Argyle仕様書

V0.2

改訂履歴

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | 日付  (改訂者) | 改 訂 内 容 | 改訂理由 | 備考 |
| 0.1 | 2015/01/27 (艾迪) |  |  | 下書き |
| 0.2 | 2015/02/24 (艾迪) | Argyle内部仕様追加 | 他の開発者に全体が分かるよう |  |

1. **はじめに**
   1. 本書の目的

本書はArgyleの仕様を定義する．

Argyleは幾つかのMining Software Repositoryの高速化ツールを扱うDSLである．

現在扱うツールは：

1. 開発メーリングリストマイニングの前処理ツール： **MLMining.jar, Coli.jar**
2. コードクローン検出ツール： **scorpio.jar**
3. リポジトリマイニングツール： Num Changes Co-Changed Files(以降，**NCCF**)
4. Git commitから欠陥予測ツール： **anko**

それぞれについて，DSLで扱う仕様を順番に説明していく．

Argyle使用目的は，次の通りである．

1. 研究者
   * Mining Software Repository という分野でソフトウェアのデータ分析に使う
   * ソフトウェアマイニングにおけるメールの解析
   1. 用語定義

|  |  |  |  |
| --- | --- | --- | --- |
| 用語 | 読み | 説明 | 出典 |
| Argyle | アーガイル | 本書が対象とするDSLの名前 | 本書 |
| DSL |  | ドメイン固有言語 | プログラミング言語 |
| input data |  | 実験，分析用の為の入力データ |  |
| code clone | コードクローン | ソースコード中での類似または一致した部分 | プログラミング |
| thread | スレッド | ソフトウェア実行時にCPU利用の単位 | プログラミング |
| mail thread | メールスレッド | 通常は最初のメールへのすべての返信をまとまった単位。 | 電子メール |
| mail Mining |  | 大量のメールに対して，分析を行う |  |
| commit | コミット | gitにおける，複数のファイルの変更を記録する最小単位 | git |
| mailing list | メーリングリスト | 複数の人に同時に電子メールを配信するためのメールアドレスのリスト | 電子メール |
|  |  | ツール名 |  |
| agl |  | DSLのファイル拡張子 |  |

* 1. 記号定義

|  |  |  |
| --- | --- | --- |
| 記号 | 例 | 説明 |
| 文字の塗りつぶし | Output <URI>; | Argyleの専用言語表記 |
| $から始まる文字列 | $ java -jar scorpio.jar -d cxf/ -o output/cxf.xml -s 6 -t 2 | コマンドラインにおける命令形式 |

1. **要求とArgyle表記**

要求は各ツールで既に実装された機能をDSLで扱えること．

各ツールの機能は以下とおり．

* 1. **開発メーリングリストマイニングの前処理ツール**

表1 オプションとDSLの対応関係(川島論文からの引用とDSLと対応付け)

|  |  |  |
| --- | --- | --- |
| オプション | 処理内容 | DSL表記 |
| -f [file name] | 入力ファイルを指定 |  |
| -o [file name] | 出力ファイルを指定 | Output <URI>; |
| DataLoader mr = new DataLoader();  mr.insertMbox(mboxfile name); | メールデータを追加する | ImportMbox <URI>; |
| mr.insertDevInfo(commit log file name); | GitLogデータを追加する | ImportGitLog <URI>; |
| new CalMetrics(mr.getDataManager()); | ソーシャルネットワークメトリクスを算出する | CalcSocialMetrics; |
| CreateSNAData csd = new CreateSNAData(mr.getDataManager());  GraphViewer gv = new GraphViewer(csd.getGraph());  gv.draw(); | グラフを描画する | DrawGraph; |
| -a | 別名の問題を解決 | ResolveMailAlias; |
| -d [file name] | 開発者情報を付加 | ResolveMailDeveloper; |
| -s | 引用部分を削除 | RemoveMailQuote; |
| -t | スレッド化 | ExtractThread; |
| -ignore [address] | 該当アドレスからのメールを削除 | RemoveUselessMail; |
| 未対応 | 引用部分を取り出す | GetMailQuote; |
| 未対応 | ある文字列を含むスレッドを抽出 |  |

* 1. **Code clone検出**

以下は大坂仕様から引用

|  |
| --- |
| ○実行例  $ java -jar scorpio.jar -d cxf/ -o output/cxf.xml -s 6 -t 2  -d は分析対象ソースコードが含まれているディレクトリへのパスです．  -o は検出結果出力先ファイル名です．  -t はスレッド数です．  -s はコードクローンとして認識するコード片の最小のサイズを指定するものです．  今回は 6 としています．  これで，ソースコードの行数で言うと大体5行程度に対応すると考えていただければと思います．  ○オプションの説明  （※ソースコードから読み取ったものなので少し間違えているかもしれない）  -d dir  解析対象のJavaソースコードがあるディレクトリdir  -ad another\_dir  解析対象のJavaソースコードがあるディレクトリanother\_dir  crossオプション参照  -s minimumsize  検出するコードクローンの最少の大きさminimumsizeを指定する．  大きさは，同形部分グラフの頂点数で与える．  「同形部分グラフの頂点数＝コードクローンを構成する分野式の数」  -o output  コードクローンの検出結果を出力するファイル名output  -t thread  スレッド数threadの指定  -cross on/off  オプションdで指定されたディレクトリdir1とオプションadで指定されたディレクトリdir2のコードクローン検出の方法を指定．  onならばdir1とdir2を比較したコードクローン検出のみを行い，dir1にあるファイル同士，及びdir2にあるファイル同士のコードクローン検出は行わない．  offならばdir1とdir2は一つのディレクトリとみなしてコードクローン検出を行う．"  -v on/off  ログを吐き出すかどうかの指定．  -C on/off  プログラム依存グラフを構築する際に用いる依存関係の種類に制御依存を指定するためのオプション．  デフォルトはon  -D on/off  プログラム依存グラフを構築する際に用いる依存関係の種類にデータ依存を指定するためのオプション．  デフォルトはon  -E on/off  プログラム依存グラフを構築する際に用いる依存関係の種類に実行依存を指定するためのオプション．  デフォルトはon  -M on/off  (わからない．．．)  デフォルトはon |

表2

|  |  |  |
| --- | --- | --- |
| オプション | 処理内容 | DSL |
| -d [directory] | 解析対象のJavaソースコードがあるディレクトリdir | ImportCodeDir <URI>; |
| -ad [another\_dir] | 解析対象のJavaソースコードがあるディレクトリanother\_dir  crossオプション参照 | ImportAnotherCodeDir <URI>; |
| -s [minimumsize] | 検出するコードクローンの最少の大きさを指定 | SetDetectMinSize <NUM>; |
| -o output | コードクローンの検出結果を出力するファイル名 |  |
| -t thread | スレッド数threadの指定 | UseThread <NUM>; |
| -cross on/off | -adオプション参照かどうか |  |
| -v on/off | ログを吐き出すかどうか |  |
| -C on/off |  |  |
| -D on/off |  |  |
| -E on/off |  |  |
| -M on/off |  |  |

実行

DetectCodeClone;

* 1. **不具合モジュール予測 NCCF**

以下は大坂仕様からの引用

|  |
| --- |
| ○実行方法  メインクラス：  src/journal/nccf/thread/exe/Execute.java  引数：  args[0]:スレッド数  args[1]:入力ファイルのパス  args[2]:出力結果を書き込むファイルのパス  実行例：  $ java journal.nccf.thread.exe.Execute 4 input/eclipse/e\_all.txt output.txt  ○入力ファイル  各行が1コミットになっていて，少し特殊な形で記述されている．  androidとeclipseのデータセットを使っており，  androidを例にファイルの違いを説明すると，  ・a\_2009.txt : 2009年の1年分のコミット  ・a\_200901.txt : 2009年1月の1ヶ月分のコミット  ・a\_200901-half.txt : 2009年1月から半年分のコミット  ・a\_200901-weak.txt : 2009年1月1-7日の1週間分のコミット  ・a\_all.txt : 2005-20011年の全期間のコミット  ses2013\_osaka.pdfの4.3にデータセットについて少し書いてあります． |

表3

|  |  |  |
| --- | --- | --- |
| オプション | 処理内容 | DSL |
| args[0]: | スレッド数 | UseThread <NUM>; |
| args[1]: | 入力ファイルのパス | ImportGitLog <URI>; |
| args[2]: | 出力結果を書き込むファイルのパス | Output <URI>; |

実行

DefectPredict;

* 1. **Git commitから欠陥予測ツール： anko**

実行例：(田中仕様から引用)

|  |
| --- |
| $ cd ${HOME\_ANKO}  $ cd ./src/  $ ruby set.rb --model glm  use glm as modeling techniques.  $ ruby set.rb --dataset eclipse.jdt.core.csv  use eclipse.jdt.csv as training data.  $ ruby set.rb --join eclipse.jdt.csv,bugzilla.csv  joined eclipse.jdt.csv,bugzilla.csv in "dataset.csv". |

表4ruby set.rbオプションとDSL仕様との対応付け

田中仕様より

|  |  |  |  |
| --- | --- | --- | --- |
| **option** | **argument** | **description** | **DSL** |
| -m, --model | glm,ksvm,nnet,rpart,rf | Change modeling techniques. | ChangeTrainModel; |
| -d, --dataset | file\_name.csv | Change training data. | ChangeTrainData <URI> |
| -j, --join | file\_name.csv,file\_name.csv | Use joined data as training data. | UseJoinData <URI> |
| -b, --[no-]bugspots |  | Use bugspots algorithm for prediction. |  |
| -s, --[no-]simple |  | Use simple version metrics when measure it. | UseSimpleMetrics; |

* 1. **グローバル拡張オプション**

by [Distribute|Parallel];

例：

DetectCodeClone by Parallel;

* 1. **結果出力パス設定**

Output <URI>;

* 1. **データ入力，削除**

ImportMbox <URI>;

ImportGitRepository <URI>;

ImportGitLog <URI>

ImportCodeDir <URI>;

RemoveMbox <URI>;

RemoveGitRepository <URI>;

RemoveGitLog <URI>;

RemoveCodeDir <URI>;

1. **Argyleの動作環境**

Version1.0では，ローカル環境のEclipse上でDSLのコードを作成し，動くようにする。

Version1.1はFX10で実際のプログラムを実行させる．

1. **Argyle内部仕様**
   1. **アーキテクチャ**

Argyleの各DSLは全て同名のメソッドが対応して定義される。ユーザーはあるDSLを書く時、同名のメソッドを実行すると等しい。

この設計を実現するために、動的メソッドの呼び出しを採用して、**複数のメソッドとパラメータをディスパッチ**している。

* 1. **開発環境**

**Eclipse LUNA** **Release 4.4.0**

**Java SDK 1.8**

**Xtext SDK 2.6.0**

**slf4j 1.7.9**

**lombok 1.14.8**

* 1. **XtextでのDSL定義**
* エディタ上入力補助が出るよう、かつプログラム側でDSLモデルを取得する時、DSLの全体が分かるように定義する必要がある。
* パラメータの型によって、名前は決まっている
* Importの場合
  + importPath<=>STRING
* Calcの場合
  + size<=>INT
  + path<=>STRING
  + 例：

|  |
| --- |
| (dsl='SetDetectMinSize' size=INT ';') |

* 1. **複数のメソッドとパラメータのディスパッチ**

パラメータを型チェックし，呼び出し先のメソッドの引数と同じ順番で並ぶ．

リストに詰める

* 1. **ツール呼び出し**

DataReceiverの invokeメソッドを参照

* 1. **新しいツールや機能の追加時に，DSLの拡張及びラッピング**
  + Xtextファイル内でDSL定義追加
  + ArgyleMethodListクラス内に新しいメソッドを作成（名前はDSLで定義したと同じ）@Aglアノテーションをつける．処理内容をここで書く．
  + - 例

|  |
| --- |
| DSL  SetDetectMinSize 2  Java code:  @Agl  **public** **void** SetDetectMinSize(**int** size) |

* + 結果出力するために、新しいXXXDataReceiverを作成し（Singletonパターンを使う）、ツール名が分かるようにXXXを名付ける。DataReceiverクラスを継承すれば、実装すべきメソッド：
    - setParameter XXXDataReceiver内持っているデータを呼び出し側のパラメータ順番に沿って並べる。パラメータの順番は上の各ツールの仕様を参照してください。
    - invoke 新しいツールのmainメソッドを呼び出す。引数String[] parametersを用いる。呼び出すmainメソッドは上の各ツールの仕様を参照してください。
    - initialize XXXDataReceiver内のデータを全て初期化。
    - canOutput 新しいツールを呼び出し時に、必要なパラメータは全て獲得したかどうかをチェックする。必要なパラメータは実装時に応じて開発者自分で判断してください。
  + 以上のメソッドを実装すれば、ArgyleMethodListクラス内新たに追加したDSLメソッドからXXXDataReceiverクラス内のexecuteメソッドを呼び出せば、自動的にcanOutput->setParameter-> invokeの順番で実行される(Template Method パターン)
  1. **initializeのタイミング**
  + XXXDataReceiverクラスは全部Singletonパターンを採用しているため、データは永遠にGCに回収されない。専用のDSLを作って、データ消去するか、RemoveMbox <URI>;などを実行する時データ消去するか、或いは更に良い方法があれば、それでもいい。いずれにせよ、タイミングを考えなければいけない。今のところ、このメソッドを使わなくてもいい。
  1. **クラス一覧**

jp.ac.kyushu.argyle.impl

ArgyleModelReader DSLで書かれたものをモデルに変換するクラス

ArgyleMethodList DSLと対応つけるべきメソッドを全てここに定義する

Agl DSLと対応しているメソッドのタグ

Activator Eclipseプラグインとして、必要な情報を書かれたクラス

jp.ac.kyushu.argyle.impl.actions

Argyle メニューから「実行」ボタンを押す時の処理, 主にrunメソッドを注目

ArgyleController runメソッドは全ての実行の始まるところ（Façadeパターン）

ArgyleDispatcher DSLの情報を取得し、ArgyleMethodListから同名メソッドを呼び出す

jp.ac.kyushu.argyle.impl.basefunction

ProjectReader 現在のプロジェクトを取得するクラス

jp.ac.kyushu.argyle.impl.exception

DispatchException 例外クラス

jp.ac.kyushu.argyle.impl.model

DataReceiver DSL実行中結果を一時的保存するクラス。ツール別でサブクラスを

持っている。DSLからツール実行を呼び出す時、ここのクラスの

executeメソッドを使う。