Comparing Two Jar Files for Similarities

Yan Gao
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, Canada
+1-613-523-4881
ygao@connect.carleton.ca

Hardik Patel
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, Canada
+1-613-255-6450
hpate2@connect.carleton.ca

ABSTRACT
In this paper, we describe a testing system to determining the percentage of similarity of two JAR files (or Java Archive). The system generates the digital fingerprints of two Jar files' under a particular algorithm, then reports the result of the comparison of the two fingerprints, which refers the similarity of the two Jar files. The fingerprint information is represented in the string form, so the comparison is based on string comparing method. The result is reported as a percentage of similar with some comments.

General Terms
Algorithms, measurement, documentation, performance, design, reliability, experimentation, security, standardization, theory, legal aspects, and verification

Keywords
Open Source Software, JAR file, digital fingerprint.

1. INTRODUCTION
As computer technique developing, the computer users can find various kinds of software. Comparing with some closed source software[1], open source software becomes more and more popular.
As its name, open source software's important source and recipe for making the software is opened to the users. Many users choose to use open source software not only because it is free, but also it is easy to make changes. In this way, some programmer can make some changes based on other open source software, and share with other users.

However, not all licenses allow users to use or modify the open source software, or some users share the software pretending as they wrote it. Under these conditions, it is necessary to check whether the two programs are the same or have the same part. [1]One way to check the two programs is by checking their JAR files. It works on the JAVA based programs.

JAR (or Java Archive) file is a file format packaging many files into one file; it has the ZIP file format. Since one JAR file can be distributed in the form of classes, metadata, and other resources such as text, images and so on, the software developers can extract one JAR file instead of getting many files to distribute the JAVA application or library. [2]

It is complex to compare two JAR files, since the JAR file is a package of many files. If the mechanism extracts two JAR files and compare each files, it need a lot of cases to consider and too many file types to handle. Sometimes, we define two programs are similar is not only based on the point they have the same code or contents, we may think more about the two programs' structures.
If a JAR file is unique, it must have some features which can distinguish from others, these features as the fingerprint of the JAR file, which is a digital fingerprint. Digital fingerprint is a unique coded string. It can identify a data file, and is generated by some mathematical algorithms. By such property, we can compare two JAR file by comparing their digital fingerprints. If two files have the same digital fingerprint, they are with 100% percentage of similar. In the converse, the digital fingerprint can be generated, and can be extracted to some information for comparing with other digital fingerprints.

The paper will describe this testing system to determine the similarity between two JAR files. It works based on comparing the two digital fingerprints, which contain the information of the JAR files. The fingerprint takes the manifest file information, how many folders and files in the Jar file, the class file's information, and the whole structure of the Jar file. All these are generated following a format string, then the system comparing the strings to know the similarity between the original JAR file and the duplicate one. After the step of comparing, the system reports the percentage of similarity between the two JAR files.

2. PROJECT COMPONENT
This project has three main components, which are JAR file, fingerprint generator detector and fingerprint result.

2.1 Jar File
Since a Jar file contains several types of files, when we extract a Jar file, we can see class files, java files, a Manifest file, maybe other types such as .txt files, xml files, etc. Some of them can be considered in this project, since they are involved in the comparison, such as the manifest file, the class files, and the java files. The other types of file do not really matter because it is hard to generate the information for those files and the information does not really matter to the similarity of the two Jar files comparison.

[1] Closed source software is developed by a single person or company, only release the final product, all the important source code or recipe for making the software is kept a secret.

For this project, the Jar files may be the Eclipse plug-in, or other application Jar files. But no matter the type of Jar files, the system deals them in the same way.

2.2 Fingerprint Generator Detector

The fingerprint generator detector is the most important part of this project. It works in the routine:

1. It takes an original Jar file, generate its digital fingerprint;
2. Then user passes another Jar file to it, the fingerprint generator detector generate the second file's digital fingerprint;
3. The fingerprint generator detector compares the two digital fingerprints, calculates the percentage of similarity, and then report result of comparison.

Basically the step of generating the digital fingerprint is an algorithm which includes two parts: one part is to conclude the information of the Jar file into a string, and the other part is to encode the original string into an unreadable form. These two methods will be described in the next section.

2.3 Fingerprint Object

In this project, the fingerprint object is defined by us. It contains three entities, which are ID, Jar file name and the encoding.
ID is the Identity of the fingerprint generator detector which defined by the programmer. It is used to determine which fingerprint generator detector is used for this fingerprint. So when to check two fingerprints are same or not, we should check the ID first, if the two Ids are same, it continues and if they are not same, that means the fingerprints generating under different algorithms, which means there is no use in comparing those two Jar files.
Jar file name determines which Jar file corresponds to the fingerprint. It can also be used for the fingerprint's ID since it is unnecessary to make an extra name for each fingerprint; it is easier just keep the Jar file's name to identify the fingerprint.

Encoding is a string which stores the information of the Jar file. It is generated by the fingerprint generator detector under some algorithms. It contains the information such as: number of files, number of folders, manifest file's information, and class files structure. All this information are gathered by the fingerprint generator detector, and encoded by a mathematical algorithm into an encrypted format. When the ID matches, the comparison is working based on this entity.

2.4 Fingerprint result

As its name, fingerprint result is a kind of report of the comparison of two valid fingerprints. It is generated by the fingerprint generator detector as well. One of its entities is the percentage, which is a float number range in 0 to 100. It determines how similar the two fingerprints are, and then we can know how many common properties the second Jar file has with the original one. The other entity is the comments, which records the comments of the comparison, it can be seen as a proof why we get the percentage number, and how it is calculated. It can be discarded since it is used to assist understanding the comparison steps, and it does not affect the result if it is not updated.

3. PROJECT DESIGN

The project is consists of four main steps. First, the system gets the useful information of the JAR file; then in the second step the system transfers the information into a tagged string, encrypts the string and stores it as the input JAR file's digital fingerprint; the third step is the comparison step, where tow fingerprints created in previous steps get compared; and in the last step the result of comparison is shown.

3.1 Getting Jar File Information

Most of the time, a JAR file contains a manifest file, the class files, and sometimes images, sound resources. For generating the digital fingerprint, the images and sound files are not that important, since they do no matter about the structure of the JAR file, thus such files are discarded from being included into fingerprint. What we are thinking about is the manifest file and the class files.

The first to be considered is the number of folders and files in the Jar file. Since the JAR file has the same format as the ZIP file, we can extract the JAR file using the same way as to extract a ZIP file. In JAVA, the easiest way to count the number of folders and files is to use the function "isDirectory()", and it is allowed to convert a JAR file to ZIP file. In this way, the JAR file is treated as a ZIP file, and enumerated all its entries, then the program checks each entry using the function "isDirectory()", if the answer is TRUE, the folder counter increases 1, otherwise the file counter plus one.

If a JAR file is executable, it should have a manifest file to determine how to use the JAR file. The manifest file is optimal existing depending on the JAR file is executable or not, but if there exists one, it is usually located in the path META-INF/MANIFEST.MF and declares the "main" class. [3] Since the information in the manifest file is used for executing the JAR file, if the two JAR files have the similar information, it should be similar in some way which can be use to determine if two JAR files are same or not. To access the manifest file, first we need to locate the file from all the JAR entries, to realize the function, we analyze the strings of the entry names, if the name string is ended with ".MF"; the file is the manifest file. The JAVA platform allows getting the manifest file and the program parse the manifest file's content as strings.

The third thing we can get from the JAR file is the class files information. To access the class files, we use the same method as to get the manifest file, checking the name which is ended with ".class". If the file is a class file, we can get some general information of it, such as the name, number of the methods and variables, enumerate the methods and variables, its hash code, source file, super class, and so on. From these data, we can get the structure of the class, if the two classes have the same structure, then they are the same.

3.2 Encoding

From the previous step, we get all information we need to generate the digital fingerprint. In this step, we need to encode the data into a format string.

Considering the comparing part, the data should be stored in an organized format. For ease of understanding, all data gets separated in following manner: all categories are separated with each other using the string "„„". In each category, we declare its name ended with a symbol "*" then followed the content of the category. For example, the folder number and the file number are stored in the string as the following style: "FOLDERS*x„„FILES*Y„„".

On the other hand, the data of manifest file and the data of class files are not that simple, they may contain more than one attributes, it is necessary to make each attribute clear. If the symbol "„„" is considered as the first level separator, then we need to make a second lever separator to make the attributes separate to each other. In this case, we use the symbol "^".

By using this method, we can group all the data into a string and separate easily. When the two fingerprints are compared, the system takes the data in the same category and checks the similarity. Generally speaking, the encoding step is to organize the data into an easy encoding and decoding format, the symbols to separate the information should be the symbols which never exists in the data, otherwise the system cannot get the correct data.

After encoding step, the system creates the JAR files fingerprint as a fingerprint object, marks the object the ID which defines the producer of the system, the JAR file's name and the string of the information as the content of the fingerprint.

3.3 Comparison

The comparison step is one of the main tasks of this project. Once the second JAR file is selected, the system generates its fingerprint, and then compares with the original JAR file's fingerprint automatically.

Basically, comparison is happening between two fingerprint objects. So first of all, the system should check the two fingerprints' ID. As mentioned in previous section, ID is used to determine the producer of the fingerprint. If the two IDs are not same, that means the two fingerprints are not generated under the same method, there's no way to check the different format fingerprint. Since the system can only handle the fingerprint encoded under its own method, if a fingerprint's ID does not match to the system's ID, the program reports an error.

In the fingerprint object, there is an attribute of JAR file name. In this step, the JAR file name is not considered. Since sometimes, two JAR file may have the same name, but the contents are different.

If the IDs are valid for the system, we can compare the fingerprints of the two JAR files. First, the fingerprint strings are decoded into data. For this part, the program is using "StringTokenizer()" to separate the fingerprint into each category, if the categories, for example manifest and class, have some attributes or sub-categories, using the same way but different separator to refine the information. After the decoding step, we restore the JAR files original information which is needed to be compared.

In this project, there are three cases to be thought about.

First, the number of folders and the number of files get compared. If the two files are exactly same, these two numbers should be exactly same. But conclusion cannot be determined here since two different JAR files could have same number of files and folders. Thus, result of comparing these two numbers represents 10% weight each out of total weight.

Second case is to compare the manifest file. As we mentioned before, manifest file stores the information to execute the JAR file. We compare each attributes of the two manifest files, if the same attribute's contents are same, then the similarity percentage increases. In this part, if the two files are exactly same, the total similarity percentage increases 30%. For example, if there are n attributes in total, and m attributes are matching, the percentage should be (30*m/n) %. For this formula, if m=n, the percentage is 30%; if m=0, the percentage is 0%.

The third one is the class files information; it takes 50% of the total similarity percentage. In section 3.1, we get the information of each class, such that the name, the number of methods, number of variables, source file, super class. If two class files' five data are matching, we say they are the same. In this project, we check the second JAR files class files one by one, if in the first jar file, there is a class matching, the counter increase. So if we define the number of matching class files is m, and there are n class files in the first JAR file, the percentage should be calculated like: 50*m/n%.

4. RESULT

The project runs in Java platform, the result of the project is stored as an object named FingerprintResult. In this object, there is a float number named percentage, which represents the total similarity between the second JAR file to the original one. The number is added up by the three comparing parts, which are the number of folders and files, the manifest file similarity and the class files comparison. If the total number reaches 100%, the second JAR file is exact the copy of the original one.

There is another attribute named comments, which is an ArrayList storing strings. These strings are added along with the comparison step, used to explain the reason that the system gets that result at that step. It does not matter the final result, just for the additional explanations.

5. CONCLUSION

In this project, we produce a testing system to checking two JAR files' similarity, based on the string comparison. It generates the fingerprint of the input JAR files, and compares two fingerprints, reports the percentage of two fingerprints similarity which can represent two JAR files similarity.

So far the system just works on the comparing the basic JAR files' information. It still has some problem on the comparing step, for example, the manifest files have some common attributes, in this project, they are counted as the similarity, but for the more precise result, they should be discarding. Also for the class checking part, the JAR files' structure is also an important part in the comparing.

6. REFERENCES

[1] *Science in Africa*. (Jan. 2004), DOI= http://www.scienceinafrica.co.za/2004/january/software.htm.

[2] *JAR(file format)*. (Nov. 2010), DOI= http://en.wikipedia.org/wiki/JAR_(file_format).

[3] *Manifest file.* (Oct. 2010), DOI= http://en.wikipedia.org/wiki/Manifest_file