CADI development manual

(version beta)

GICI group

Department of Information and Communications Engineering

Universitat Autònoma Barcelona

http://www.gici.uab.es - http://www.gici.uab.es/CADI

September 2007

The goal of this document is to provide a quick guide to developers that need to modify CADI application in some sense. An overview of the CADI design and the directory structures as well as the basic compilation instructions is explained. However, the detailed information about data structures, algorithms and classes is documented in the CADI API.

1 Overview

The main motivation in the design and development of CADI is to generate a completely modularized scheme where each module works independently. In order to understand it better, all modules have the same skeleton and only basic programming language tools are used. The main advantage of these independent modules is that one module can be replaced without compromising the others, easing the testing of new ideas, the extension on some operations, and even the replacement of some coding operations.

All compression stages are divided in simple modules that interact among them. Each module is programmed independently and has its own parameters. Replacement of some coding operations, testing new ideas, debugging tasks and even extending coding operations are an easy task with CADI. The interaction between modules is performed with structures, passed among classes. The coder manages these structures and controls their destruction.

2 Files & directories

To organize all the classes and files, CADI is organized using directories. The root directory contains the following items:

- **build**: object files used in the compilation (.class files)
- **dist**: the encapsulated coder and decoder in .jar files (CADIServer.jar and CADIViewer.jar)
- **docs**: CADI manuals (installation, user and development) and the api documentation
- **workDir**: a temporal working dir that can be used by users
- **src**: all the source codes are contained in this directory. It contains all the external libraries used in BOI (developed by the GICI group and used in other implementations) and the source code of CADI. The libraries are in the directories (GiciAnalysis, GiciException, GiciFile, GiciImageExtension, GiciStream and GiciTransform) and the sources in CADI directory.

As we can see in the CADI source directory contains the coder and the decoder sources separately. All the source code is structured in small classes that are called from the server or viewer (*src/CADI/CADIServer.java* or *src/CADI/CADIViewer.java*). Each one of these small classes performs simple operations and all of them use the same skeleton structure, easing its comprehension and modification. In addition, for each class the execution processes are the same: first, they perform initializations, then a parametrization can be used, and last the run procedure executes module actions.

3 Compilation and execution

The ant tool is used to compile the source files. This tool is configured with the file *build.xml* located in the root directory and contains all the compilation instructions. Executing *ant compile* in this root directory all the sources contained in the *src/* directory will be compiled and the .class files that this process generates will be located at the *build/* directory. If compilation does not generate any error the .jar files will be generated at the *dist/* directory. This jar files can be executed using the shell scripts *CADIServer* and *CADIViewer* or using the command lines *java -jar dist/CADIViewer.jar* and *java -jar dist/CADIViewer.jar* (see the user manual for more information about how to run CADI).

To clean all directories of garbage and swap files you can run *ant run*. Moreover, the whole source code is well commented to facilitate its understanding and modifications. *ant doc* generates the api documentation. This command creates the *docs/api* directory with html that can be viewed with and standard browser.

Add, replace or even remove source files does not effect the compilation process. However, if you need to create or remove the directory structure you will have to modify the *build.xml* file in order to indicate your changes to the compilation process. However, manual compilation with the *javac* command is also possible and the compilation options are also contained in the *build.xml* file.