UiB | Dept. of Information Science and Media Studies
UiTø | Dept. of Computer Science
NTNU | Dept. of Computer and Information Science
Telenor | Telenor R&I

CAIM
CONTEXT-AWARE IMAGE MANAGEMENT

A Back-up system for BergenBy –

a Multi-Modal Image Database

Erik Parmann

October 2010

CAIM-UIB

TR #13

Contents

1 Introduction

During the summer of 2010 the database had a week of downtime, and during that time it became clear that we needed a better backup strategy. This program was made to remedy this.

1.1 Objectives

We wanted to be able to dump the essential data, such as images, keywords and objects, from the database to local files that we then can import again if that is needed. Because of limited time we only made the exporter, and leave an importer as further development.

2 Tools used

Eclipse (version 3.5.0): Eclipse is a graphical integrated development environment (IDE) for Java. Eclipse supports a wide array of plug-ins, making it expandable and very customizable. For this project we only use the subversion plugin.

Oracle SQL Developer (version 2.1.1): SQL Developer is a free, graphical tool for database development. It is used to test queries before they are added to the software.

3 Usage guide

The program is a simple command line program that dumps the database into two files, one for the images and one for the objects. The most basic usage of it is as follows

```
java -jar caimBackuper.jar
```

This will then ask the user if it is acceptable that it dumps the data to two files. These files are called respectively "objects-DATE.bak" and "images-DATE.bak", where date is the long representation of the current date. If the user accepts the program will start downloading first the objects and then the images.

Alternatively the user can call the program with two arguments, which will then be the files the backup is written to, the first argument being the files of the objects and the other beeing the file of the images. An example:

```
java -jar caimBackuper.jar objectFile05092010 imageFile05092010
```

4 Design and implementation

4.1 Design

The process logically consists of two main parts, fetching of the data and writing of the data to the files. The collection of images can become pretty big, so to avoid a potential out-of-memory exception the naive solution of first fetching the information to one big List and then writing that list to a file was not chosen. Instead we merge the two processes, and write the image data directly to the backup file as the data is downloaded.

The main classes are as follows

Backuper
This is the main class, and is responsible for the flow of the program. It also handles files.

DbDAO
This class interacts with the database, and handles all the communication with the database.

Data classes
There are 6 data classes that do nothing but store and structure data. These are "Author","BareObject","FullObject","Image","Keyword" and "Text".

4.1.1 Data stored

The data stored for the images are as follows:

```
byte[] imageData;
int imageId;
String imageTitle;
long latitude;
long longitude;
long timestamp;
List<Integer> objectIds;
List<Keyword> keywords;
```

while the data stored for the objects is as follows:

```
int id;
String name;
int age;
String material;
String longitude;
String latitude;
String area;
String classification;
Text text;
List<Keyword> keywords;
byte[] audioData;
```

4.2 Implementation details

4.2.1 Binary format

We use the built-in serialization options in java to serialize the data to a ObjectOutputStream. We then build objects in java with the needed data that implements the “Serializable” interface, and as long as they contain only serializable data the objects are serializable. All primitive types in java are serializable. Unfortunately the object types that Oracle uses to store images and audio are not serializable, so to serialize these we have to extract the important information (being the byte array that contains the image/audio).

An important note is that first in the file we write an integer. This integer is the number of objects/images in the file. After this follows that number of objects.

An example decoder is included in the class Backuper as the method “getObjectsFromFile”, that returns a list of objects from the file.

4.2.2 Flow of execution

The flow for both the writing of images and objects goes as follows. First we setup the file to write to and the ObjectOutputStream attached to it. This stream is then sent to the method in DbDAO responsible for fetching the data from the database. This method then writes the data to the stream, and when it is done we close the streams. This means that the class DbDao that interacts with the database also does the writing to the file. It does though not “know” that it is a file it writes to, it only writes the objects to the passed ObjectOutputStream. It is the callers responsibility to do the file handling.

5 Further development

Importer

If we ever need to use the backup we will need an importer. This needs to read the data from the files (an example of how this can be done follows this revision) and write that data to the database.

Automatic backup

It could be a good idea to setup the program to run automatically every x days, and then delete old backups as they fill up.

Follow the database evolution

As the database structure changes this program must as well. If attributes are added or removed from objects/images, or other essential information is added this program must be updated.