[P2PBrowser group] Code Architechture

Steffan Norberhuis Quinten Stokkink

12 June 2013

1 Introduction

Our projects consists of two seperate extensions of Tribler. Both use wxWebview to browse the internet. The first extension is the Eternal Webpage extension. This extension will allow a user to easily browse the web using peer-to-peer-swarms and also add new pages to the swarm. The second extension is the TUPT extension. This extension allows you to browse websites and if a movie is detected on the website, then you can begin streaming this movie with one click of a button.

The only common part between the two extensions is the WebBrowser class. After we finished the Eternal Webpages extension, we started working on the TUPT extension. We transformed the WebBrowser class and stripped all functionality specific for the EternalWebpages extension that was no longer needed. All needed functionality for the TUPT extension to the WebBrowser class has been added in a generic way. This generic approach was much harder for the WebBrowser, because we changed the way how WebBrowser loads pages.

The projects use Python 2.7 and are compiled as part of the Tribler project (see `https://github.com/Tribler/tribler`). Because we cannot/should not deliver code that is not our own we have ripped out our code from the main Tribler code. The result of this move is that the project is uncompilable and some of the tests unrunnable. This should not really be of any hindrance to evaluating the code though.

We are hosting our code on github (see `https://github.com/norberhuis/tribler`)

2 Eternal Webpages diagrams

The Eternal Webpages package is a Proof Of Concept package no longer undergoing active development. Its main purpose is to download and bundle resources referenced by a webpage in such a way that they can be fetched offline. To accomplish this, the webbrowser has 2 modes for viewing websites: one by browsing the internet and one by downloading P2P cached copies of websites (called internet viewmode and swarm viewmode respectively). This is shown in Figure 1. The package is referenced from the main Tribler projects web browser for every resource it ecounters while loading a webpage. This is shown in Figure 2. The full class diagram of the package is included in Figure 3.

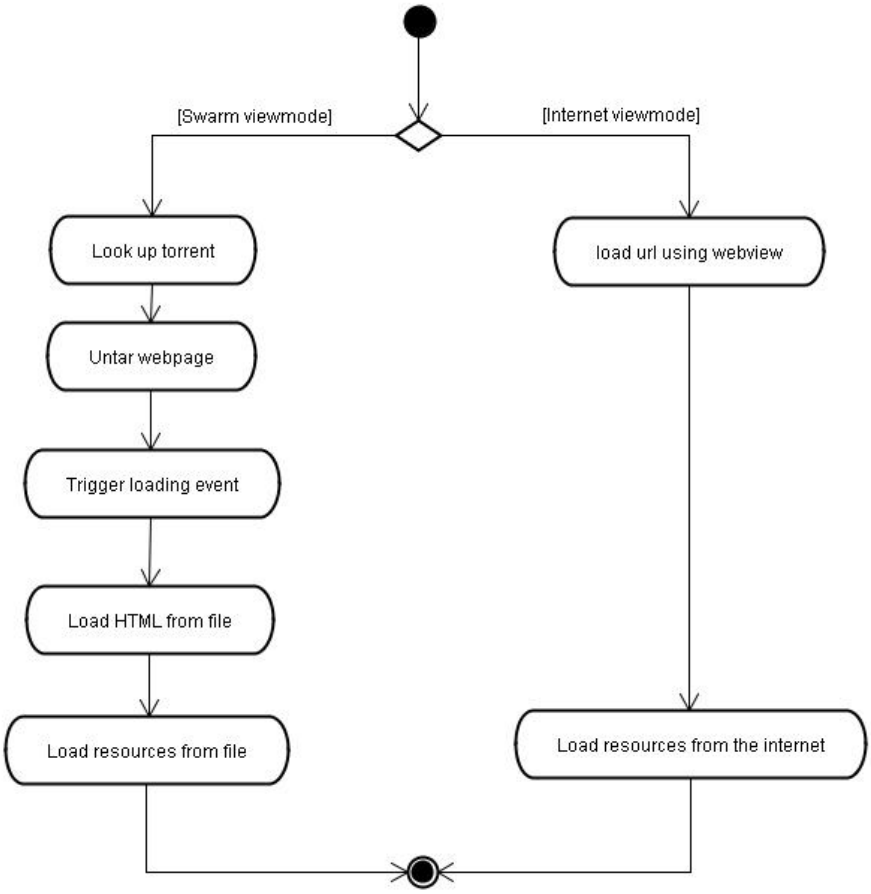

Figure 1: Activity diagram of viewmode based webbrowsing

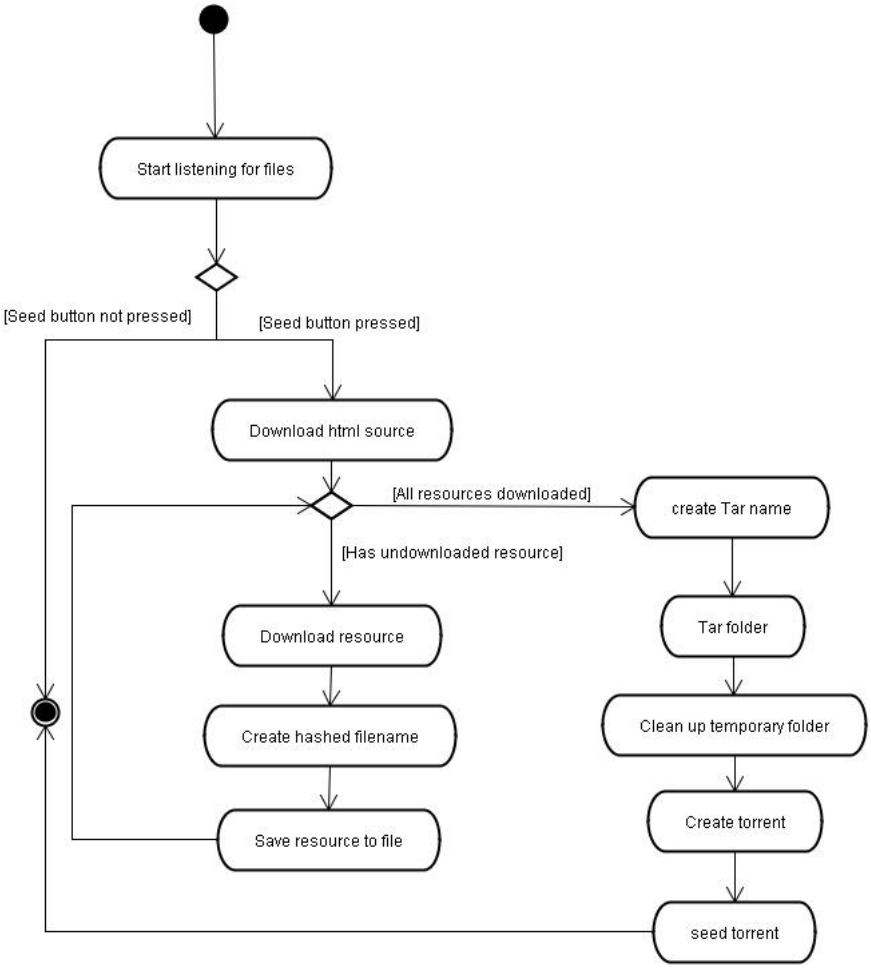

Figure 2: Activity diagram of seeding mode webbrowsing

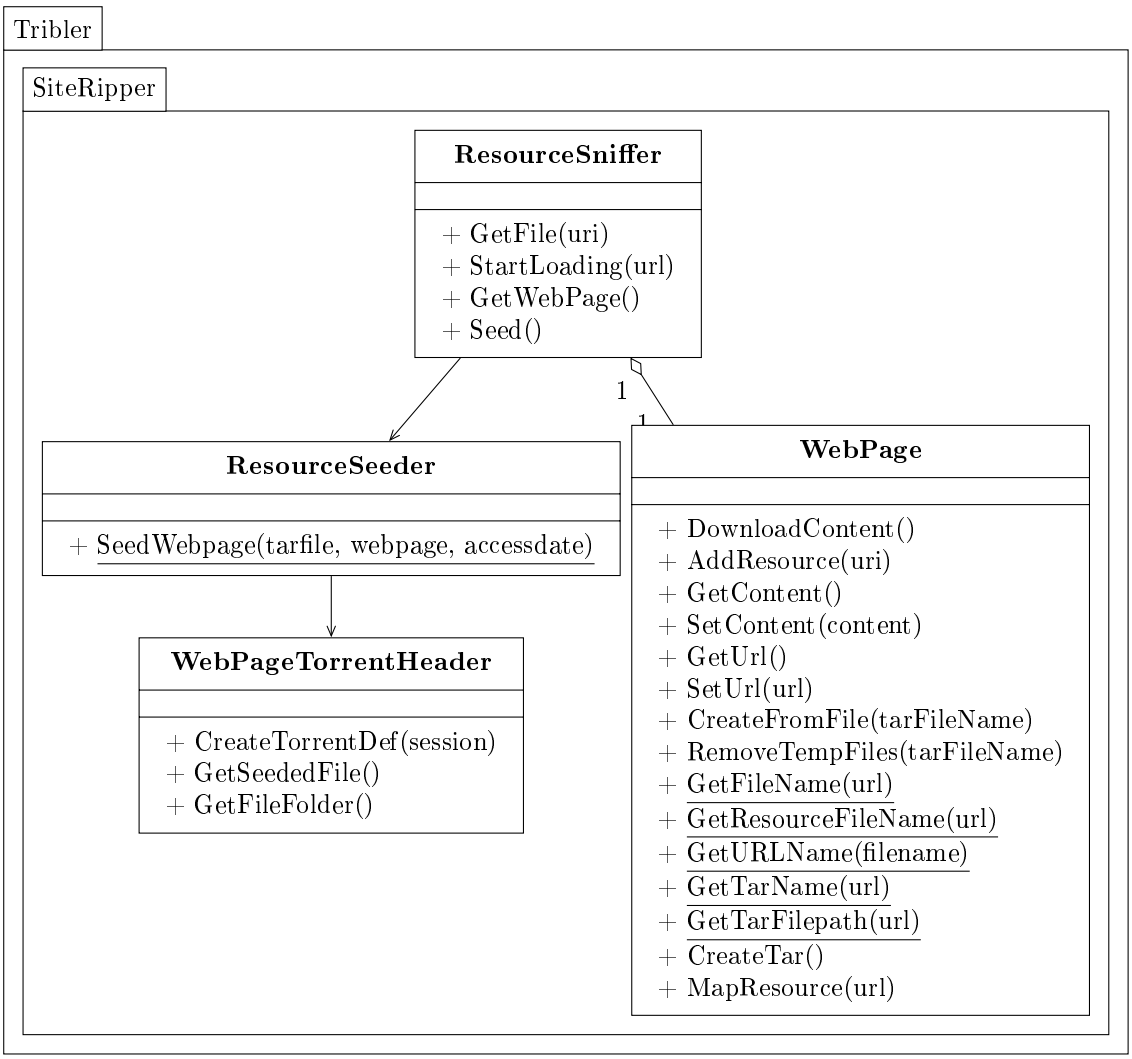

Figure 3: Class diagram for Eternal Webpages

3 TUPT diagrams

The TUPT package is a package in active development. Its main purpose is to (a) scan a site for metadata, (b) correct metadata and (c) find resources corresponding to this corrected metadata. The current implementation of TUPT is aimed at movies, with the resource collection being done through torrent sites. The main activity diagram for this functionality can be found in Figure 4. The general package class diagram for the TUPT project can be found in Figure 5. The specific top-level TUPT package classes class diagram can be found in Figure 6. The main entrypoint for the package is the TUPTControl class. The process of the aforementioned TUPT chain goes through 4 sub-packages, in order these are: the Parser, the Matcher, the TorrentFinder and then the Channels sub-package.

First a webpage is parsed for possible movies by the Parser package (for class diagram see Figure 7). Given an URL the Parser control class will query its plug-ins whether they can parse the website. If they can, they are queried to process the raw html of the website and return none or more found movies (in the form of Movie classes). The main entrypoint for the sub-package is the ParserControl class.

Secondly this (possibly very minimal) movie metadata will be corrected and supplemented. This is done by the Matcher package, see Figure 8 for the activity diagram of how different results from different plugins are bound together to correct the metadata. The class diagram of this sub-package is shown in Figure 9. The main entrypoint for the sub-package is the MatcherControl class.

Thirdly this corrected movie metadata is used to find a corresponding torrent file in the TorrentFinder sub-package. Plug-ins are queried to produce .torrent-file links and/or magnet links for the given movie metadata, see Figure 10 for the activity diagram of this function. The class diagram of this sub-package is shown in Figure 11. The main entrypoint for the sub-package is the TorrentFinderControl class.

Lastly the corrected movie metadata coupled to the best torrent file that could be found and inserted cleanly into a user's Tribler channel. The class diagram for this Channels sub-package is given in Figure 12. The main entrypoint for the sub-package is the MovieInserter class.

4 Tests

Our package comes with a set of python 'unittest' files. This section maps these unit-tests and explains how to run them. Some unit-tests cannot be run however due to the separation from the main Tribler project. There are two unit-tests for which this is an issue. As with the main project, all the tests are written for Python 2.7 . We also want to note that some unit-tests require an active internet connection.

4.1 Required packages

To run our tests we use Ubuntu machines. In Table 1 are the required 3rd party python packages and their Ubuntu package names required to run the unit-tests.

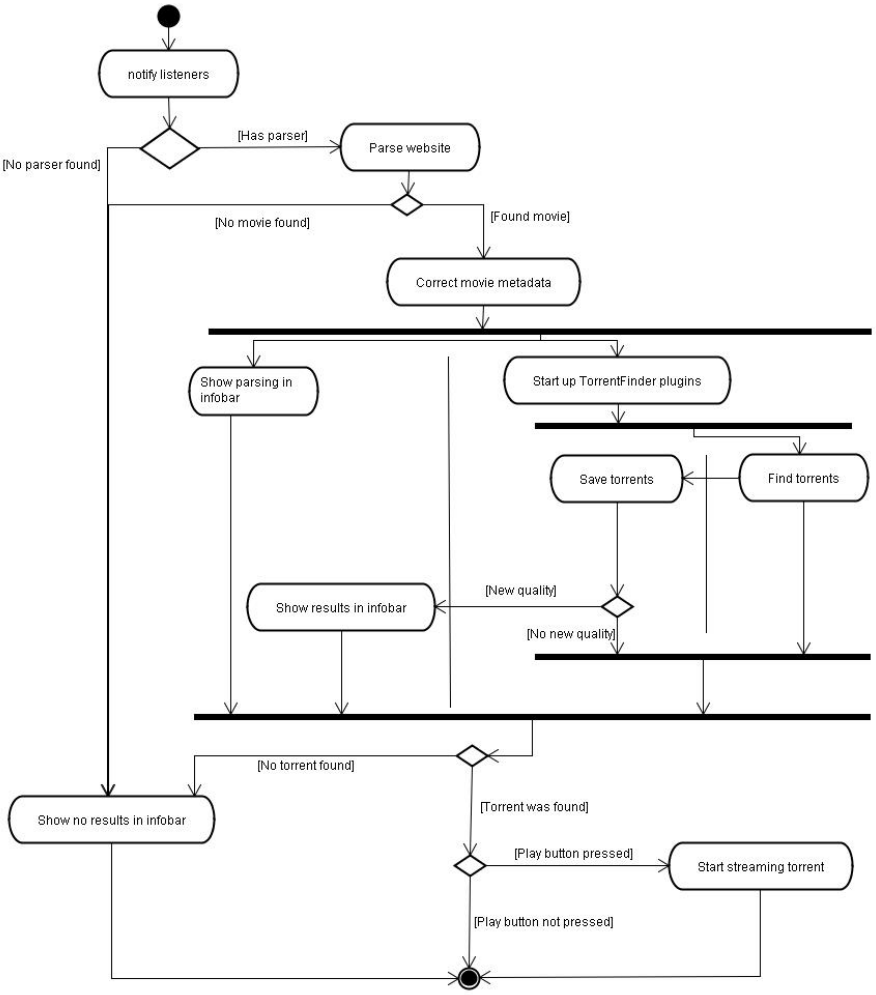

Figure 4: Activity diagram of general TUPT functionality

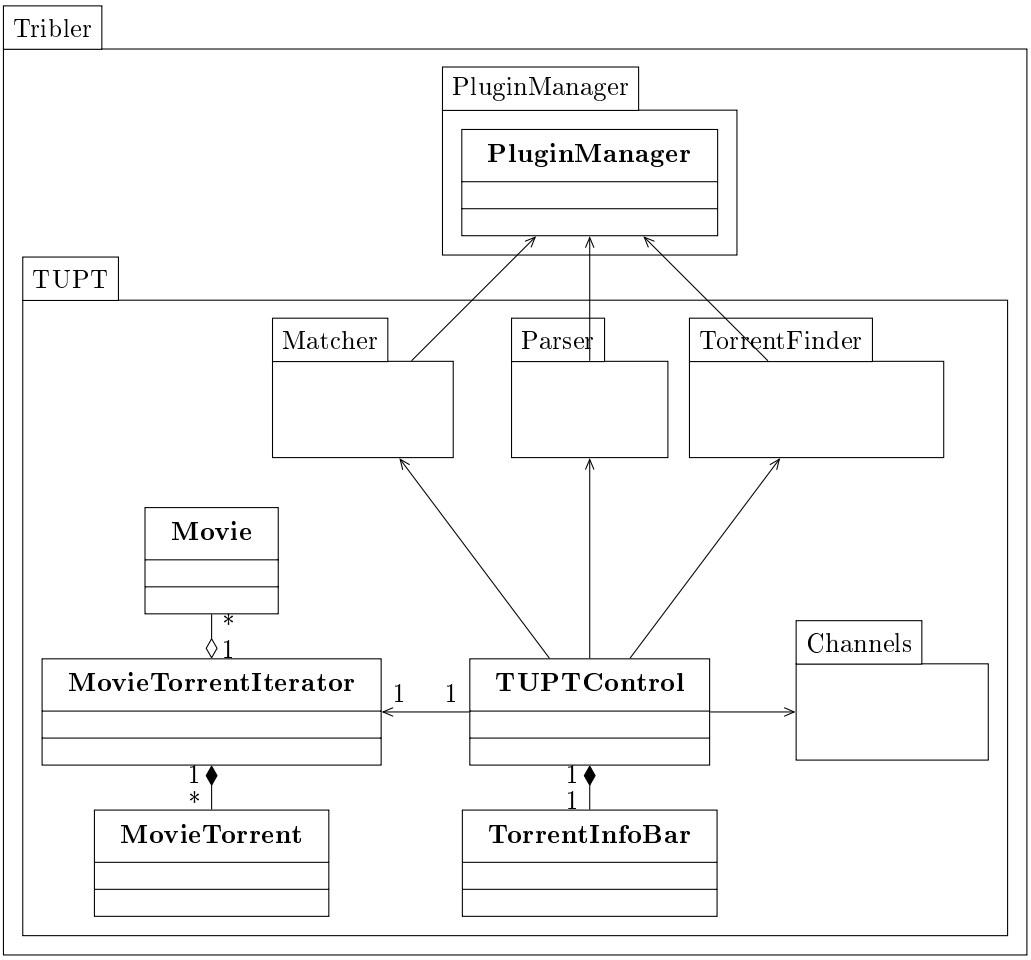

Figure 5: Class diagram for the entire TUPT package

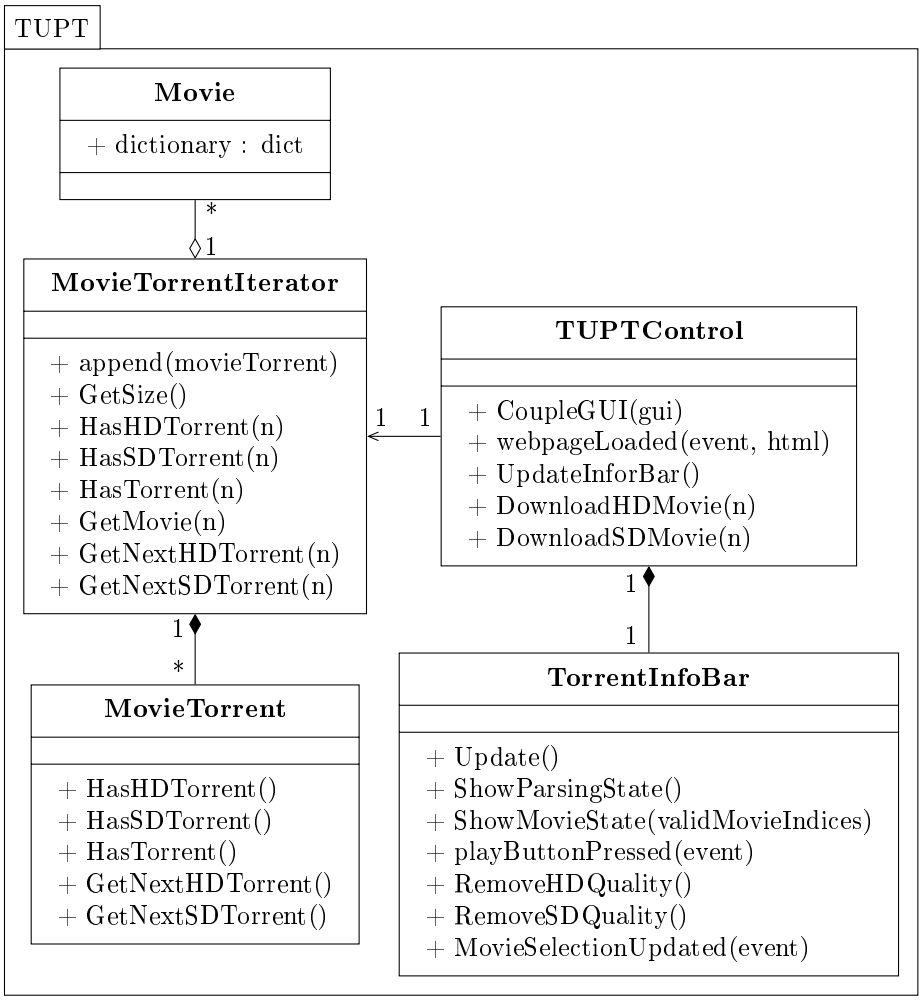

Figure 6: Class diagram for the top-level TUPT package

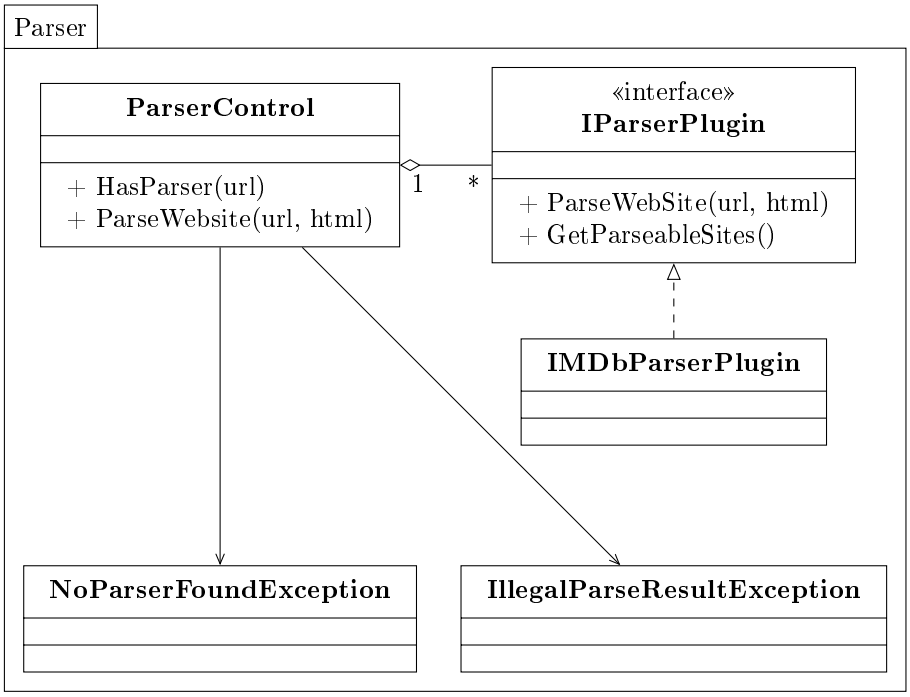

Figure 7: Class diagram for the Parser sub-package

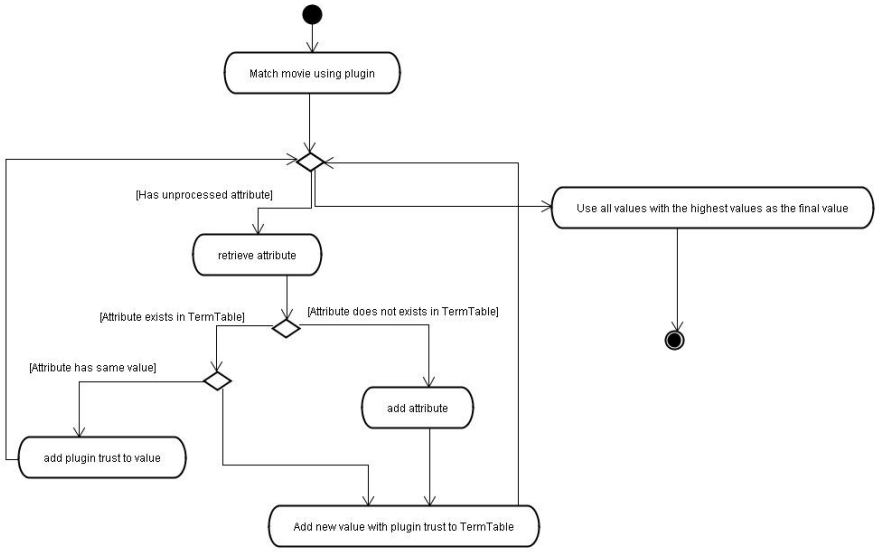

Figure 8: Activity diagram of the Matcher sub-package

Table 1: Required packages

Name	Ubuntu package
Yapsy	python-yapsy
wxPython	python-wxgtk2.8
IMDbPY	python-imdbpy
BeautifulSoup	python-bs4

4.2 Non-runnable tests

There are two unit-tests we cannot deliver runnable. Table Table 2 lays out the tests and the rationale behind why they are not runnable. Note all of the paths in the file names are offset from the *Tribler/Test/TUPT/* folder. These tests rely so heavily on core Tribler functionality, stubbing the Tribler classes would lead to unacceptable overlap of the stubs with the actual classes.

Table 2: Non-runnable tests

File	Rationale
test_TUPTControl.py	Requires a development build of wxPython (2.9.4.0) and relies heavily on core Tribler functionality.
Channels/test_ChannelControl.py	Relies heavily on core Tribler functionality.

4.3 Runnable tests

The following tests in Table 3 can be run normally (provided the 3rd party packages were properly installed) from the terminal. The file names are offset from the *Tribler/Test/* folder.

4.4 Test output

This subsection contains the console output for all of our tests. The raw console output is shown for each of the tests in Table 4

Table 3: Runnable tests

PluginManager/test_PluginManager.py
TUPT/Parser/test_ParserControl
TUPT/TorrentFinder/test_SortedTorrentList
TUPT/Matcher/test_MatcherControl
TUPT/Matcher/test_TestTheMovieDBMatcherPlugin
TUPT/Parser/test_IMDbParserPlugin
TUPT/TorrentFinder/test_KatPhTorrentFinderPlugin
TUPT/TorrentFinder/test_TorrentFinderControl

Table 4: Test output

File	**Output**

TUPT/test_TUPTControl	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 13 tests in 0.002s`
	
	`OK`

TUPT/Channels/test_ChannelControl	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 5 tests in 0.002s`
	
	`OK`

Table 4 – continued from previous page

File	Output

PluginManager/test_PluginManager	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 3 tests in 0.031s`
	
	`OK`

TUPT/Parser/test_ParserControl	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 6 tests in 0.001s`
	
	`OK`

TUPT/TorrentFinder/test_SortedTorrentList	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 6 tests in 0.004s`
	
	`OK`

TUPT/Matcher/test_MatcherControl	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 2 tests in 2.541s`
	
	`OK`

TUPT/Matcher/	

Table 4 – continued from previous page

File	Output

test_TestTheMovieDBMatcherPlugin	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 3 tests in 2.274s`
	
	`OK`

TUPT/Parser/test_IMDbParserPlugin	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 7 tests in 4.631s`
	
	`OK`

TUPT/TorrentFinder/
test_KatPhTorrentFinderPlugin
	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 3 tests in 0.970s`
	
	`OK`

TUPT/TorrentFinder/
test_TorrentFinderControl
	`Finding files... done.`
	`Importing test modules ... done.`
	
	`----------`
	`Ran 10 tests in 0.007s`
	
	`OK`

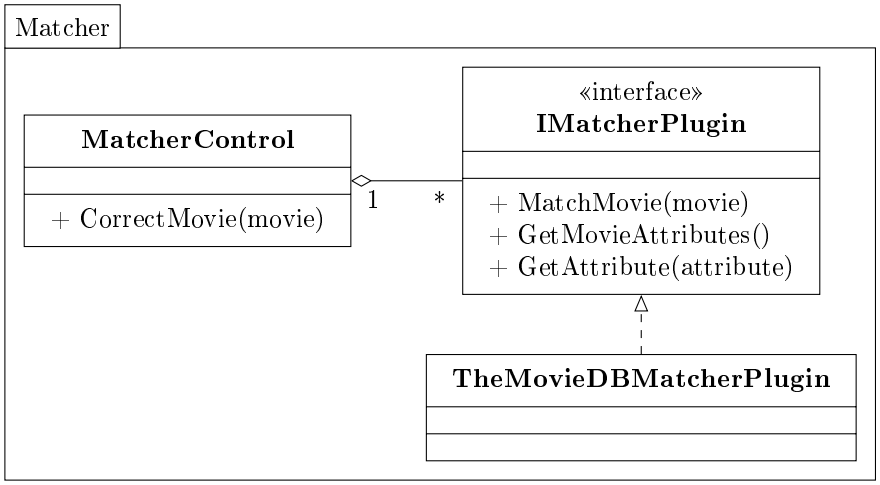

Figure 9: Class diagram for the Matcher sub-package

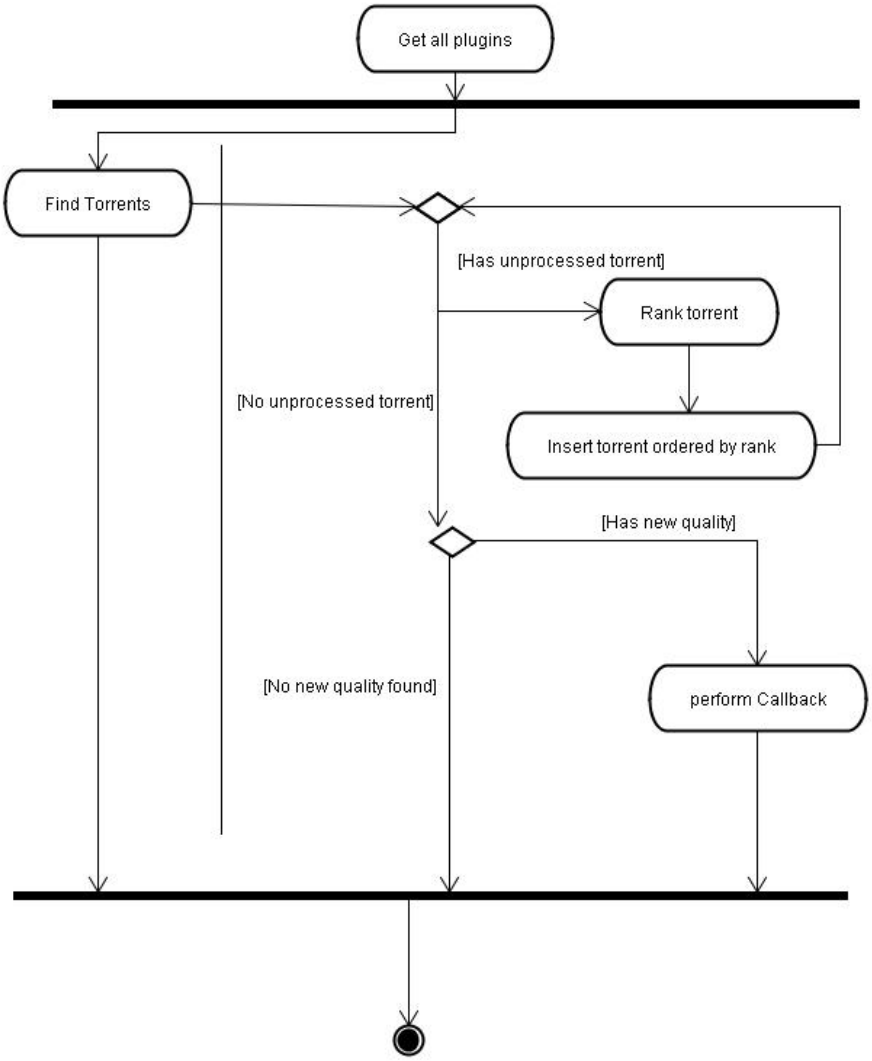

Figure 10: Activity diagram of the TorrentFinder sub-package

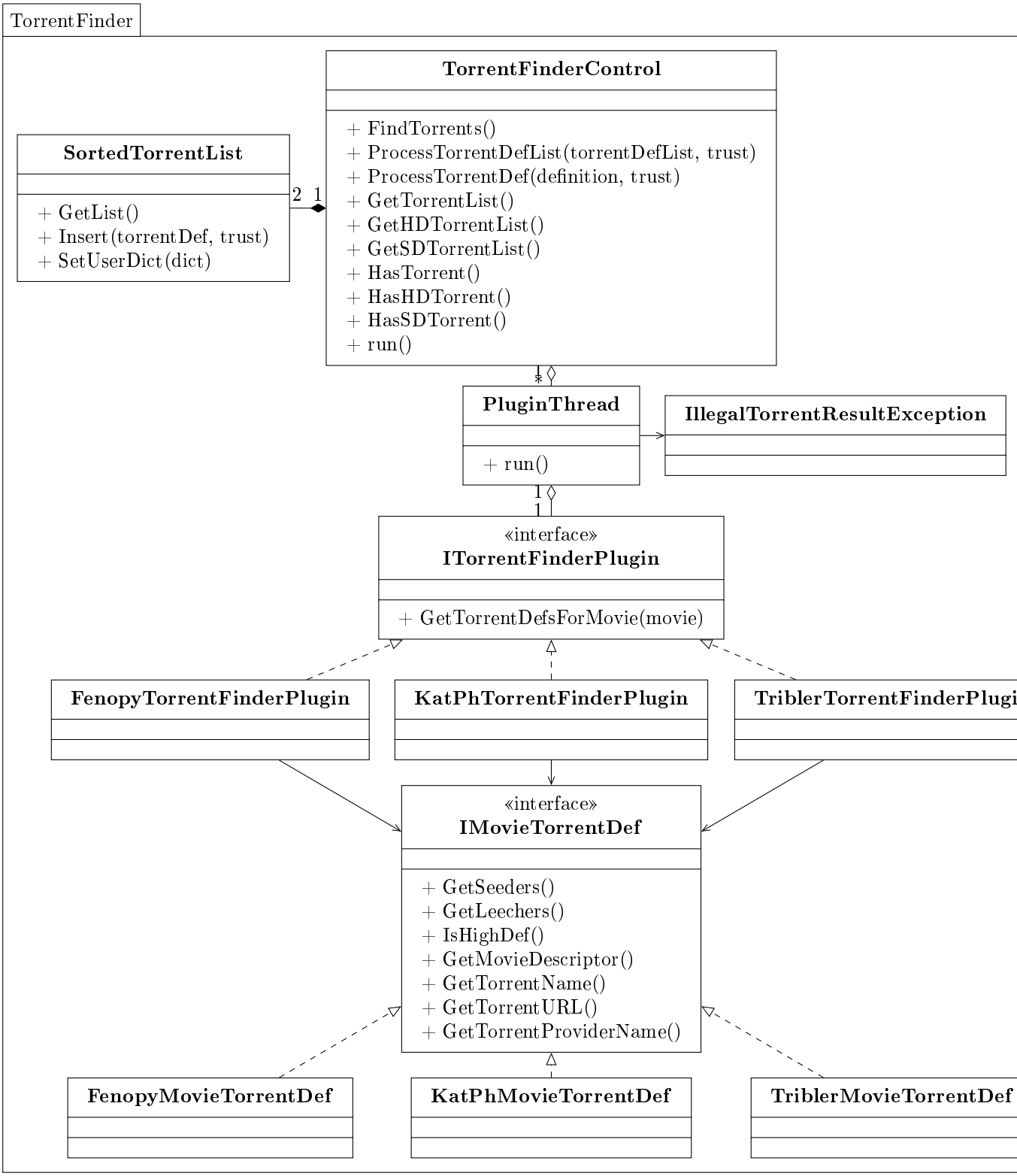

Figure 11: Class diagram for the TorrentFinder sub-package

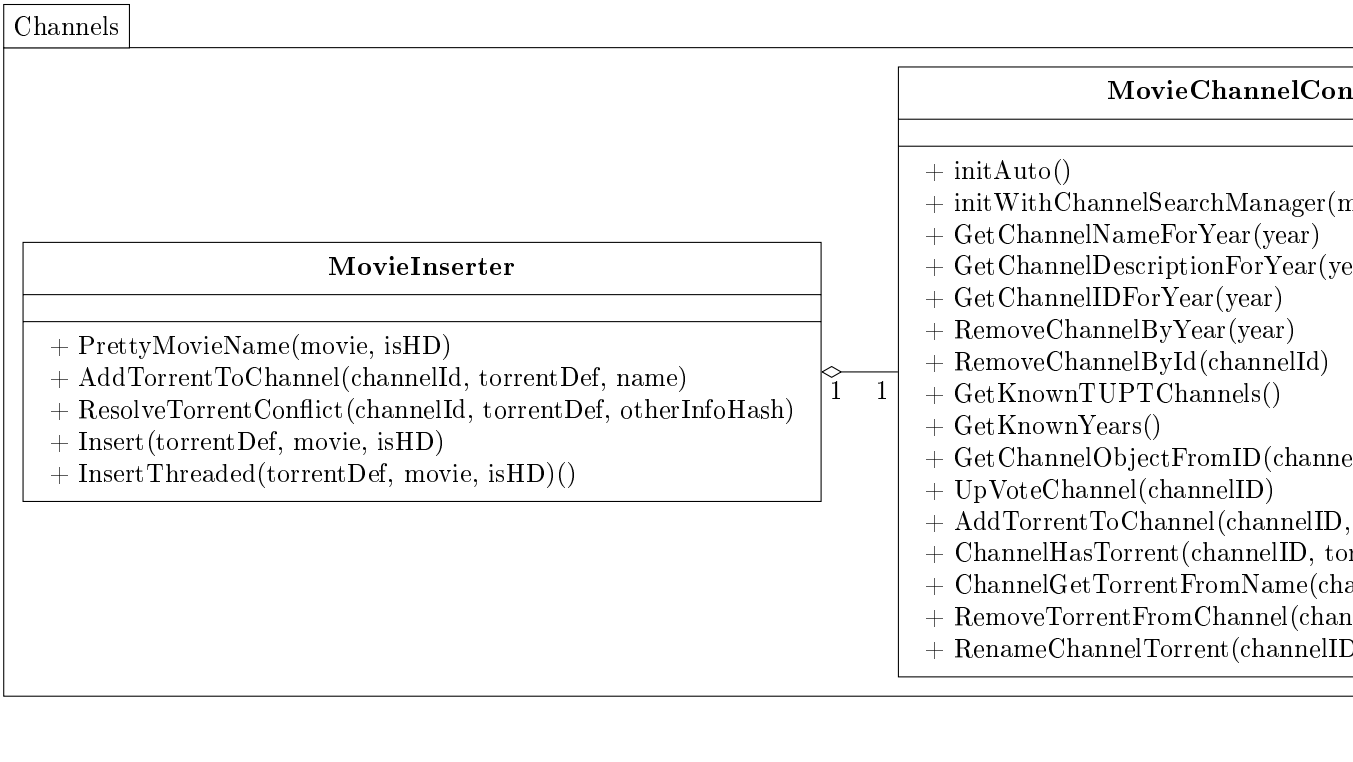

Figure 12: Class diagram for the Channels sub-package