Dependency Graph Creator Engine

Job Heersink (s3364321) Richard Westerhof (s3479692)

November 24, 2020

github repostiory: https://github.com/richardswesterhof/pyne

1 User Guide

Installing
To install Pyne, Java JDK 11+ and Maven is required. The installation itself is rather straightforward: Clone or download our fork of the Pyne repository:

```
git clone https://github.com/richardswesterhof/pyne.git
```

Install Pyne's dependencies and build an executable jar by running the following command inside Pyne's root directory:

```
mvn install
```

Executing
After the application has been built, one can use this application via CLI. This can be done by running the following command from the root directory:

```
java -jar <path-to-jar>/pyne-cli-1.0-SNAPSHOT-jar-with-dependencies.jar
  ↪ <github-url>
```

Where the `<path-to-jar>` is the path to the built jar file (by default this is in `<pyne_root>/pyne-cli/target/`) and `<github-url>` is the url of the project you want to create a dependency graph for. For example, this would be the command to create a yearly dependency graph for the commits of Apache Tajo from 2015 to November 2020:

```
java -jar pyne-cli/target/pyne-cli-1.0-SNAPSHOT-jar-with-dependencies.jar
  ↪ https://github.com/apache/tajo -s "2015-01-01" -e "2020-11-21" -p
  ↪ "YEAR"
```

2 Design

This application is divided into 3 different packages: pyne-demo, pyne-cli and pyne-api. The relations between these packages and their containing classes can be seen in figure 1.

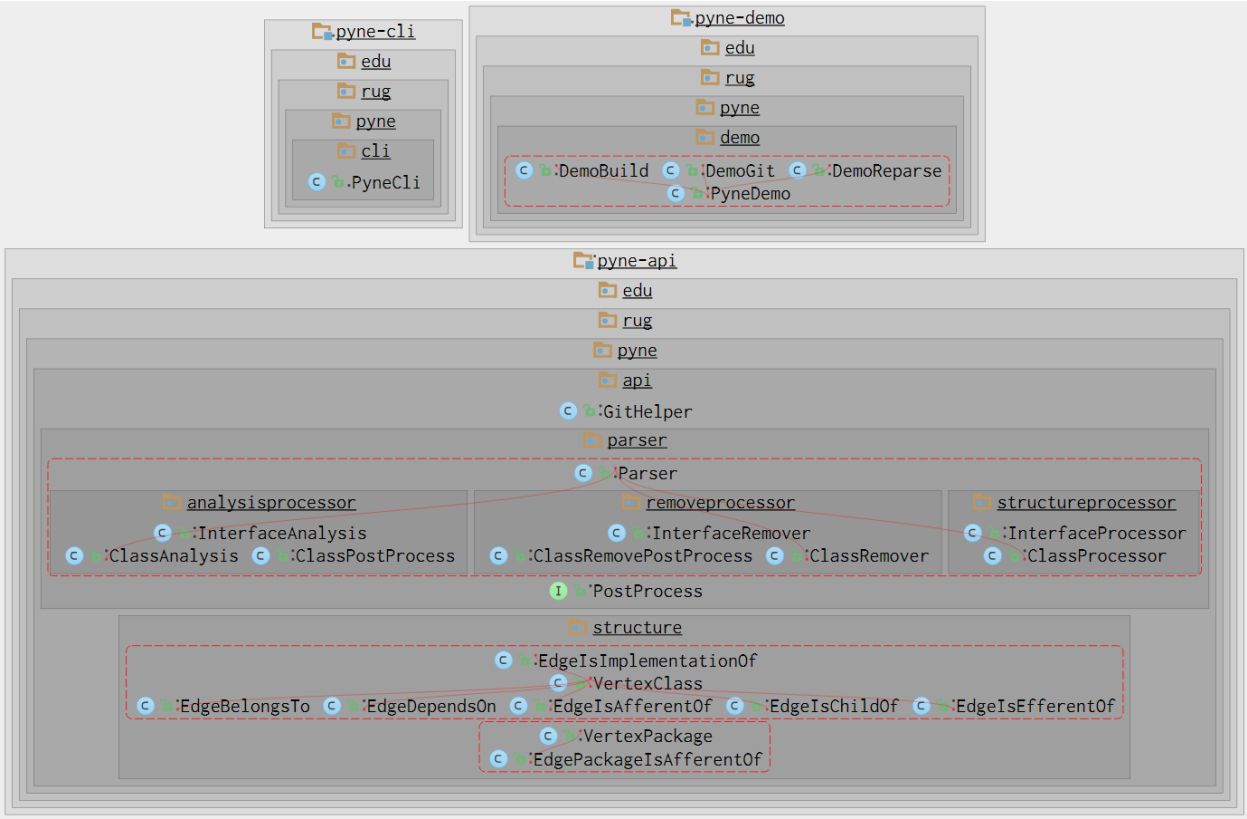

Figure 1: The full graph of the relations between the classes of pyne

The purpose of these 3 packages will be briefly explained here. The following sections will go into more detail on pyne-api, since this is the main component of the system.

2.1 Pyne-demo

This package acts as a testing ground for pyne. Most of the data in this package has been hard-coded to work only with a specific repository that was probably present on the creators local machine. Since we do not know which repository they used for testing and it is not present within the project, This package will not be able to work and we will consider this package deprecated.

2.2 Pyne-cli

This package acts as a layer of communication between the user and the Pyne-api via a CLI (Command Line Interface). Given a git repository as a program argument, it'll automatically start communicating with the pyne-api and create a dependency graph in `.graphml` format for each commit. A number of options can be given to the pyne-cli program. The first required argument is of course the repository itself, but one can also specify the starting and end date to parse from, the interval between each commit and an input and output directory.

2.3 Pyne-api

This package is the heart of the program and is the one that is actually responsible for the dependency graph creation. This package is again subdivided into 2 packages: parser and structure, and contains one class: GitHelper.

GitHelper
The GitHelper is a part of the API responsible for git related functionality like cloning a repository in a temporary location, parsing a commit or processing a commit. Most of the functionality of the GitHelper depends on the parser class to create the dependency graph.

structure
The structure package does not contain any functionality, instead it provides abstract classes to build a dependency graph from. Special edges have been created for different kinds of relations: BelongsTo, DependsOn, AfferentOf, ChildOf, EfferentOf, ImplementationOf and PackageisAfferentOF. Special vertexes have also been created to represent either a package or a class.

2.3.1 parser

The parser package again consist of three sub-packages: analysisProcessor, removeProcessor, and structureProcessor. It also contains the Parser class and the PostProcess interface.

Parser
The Parser class is used to keep track of the files that have changed in the selected commit, and it holds lists of all types of processor classes, each of which are responsible for analysing the code in a different way.

PostProcess
The PostProcess interface simply contains one method for post processing the graph when all information has been extracted from the code by the relevant processor classes.

StructureProcessor
This package contains the following classes:

ClassProcessor
Processing a class means adding a node representing this class to the graph. If needed, it also creates a node for the package this class is included in, in the graph.

InterfaceProcessor
InterfaceProcessor is essentially the same as processing a class, therefore it simply consists of a ClassProcessor class and some wrapper methods.

AnalysisProcessor
This package contains the following classes:

ClassAnalysis
The ClassAnalysis class analyses one class from the source code and gathers its details, like the declaration, dependencies, superclass, etc.

ClassPostProcess
An implementation of the PostProcess interface. It removes orphan nodes and creates any missing edges. It can do this because after the whole class has been analysed, we can be sure that we have all relevant information, whereas some of this information may have been still missing during the initial analysis.

InterfaceAnalyis
InterfaceAnalysis is essentially the same as analysing a class, therefore it simply consists of a ClassAnalyis class and some wrapper methods.

RemoveProcessor
This package contains the following classes:

ClassRemover
The ClassRemover removes any classes from an existing graph if the corresponding class in the source code was removed in the selected commit.

ClassRemovePostProcess
The post processing for removing a class consists of removing all edges that point to or from the deleted node.

InterfaceRemover
InterfaceRemover is essentially the same as removing a class, therefore it simply consists of a ClassRemover class and some wrapper methods.

3 Problems:

1. **Bug in graph creator**:
   When I open a graph in a graph editor like draw.io or yEd graph editor, only one square appears. But the file is 2.25mb. Turns out that all the squares are stacked upon each-other, but the relation between the squares is still there (which is visible from the neighbours tab in yEd graph editor).
   It seems that the squares itself do not contain any data on the classes they represent. They do not have any value assigned to them. Furthermore, opening any graph will give a warning about the 'linesOfCode' property being of type long, which apparently isn't supported. I sincerely hope that an integer would be enough to count the amount of lines of code anyway, so we'll change that.
2. **Default cli values**:
   the default options for cli are not the best choice. Namely the period option is set to days by default, which will create a separate graph for each day. The start date is set to 5 periods (so 5 days on default) from the end date (which is the current date by default). It is relatively rare for a git repository to have commits from the past 5 days. We shall therefore change the default values.
3. **Error handling**:
   In a few cases, where something goes wrong or no graph is created, no error is thrown and the user does not know why no graph has been made. For that reason more error logging statements need to be added.