Enrico Biermann (enrico@cs.tu-berlin.de)
Timo Glaser (timog@cs.tu-berlin.de)
Marco Kunze (makunze@cs.tu-berlin.de)
Sebastian Nowozin (nowozin@cs.tu-berlin.de)

WS 2002/03
18. 2. 2003

YAVA - Programmers Overview

1 Introduction

This is a short primer on how the YAVA program is structured from the developers point of view. You should read this document to understand the overall design of the YAVA program and when you are about to hack the source. For an detailed in-depth view to the source and its individual classes and methods, please consult the source-generated "Programmers Reference Manual" [1] instead.

2 Overview

The program is split into four modules, where two parts - the GUI and vowel modules - are limited to the application domain, and the remaining parts - the audio file processing module and the neural network module are independant and reuseable. Within the program, the modules are dependant on each other as shown in figure 1.

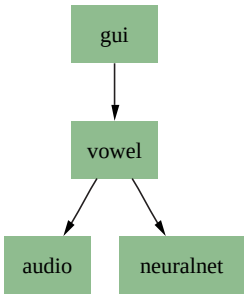

Figure 1: Module dependency overview

3 Module Descriptions

The modules are organized by directories within the main `src/` source directory. The tree is as follows:

```
src/
   audio/        libaudio.a, classes AUFile and AudioPreprocessing
   gui/          main program, GUI interface and custom GUI widgets
   neuralnet/    libperceptronnetwork.a, main class PerceptronNetwork
   vowel/        libvowel.a, abstraction classes, main class VowelManager
```

3.1 Audio

The audio module abstracts away file format dependant details and audio processing algorithms. The `AUFile` class keeps all the fancy details of the Sun AU sound file format under a simple to use interface. While loading and saving are the core functionalities, it provides a handful of additional utility methods to conveniently deal with raw sample data, such as finding a limited maximized sample range within the overall sample or storing arbitrary non-sample payload data within an AU output file.

The `AudioPreprocessing` class builds on top of the foundation provided by the `AUFile` class. Taking a raw audio sample and processing parameters as input, it applies a number of simple audio processing algorithms, such as FFT fast fourier transformation and normalization to provide a discrete low-dimension output vector representation of the sample. Also, serialization of the parameters is provided by both a `load` and `save` method. During runtime, one `AudioPreprocessing` object is kept as a quasi-template, which is copied whenever a sample is processed. The settings of this template are modified by the GUI's audio tab.

The individual modules are described below.

3.2 Neural Network

The neural network module implements a multilayer perceptron neural network system. The module is build around a main class called `PerceptronNetwork`, which is probably the only part one would use from outside the class. Internally the `PerceptronNetwork` class builds a hierarchy down to a `PerceptronLayer` and `PerceptronNeuron` class, naturally modelling the systems layout by means of object orientied programming. The helper classes `RandomFunctions` and `ActivationFunctions` are provided to allow a more convenient initialization of the network parameters.

3.3 Vowel

The vowel module is the glue between the GUI and all underlying classes. It provides the real vowel sample recognition capabilities, the abstract classification system and its training and testing functionality. There are only few parts the GUI interacts with that are not kept here, so if you are about to really modify the application domain specific behaviour, this is the place to look. The main class is the `VowelManager` which makes use of the `VowelSample` and `VowelSet` to manage the training and testing set of vowels. The `VowelClassifier` class represents the expert system based on three neural networks. It is trained by the manager, using its training set, and its quality is verified with the test set of samples. One global `VowelManager` object is kept for the program, which is kept in `Yava.cpp`. All parts of the `VowelManager` object are accessible through the GUI.

3.4 GUI

The GUI is made up be the main program `Yava.cpp`, an automatically generated class `GUIMain`, a GUI interface glue class `GUIMainImpl`, which inherits from `GUIMain` and finally the custom widgets `Graph` and `WeightMatrix`. The GUI itself is created using the QT Designer, which in turn generates `.ui` files. During the build process, the files are converted by the `uic` utility to auto-implemented classes, from which the manually written implementation class `GUIMainImpl` inherits. The events and signal-slot relationships are already defined within the designer, so all whats left to the real programmer is implementing the actual code behind the widget action. The GUI is organized in five tabs, each reflecting a functionally related part of the program. The tab categories are seperated within the source file for easier modification.

References

[1] pedantic project team, "YAVA Programmers Reference Manual"