Harmony

User Manual

LaBRI Software Engineering Group
http://se.labri.fr/

Version 0.9

Contents

Chapter 1

Introduction

Harmony is a framework that helps you design and run analysis done on source code or bug repositories. Examples of such analysis are (1) the computation of source code metrics with the intent to perform a correlation with bugs distribution, (2) the measurement of software developers practices to infer their impact on software development, or (3) the computation of statistics to better understand a set of software projects.

Harmony offers the following facilities that are mandatory to perform any analysis:

- It defines an abstraction layer to design your analysis. This abstraction layer defines few simple concepts used by any analysis (Source, Event, Item, Action and Authors).
- It manages interoperability by providing connectors to many well-known repositories (Git, Mercurial, SVN, BugZilla, etc.).
- It handles parallelism that can be done when running your analysis. Such a parallelism allows a better efficiency specially when the analysis has to be ran on a large set of software projects.
- It supports the scheduling of several analyses with the objective to design analysis by composition.
- It provides a persistence facility to store the results of analysis into data bases.

This document explains how to install, run, configure Harmony and how to design your own analysis.

Chapter 2

Try Harmony in 2 minutes

This brief chapter describes the few steps required to try Harmony on your computer. Harmony is written in Java. You must have at least Java 1.7 to run Harmony. You can download it from: `http://www.java.com/fr/`. Once it is done or if you already have Java on your system follow these simple steps:

- Download the Harmony executable from this repository: `https://code.google.com/p/harmony/downloads/list`
- Extract the archive
- Run the native launcher of the application located in the extracted folder:
  - *Harmony.exe* on Microsoft Windows
  - *Harmony* on Linux and Mac OS X

  You should now have loaded all the Harmony components and see the *OSGi* prompt : 'osgi>'. If you have an error check that your path is correctly set (see `http://www.java.com/fr/download/help/path.xml` for help).
- Finally just type *harmony* and wait for the default analysis to be executed.

Congratulations, you have just run your first Harmony analysis. You should have now an output directory named *out* that contains the results of this analysis. The default analysis is called *reporting* and generates several graphs on relationships between authors, events or item (see Harmony model for more explanation). The default source was used here, if you want to test Harmony on your own source repository the next chapter is for you.

Chapter 3

Installation

This chapter explains step-by-step how to set up a development environment for Harmony. The Harmony framework is based on the OSGi component model. Even though you may use any OSGi implementation for developing and running your analyses, this documentation shows how to do it with the Equinox implementation developed by the Eclipse community. That is why we use Eclipse as IDE in order to ease the development of new analyses.

Screencast available for this procedure:
`http://youtu.be/wFGXayQADeA`

3.1 Install

- Install Eclipse from `http://www.eclipse.org/downloads/`.
  - We recommend **Eclipse Standard 4.3 (Kepler)**
  - No installation support for **Eclipse IDE for Java Developers**
- Once downloaded, unzip the Eclipse archive
- Run Eclipse
- Go to `Help` → `Install new software`
- Click on `Add...` at the right of the "work with" field
- On the popup type:
  - *Name*: Harmony
  - *Location* `http://se.labri.fr/data/harmony/update-site`
- Click `OK` and select in the "work with" list the Harmony update site.

- Select `Harmony` on the list (see figure 3.1)
- Click on `Next`
- Accept the terms and click on `Finish`
- If a pop-up with *Security Warning* appears, Click on `OK`
- Restart Eclipse
- Harmony is now installed into your Eclipse installation

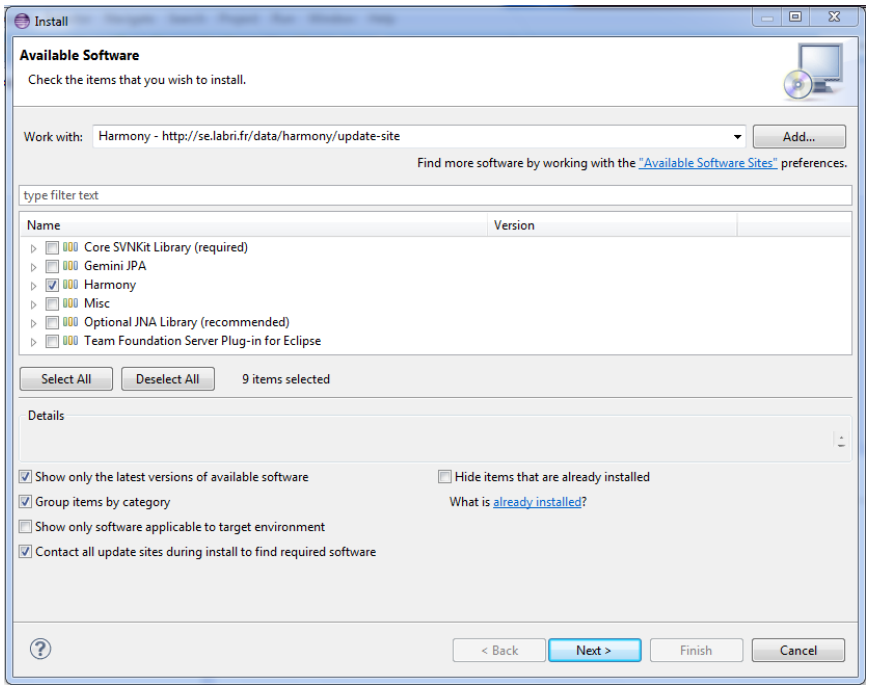

Figure 3.1: Harmony installation within the Eclipse IDE

3.2 Run

- Go to `File` → `New` → `Project...`
- Select `WorkingDirectory` under the `Harmony` category and click on `Next` (see figure 3.2)
- Click on `Finish`: the wizard will configure your development environment and launch the OSGi console.
- Type `harmony` in the OSGi console and press Enter (see figure 3.3)

- Wait for the harmony analysis to finish. Once done, make a right click on the *WorkingDirectory* project and select `Refresh`.
- If the analysis was successful the folders *tmp* and *out* should appear (see figure 3.3). You can take a look at the analysis results by looking at the produced files in the *out* directory.

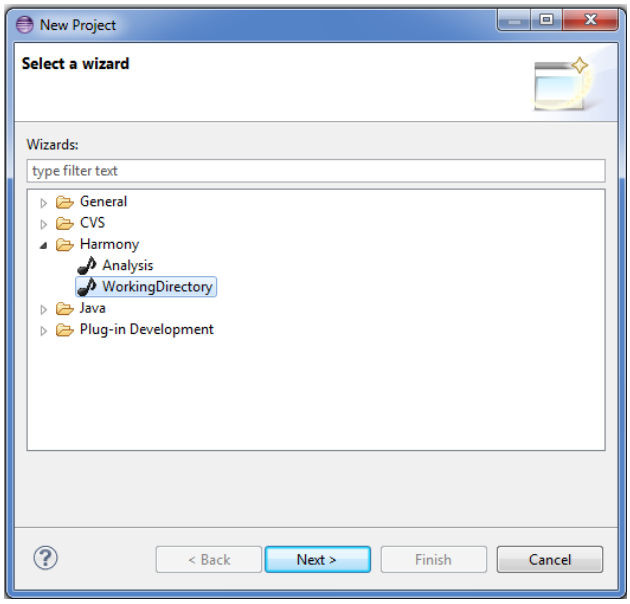

Figure 3.2: Selection of the wizard for creating a new working directory

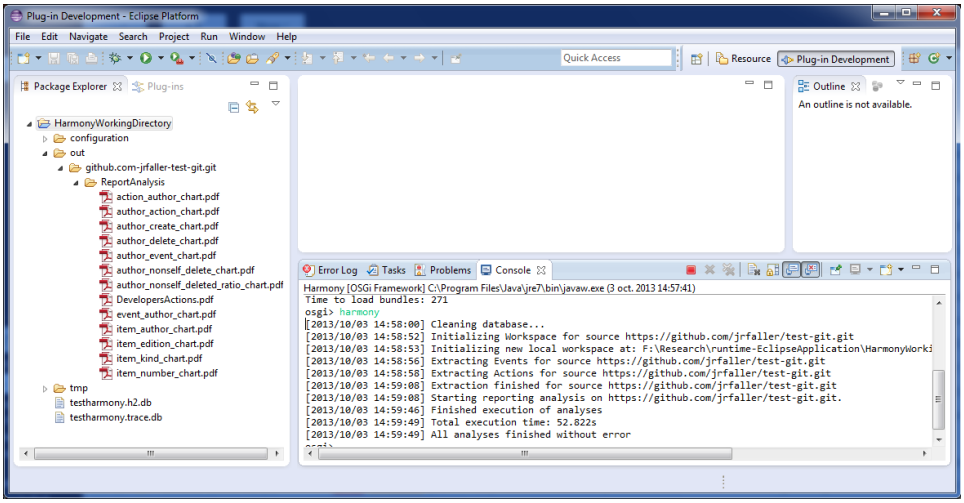

Figure 3.3: Eclipse after the first run of Harmony

Chapter 4

Configuration

This chapter introduces the configuration of Harmony for advanced use : selection of sources and analyses, database configuration, parallelism.

4.1 Configuration files

The configuration of Harmony is achieved with two configurations files: the global configuration file presented in section 4.2 and the source configuration file presented in section 4.3. When you launch Harmony without arguments like this:

```
harmony
```

default configurations files are used in this case. You may found them under:

```
/configuration/fr.labri.harmony/
```

They are named *default-global-config.json* and *default-source-config.json*. You can tell Harmony that you want to use other configuration files by indicating the path of the files like this:

```
harmony ./my-global-config.json ./my-source-config.json
```

4.2 Global configuration

In this section we will look at the parameters of the JSON files used for the global configuration. By convention the name of these files end by 'global-config.json'. The listing 4.1 shows the default file for the global configuration.

```
{
  "database":{
    "url":"jdbc:h2:test",
    "user":"sa",
    "password":"",
    "driver":"org.h2.Driver",
    "clean-database":true
  },
  "scheduler":{
    "threads":1,
    "timeout":100000
  },
  "source-analyses":[
    {
      "class":"ReportAnalysis",
      "require-harmony-model":true,
      "require-actions":true
    }
  ]
}
```

Listing 4.1: Default JSON file for the global configuration

4.2.1 Database

As shown in the listing 4.1, Harmony uses by default the H2 database engine (`http://www.h2database.com/`) in its embedded form. Even though this configuration is sufficient for testing we recommend to use a database server for running large studies with Harmony. In addition to H2, Harmony can use a MySQL server, the Community Server Edition for example: `http://dev.mysql.com/downloads/mysql/`. Once the MySQL server is installed and running, you will need to modify your global config file to set up the connection with the server. The listing 4.2 presents an example of such configuration.

```
  "database":{
    "url":"jdbc:mysql://localhost:3306/",
    "user":"root",
    "password":"strongPassword",
    "driver":"com.mysql.jdbc.Driver"
  }
```

Listing 4.2: Example of database configuration for a MySQL server

The *clean-database* parameter indicates if you want to delete all the existing Harmony model stored in the database before launching the analyses. If not explicitly specified, the *clean-database* is set at true.

4.2.2 Parallelism

Harmony uses Java threads for launching the analyses. Thus it can exploit shared memory architectures such as multicore architectures. You just have to modify the number of threads you want to use (see listing 4.3) in the execution section. Only one thread is used by source in order to minimize concurrent access to the same resources. Hence you will need multiple source to benefit from parallelism. The *timeout* parameter indicates in minutes the maximum execution time allocated to a thread for executing all the analyses asked on its associated source.

```
"execution":{
    "threads":4,
    "timeout":108
}
```

Listing 4.3: Example of parallel execution configuration

4.2.3 Analyses

Several analyses can be executed on the same source, these analyses can even be chained and share data. This aspect will be further discussed in chapter 5. Harmony possesses its own scheduler so you just have do declare the list of analyses as described in listing 4.4 and Harmony will take care of executing them in a correct order.

```
"source-analyses":[
    {
      "class":"ReportAnalysis",
      "require-actions":true
    },
    {
      "class":"ClocAnalysis",
      "require-actions":false
    }
  ]
```

Listing 4.4: Example of multiple analyses configuration

The construction of the complete Harmony model can be quite costly and some analyses do not require this complete model. The *require-actions* parameter aims to tackle this problem by telling Harmony if it necessary to build the actions in the Harmony model (see Chapter 5 for more information).

By default Harmony is shipped with several analyses:

- Reporting

- ScanLib
- SourceAnalyzer

4.3 Source configuration

In this section we present the parameters of the JSON files used for the configuration of the sources. By convention the name of these files end by 'source-config.json'.

The listing 4.5 shows an example of configuration file with three sources. You can add as many source as your study requires especially if you wan to exploit the parallel architecture of your computer.

```
[
  {
    "url":"https://github.com/isylhdin/PED_MaisonHote.git",
    "class":"JGitSourceExtractor"
  },
  {
    "url":"https://github.com/twitter/bootstrap.git",
    "class":"JGitSourceExtractor"
  },
  {
    "url":"http://macchiato.fr/svn/",
    "class":"SvnKitSourceExtractor",
    "user":"myself",
    "password":"123456789"
  }
]
```

Listing 4.5: Example of configuration file for the sources

For each source specified using the *url* attribute you must indicate an extractor that is compliant with it (*class* attribute). You can develop your own but Harmony already provides a fair number of source extractor :

- *JGitSourceExtractor* for Git repositories. This extractor is based on JGit, a Java implementation of Git.
- *GitSourceExtractor* for Git repositories. The Git native client must be installed on your machine to use this extractor.
- *SvnKitSourceExtractor* for SVN repositories.
- *Hg4jSourceExtractor* for Mercurial repositories.
- *BugzillaSourceExtractor* for Bugzilla repositories.
- *TFSSourceExtractor* for Team Foundation repositories.

Additionally if your source requires an authentication, you can use the *user* and *password* attributes as shown in the last source declaration of the listing 4.5.

Chapter 5

Develop your own analysis

This chapter explains step-by-step how to develop and run your own Harmony analyses.

Screencast available for this procedure:
`http://youtu.be/iQCeLmu2WDg`

5.1 New analysis wizard

- Go to `File` → `New` → `Project...`
- Select `Analysis` under the `Harmony` category and click on `Next`
- Fill out the form (see Figure 5.1)
  - *Project name*: this name must follow the Java package naming convention as the project name will be used as your root package as well as your plugin unique identifier.
  - *Analysis class name*: name of the class that will implement the analysis.
  - *Database facilities*: we will not need that for the moment.
- Click on `Finish`

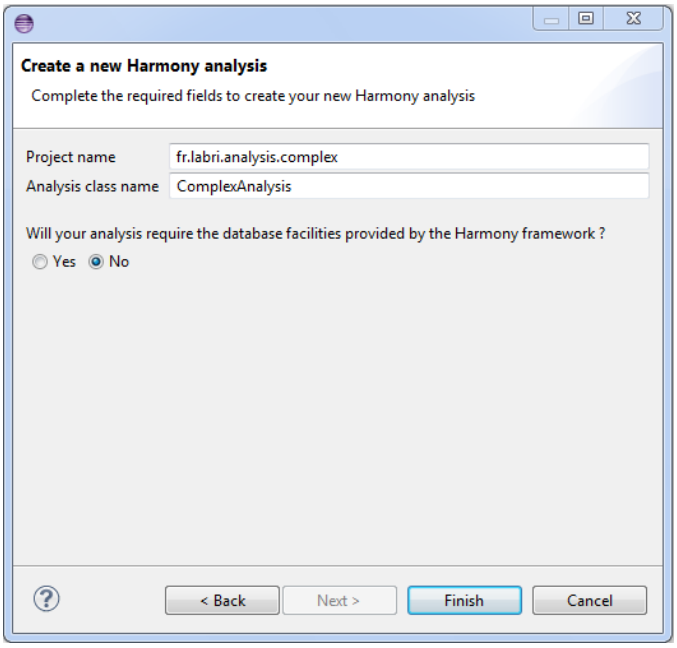

Figure 5.1: Form for creating a new analysis

5.2 Implement your analysis

- Implement your analysis in the method *runOn(Source src)* located in your analysis class.
- Modify the general configuration file : *default-global-configuration* to indicate the name of your new analysis (see Chapter 4 for more explanation).
- Run the OSGI console based on the Harmony *Run Configuration* (see Figure 5.2)
- Type `harmony` in the OSGi console and press Enter.

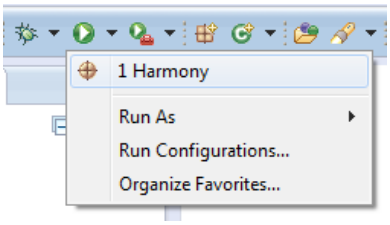

Figure 5.2: Harmony launching

Chapter 6

Advanced Use

This chapter gathers several topics that are reserved to advanced user of Harmony.

6.1 Starting from scratch

If for some inexplicable reason you want to create your analysis project from scratch, this section will guide you. The first step is to create a plug-in project for this analysis. Make a right-click on the *Project Explorer* view and select `New` → `Project` → `Plug-in Project` and *Next*. You a wizard similar to the one presented in figure 6.1. For demonstration purpose we will call it `dummy`. Set the target platform to use the Equinox OSGi framework and press *Next*.

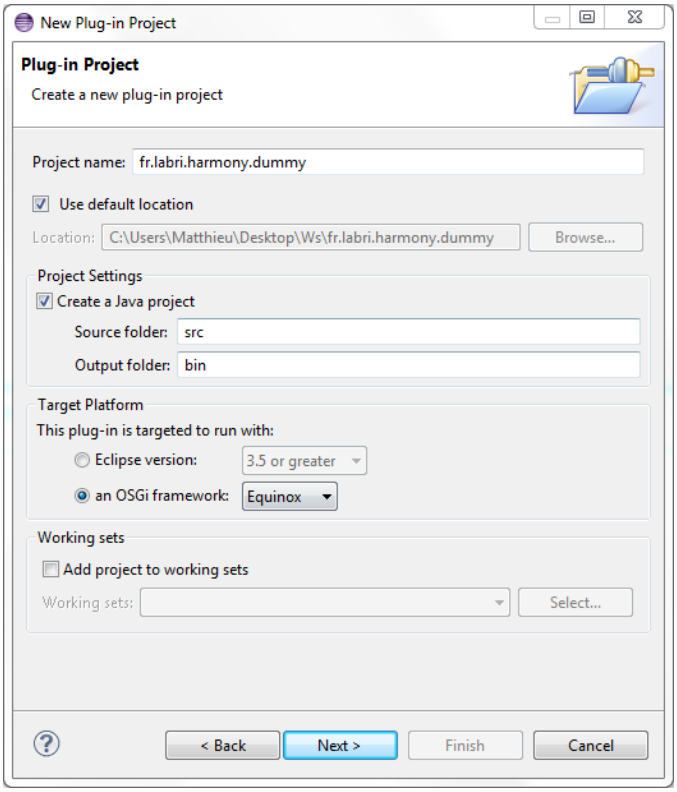

Figure 6.1: First page of the plug-in wizard

Press next. You should now see the `Content` form as shown in figure 6.2, uncheck the `Generate an activator...` checkbox. Finally press *Finish* to create your new project.

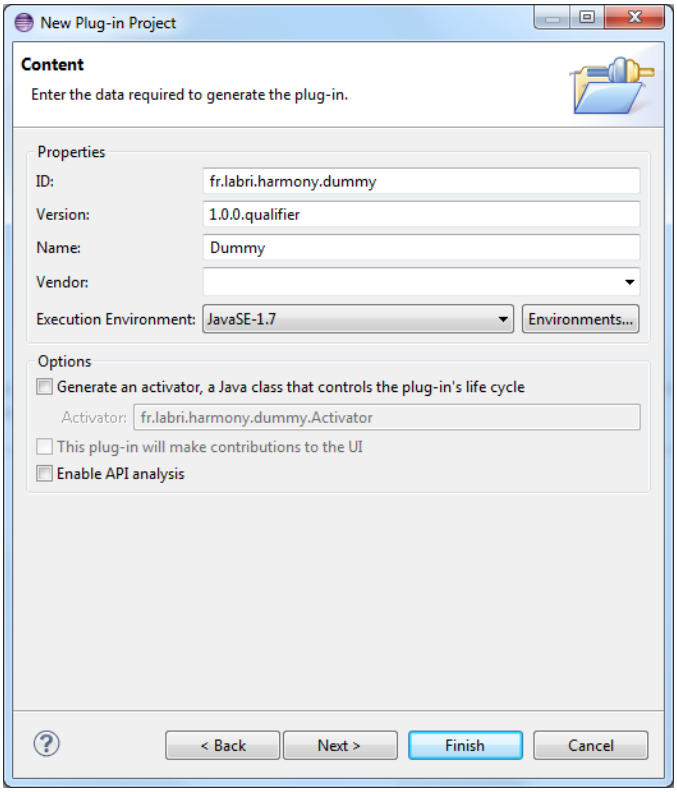

Figure 6.2: First page of the plug-in wizard

In order for your analysis to use the harmony framework, you need to add the corresponding dependencies to your plug-in configuration. If it is not already the case open the `MANIFEST.MF` file located in the `META-INF` folder. Open the `Dependencies` tab, and add the following imported packages:

```
fr.labri.harmony.core
fr.labri.harmony.core.analysis
fr.labri.harmony.core.config.model
fr.labri.harmony.core.dao
fr.labri.harmony.core.log
fr.labri.harmony.core.model
javax.persistence
```

You should obtain a configuration similar to the one presented in figure 6.3

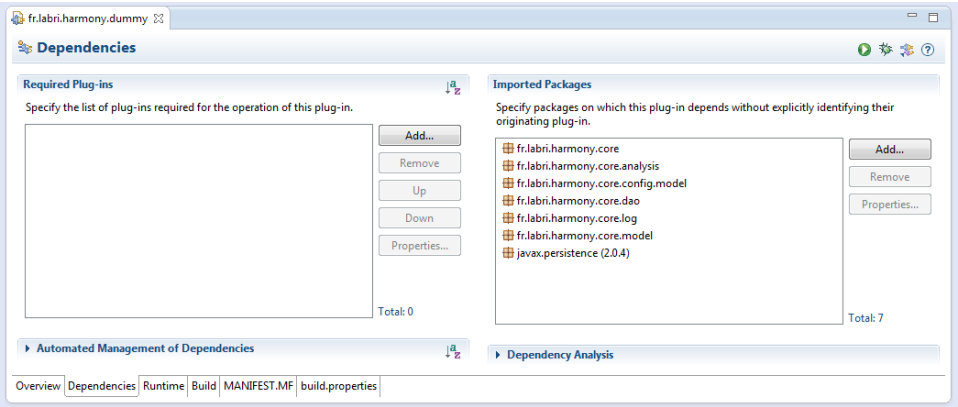

Figure 6.3: First page of the plug-in wizard

You now have to add your analysis main class that will implement the *AbstractAnalysis* class defined in the Harmony Core. To do so, use the Eclipse *new class* wizard, and check the `Constructors from superclass` checkbox as shown in the figure 6.4.

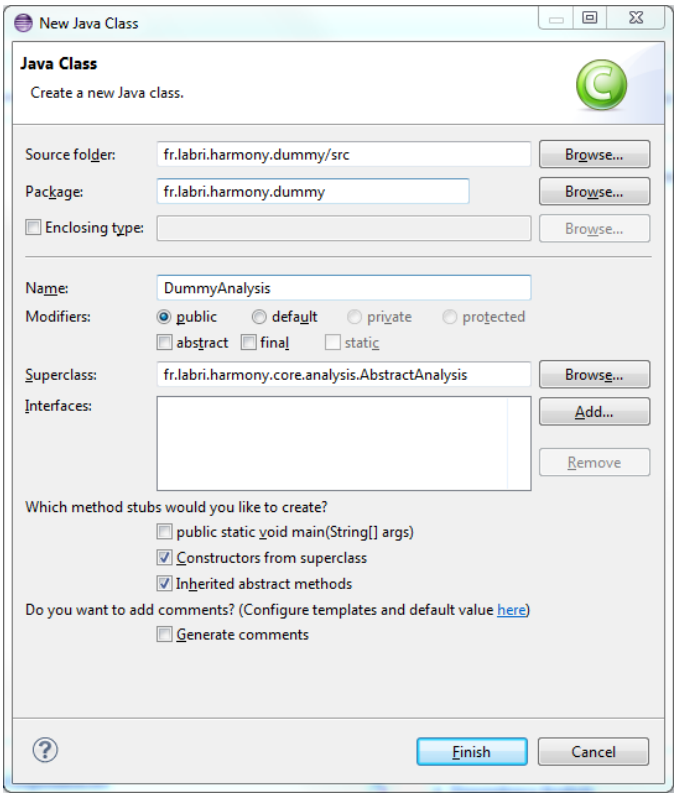

Figure 6.4: Creation of the main class of class of your analysis

This is very important, as **both constructors from AbstractAnalysis are mandatory**. Also add as your superclass:
fr.labri.harmony.core.analysis.AbstractAnalysis
Press *Finish*. You should have something looking like the class presented in the listing 6.1. The method *runOn(Source src)* is where you must implement your analysis. Take a look at existing analyses for inspiration. Follow the same procedure as the one you have followed for importing the *fr.labri.harmony.core* (see section 3.2).

```
package fr.labri.harmony.dummy;

import java.util.Properties;

import fr.labri.harmony.core.analysis.AbstractAnalysis;
import fr.labri.harmony.core.config.model.AnalysisConfiguration
    ;
import fr.labri.harmony.core.dao.Dao;
import fr.labri.harmony.core.log.HarmonyLogger;
import fr.labri.harmony.core.model.Source;

public class DummyAnalysis extends AbstractAnalysis {

  public DummyAnalysis() {
    super();
  }

  public DummyAnalysis(AnalysisConfiguration config, Dao dao,
      Properties properties) {
    super(config, dao, properties);
  }

  @Override
  public void runOn(Source arg0) {
    HarmonyLogger.info("Hello, I am Dummy!");

  }

}
```

Listing 6.1: DummyAnalysis lass

You now have a class that implements the analysis service so you need to inform the OSGi runtime about your implementation. Create an empty folder called *OSGI-INF* at the root of your project. Then create a new component definition : make a right-click on your plug-in project and select `New` → `Component Definition`. Complete the form by taking example of the figure 6.5, for example the component definition should be in the newly created *OSGI-INF* folder. Finally press *Finish*.

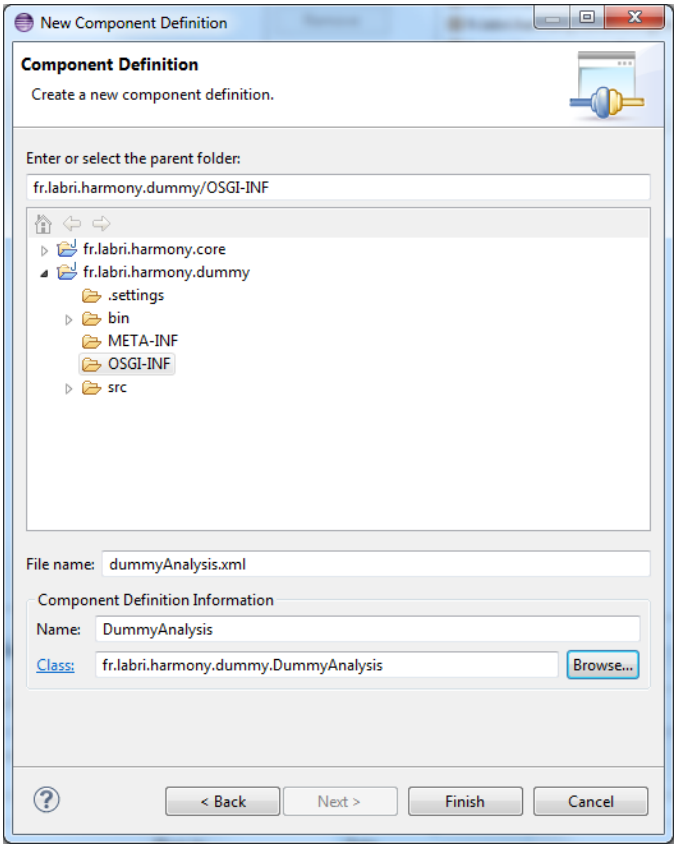

Figure 6.5: Wizard for creating a new component definition

Open the newly created file and go the *Service* tab as shown in the figure 6.6. Click on the *Add* button of the *Provided Services* section and select `fr.labri.harmony.core.analysis.Analysis`, press *Ok*.

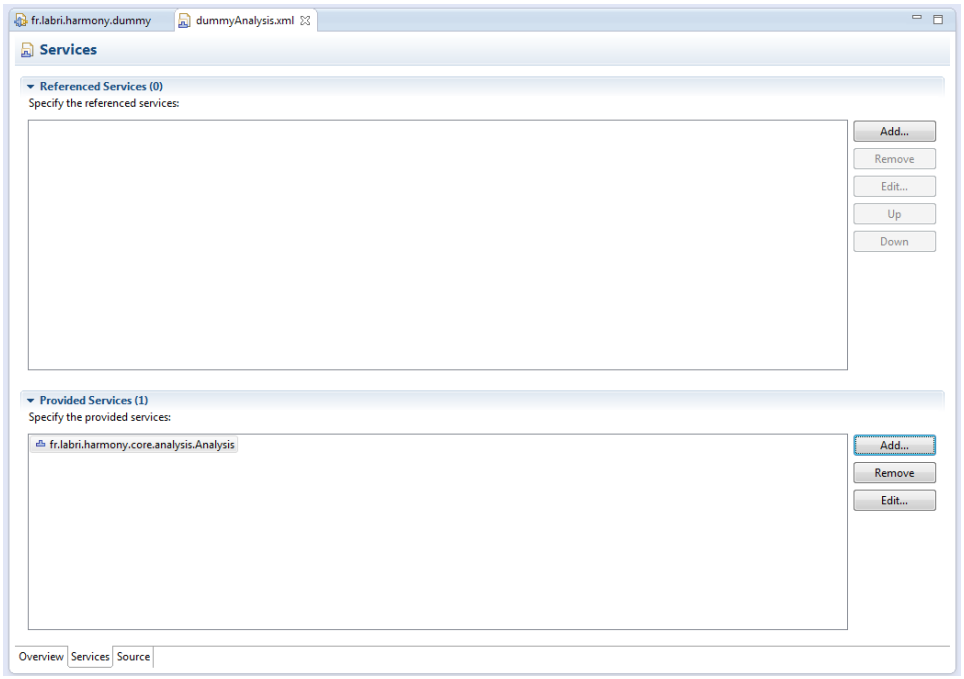

Figure 6.6: Provided services

Your analysis project is now set up. You can run it by modifying your configuration files (see Chapter 4) and your *Run Configuration* (see Section 3.2).