWorld.Entities[1].listOfActors().toArray(TestWorld.Elements2);
    }
}
```

Connexion avec l'outil Ptidej UI

- **Tests pour vérifier les entités (class, struct, etc) et les éléments (constructor, method, etc).**

Welcome | TestWorld.java

```
}

public void testClass() {
    //----------------------------------------
    TestWorld.assertEquals(
        "Class name",
        "Game",
        ((IClass) TestWorld.Entities[0]).getDisplayName());

    TestWorld.assertEquals(
        "Class type",
        Modifier.PUBLIC,
        ((IClass) TestWorld.Entities[0]).getVisibility());

    //----------------------------------------
    TestWorld.assertEquals(
        "Class name",
        "Team",
        ((IClass) TestWorld.Entities[1]).getDisplayName());

    TestWorld.assertEquals(
        "Class type",
        Modifier.PUBLIC,
        ((IClass) TestWorld.Entities[1]).getVisibility());
}

public void testEnum() {
    TestWorld.assertEquals(
        "Enum name",
```

Connexion avec l'outil Ptidej UI

- **Tests pour vérifier l'héritage multiple**

Welcome | TestInheritance.java

```
        TestInheritance.Entities[2].listOfInheritedActors());

    //----------------------------------------
    //Rectangle extends Forme et Forme2
    nonEmptyList.add(TestInheritance.Entities[0]);
    nonEmptyList.add(TestInheritance.Entities[2]);
    TestInheritance.assertEquals(
        "Inherited Rectangle Actors",
        nonEmptyList,
        TestInheritance.Entities[3].listOfInheritedActors());
    nonEmptyList.clear();
}

public void testInheritingActors() {
    //----------------------------------------
    //Forme is superClass of Rectangle
    nonEmptyList.add(TestInheritance.Entities[3]);
    TestInheritance.assertEquals(
        "Inheriting Forme Actors",
        nonEmptyList,
        TestInheritance.Entities[0].listOfInheritingActors())
    nonEmptyList.clear();
```

Connexion avec l’outil Ptidej UI

- **Autres tests…**

  - tests pour vérifier la relation d’usage
  - tests pour vérifier l’entité ghost

Discussion

- **Problèmes rencontrés**
  - Recherche d’un bon analyseur…
  - Analyse de projet incomplet avec l’analyseur de JavaCC.

Discussion

- **<u>Améliorations possibles et futures</u>**
  - Spécification des liens entités
  - Modernisation de la grammaire
  - Élimination de *SymtabManager.java*
  - Améliorer les nouveaux constituants du méta-modèle.

Conclusion

- **Évolution personnelle**

FIN

Fiou…