University of Salzburg

Institute of Computer Science

Visualization of NetJEN problem domains and results

YARB

Yet Another Result Browser

Thomas Aschauer

Erich Eichberger

Thorsten Engl

Christian Zödl

Date: July 5, 2000

Version: 1.0

Abstract

NetJEN is a system for evaluating artificial neural networks and applied to a variety of different problem domains, ranging from stock exchange forecasts to classification of satellite images to extracted rules of ANNs. Now the difficulty when working with different problem domains is that there is currently no software tool that is able to visualize the network in- and output (also called learning, training and validation pattern).

This is where YARB takes place.
YARB a general purpose result browser written in Java for nearly every possible problem domain. This is realized by the special, unique and expandable structure of this great software package.

This paper describes the whole development cycle, from requirements analysis to design till implementation. Further a guide for custom expansion and configuration of YARB is included.

Contents

List of Figures

Chapter 1

Introduction

This chapter gives an overview of the problem our approach.

1.1 The problem

Imagine you have got an neural network simulator and you want to do some stock rate prediction. So you have a chart, representing the stock rate, and you want to have a graphic, which displays the prediction of the rate. You have to write a tool that is able to show the chart. But what, if you want do do some other things with the simulator for NNs[1], like character recognition, and display the results? For these other tasks you have to develop similar tools which do the visualization for each problem.

1.2 The project

The project is going to be a graphical tool, written in Java, for the visualization of netJEN problem domains. The aim is to show the pattern file[2] before and after netJEN evaluated it (they are called learning and result pattern). The pattern files are in the pattern file format of the SNNS [3]. Thus the program has to be able to read this files and display the information contained in a manner that depends on the problem domain. This, perhaps, could be an image from a satellite that should be classified or prognosis of stock rates.

1.3 Titling

Since we started working on this project we have been looking for a name. After hours of brain storming we finally found one:
YARB [4]

[1]Short form of **N**eural **N**etwork, sometimes used as synonym for ANN (**A**rtificial **N**eural **N**etwork)
[2]The data neural network simulators work with is called pattern
[3]**S**tuttgart **N**eural **N**etwork **S**imulator
[4]**Y**et **A**nother **R**esult **B**rowser

1.4 Where are NNs used?

Neural Networks are very often used in industry nowadays, and there is a wide spread of applications where the use of NNs could bring big advantages. So it is useful to examine where NNs are already used today. Additionally, we have tried to determine how the problem could be visualized and in which problem domain it could fit.

1.4.1 Applications for NNs

Manufacturing

Literature: [1], [2], [3]

- process optimizing
  Visualization: Graph
  Problem Domain: Optimization, Time series prediction, Parameter determination
- process monitoring and controlling
  Visualization: 3D - Chart
  Problem Domain: Signal analysis and prediction
- determining operational policies
  Visualization: Graph
  Problem Domain: Parameter determination
- predictive control of quality
  Visualization: 2D - Chart
  Problem Domain: Signal analysis and prediction, Time series prediction
- flexible manufacturing scheduling
  Visualization: Graph
  Problem Domain: Rule extraction, Time series prediction

Chemical Industry

Literature: [1], [4]

- sensor analysis
  Visualization: Line - Chart
  Problem Domain: Signal recognition
- spectroscopy
  Visualization: Bar - Chart
  Problem Domain: Signal recognition, Classification
- molecular modeling
  Visualization: 3d - Model
  Problem Domain: Modeling

- data rectification for dynamic processes
  Visualization: 2D / 3D - Chart
  Problem Domain: Signal recognition, Time series prediction

- process modeling for fault detection
  Visualization: Graph / 3D - Chart
  Problem Domain: Rule extraction, Optimization

- statistical analysis
  Visualization: 2D / 3D - Chart
  Problem Domain: Time series prediction

Medicine

Literature: [1], [5]

- diagnosis of coronary artery disease
  Visualization: Graph / Chart
  Problem Domain: Classification

- structure analysis of virus proteins
  Visualization: 3D - Model
  Problem Domain: Parameter determination

- diagnosis of epilepsy
  Visualization: Graph / Chart
  Problem Domain: Classification

- rehabilitation of persons with incomplete spinal cord injury
  Visualization: Graph
  Problem Domain: Optimization, Rule extraction

- analysis and classification of EEG signals
  Visualization: Bar - Chart
  Problem Domain: Classification, Signal analysis and prediction

Nuclear engineering

Literature: [1], [6]

- spectral analysis
  Visualization: Bar - Chart
  Problem Domain: Signal recognition, Parameter determination

- signal prediction in nuclear power plants
  Visualization: Line - Chart
  Problem Domain: Signal analysis and prediction, Time series prediction

- classification of energy levels
  Visualization: Bar - Chart
  Problem Domain: Classification

- alarm handling in nuclear power plants
  Visualization: Line - Chart
  Problem Domain: Signal analysis and prediction, Time series prediction

Miscellaneous applications

Literature: [1], [7], [8], [3], [4], [5], [9], [6], [10], [2]

- classification of satellite pictures
  Visualization: Areas / 3D - Model
  Problem Domain: Classification
- sensor analysis
  Visualization: 2D / 3D - Chart
  Problem Domain: Signal recognition
- optical character recognition
  Visualization: Picture / Text
  Problem Domain: Classification, Signal recognition
- voice recognition
  Visualization: Line / Bar - Chart / Text
  Problem Domain: Signal recognition, Classification
- stock rate prognosis
  Visualization: Line - Chart
  Problem Domain: Time series prediction
- texture recognition
  Visualization: Bar - Chart / Text
  Problem Domain: Classification
- temperature regulation
  Visualization: Line - Chart
  Problem Domain: Signal analysis and prediction, Time series prediction, Rule extraction, Parameter determination
- sunspot evaluation
  Visualization: 2D / 3D - Chart
  Problem Domain: Time series prediction, Classification, Rule extraction
- business prognosis and optimization
  Visualization: Graph / 3D - Chart
  Problem Domain: Optimization, Time series prediction
- sale prognosis
  Visualization: 2D / 3D - Chart / Graph
  Problem Domain: Time series prediction

1.4.2 Problem domains

The next task was to itemize all of the problem domains we found.

- classification
- time series prediction
- parameter determination
- signal recognition
- signal analysis and prediction
- rule extraction
- optimization
- modeling
- ...

1.5 How to visualize these problem domains?

This section lists examples for possible visualization of many different problem domains.

1.5.1 Summary of possible visualizations

- picture data as bitmap
  (see Figure 1.1)
- classification as areas
  (see Figure 1.2)
- Various charts: Bar, Line, Circle, ...
  (see Figures 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9 and 1.10)
- 3D-graphics
  (see Figures 1.11, 1.12, 1.13 and 1.14)
- Frequency diagram
  (see Figure 1.15)
- graphs
  (see Figure 1.16)
- text
- plain pattern text

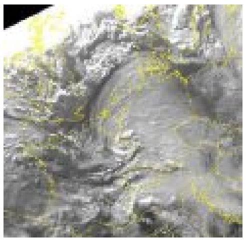

Figure 1.1: Example: Satellite bitmap

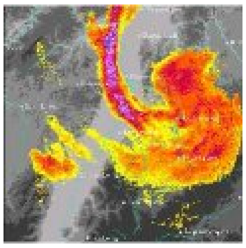

Figure 1.2: Example: Satellite bitmap with result areas

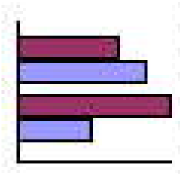

Figure 1.3: Example: Bar chart

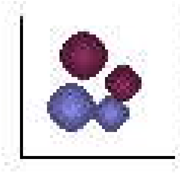

Figure 1.4: Example: Bubbles chart

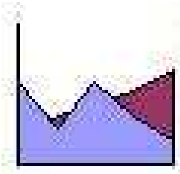

Figure 1.5: Example: Area chart

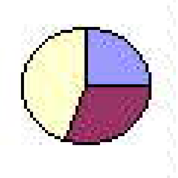

Figure 1.6: Example: Circle chart

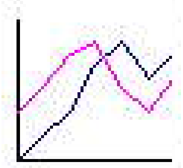

Figure 1.7: Example: Line chart

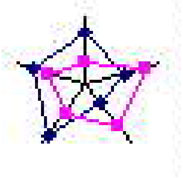

Figure 1.8: Example: Net chart

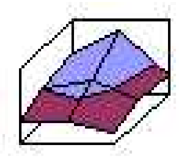

Figure 1.9: Example: Surface chart

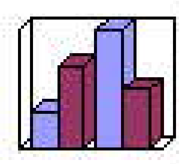

Figure 1.10: Example: Pillar chart

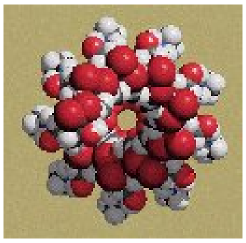

Figure 1.11: Example: Molecule

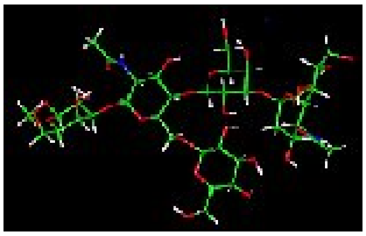

Figure 1.12: Example: Molecule Structure

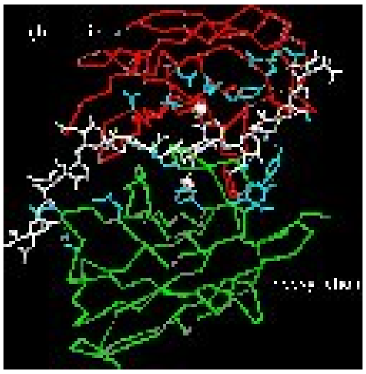

Figure 1.13: Example: Molecule Structure 2

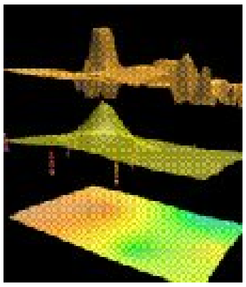

Figure 1.14: Example: Landscape

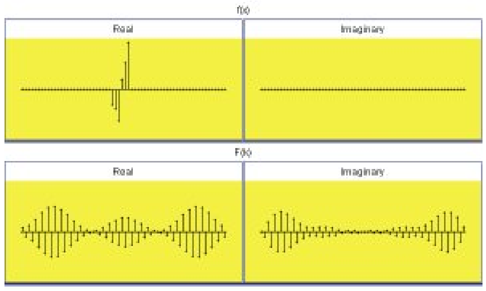

Figure 1.15: Example: Frequency diagram

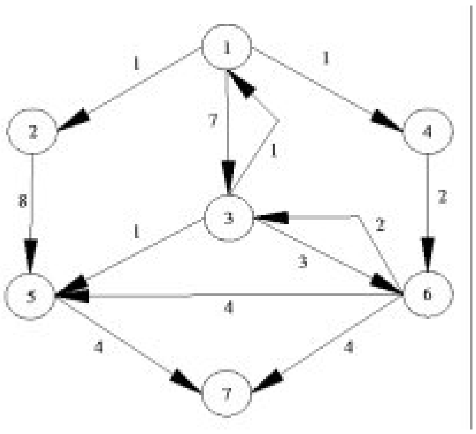

Figure 1.16: Example: Graph

1.5.2 Examples

We have collected some examples for visualizations. These examples are shown in figures 1.1,1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 1.10, 1.11, 1.12, 1.13,1.14, 1.15 and 1.16.

1.6 How YARB uses NetJENs Files

1.6.1 General file concepts in NetJEN

If you want to use NetJEN you have to format your problem into pattern files. These pattern files are used by NetJEN, the generated output is saved to the so called result files, which are having the same file format.
During the learning of the ANN the training pattern files are used to describe the input data and the corresponding output. After this learning phase has completed, the test/validation pattern files are used to feed the ANN with input data. The output of NetJEN is written into result files.

All files used are in SNNS-file formats (see [11]).

1.6.2 General file concepts in YARB

Because YARB is a browser for pattern and result files, the same files are used.
In some problem domains the raw input data has many redundant information. In these cases only the essential part of the data is converted to a pattern file. If the users wants the whole information from the input to be displayed, a second file is needed, GIF, JPEG or another pattern file.

1.7 Custom Configuration of YARB

While analyzing the problem we figured out that for some problem domains its very important to customize the appearance of the visualization (Scale, Zoom, ...). To prevent the user to set the display properties again and again we introduced the concept of configuration files. These files contain all the information needed to display pattern- and result files in a postulated way.

Chapter 2

Project specification

This chapter describes the specification this project should match.

2.1 Main Appointment

The user should be able to get different visualizations of various problem domains.

2.1.1 Mandatory criteria

- Visualize pattern in SNNS pattern files
- Visualize pattern in SNNS result files
- Choose between visualization options
- Customize the visualization options

2.1.2 Additional criteria

- Easy to implement new types of visualization
- The ability to be called from other programs (RMI [1])
  (This feature would be nice to have, but is still open ...)

2.1.3 Delimitation

- No modifying features of SNNS files

2.2 Product use

2.2.1 Areas of application

- All possible problem domains

[1] **R**emote **M**ethod **I**nvocation

2.2.2 Target group

- NetJEN users

2.3 Operational environment

2.3.1 Software

- All platforms supporting Java

2.3.2 Orgware

- Eventual systems that support RMI

2.3.3 Product interfaces

- Ability to read SNNS files
- Ability to read JPEG and GIF files
- Eventual RMI

2.4 Product functions

- Open pattern and result files
- Visualize data by manually choose type
- Visualize data by using a wizard
- Visualize picture data (GIF, JPEG)
- Change visualization configuration
  - Scaling
  - Axis titles
  - ...
- Save visualization configuration
- Load visualization configuration
- Change visualization type

2.5 Product data

- Configuration files

2.6 Product performance

Not specified

2.7 User interface

- Standard Java GUI with Swing components
- Tree view of possible problem domains and their visualization
- Every file to be displayed in a single MDI [2] window

2.8 Development environment

- OTW [3] for UML [4] designs
- Forte for Java as IDE [5]

[2]**M**ultiple **D**ocument **I**nterface
[3]**O**bjekt **T**echnologie **W**erkbank
[4]**U**nified **M**odeling **L**anguage
[5]**I**ntegrated **D**evelopment **E**nvironment

Chapter 3

Software design

Literature:

- General software analysis and design: [12, pages 741-822]
- UML: [13]
- Software pattern (Design pattern): [14], [15], [16]

3.1 General design aims

While we were designing YARB we had to consider the following subjects:

- Flexibility in visualization and transformation of visualization data
- Maintain ability
- To be dynamically extendable for further problem domains

To gain this aims it was necessary to use bullet proof design pattern and up to date design processes.

3.2 Use Cases

The first step to perform was to identify the use cases.

3.2.1 YARB - User - NetJEN interaction

The most common use case for YARB was the interaction of YARB with users and the NetJEN program. You can see this in figure 3.1. Here the actors are the YARB, NetJEN and the user. As you can see, the user and NetJEN are the active parts.

3.2.2 General User interaction

The refinement of use case 3.1 to the user interaction with YARB is the use case 3.2. In this figure the main action of an user are displayed.

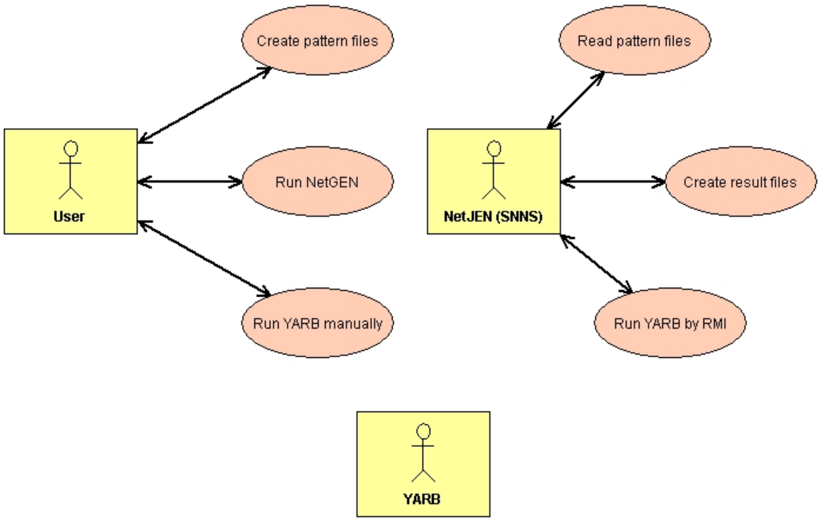

Figure 3.1: UCD: Interaction of User, NetJEN and YARB

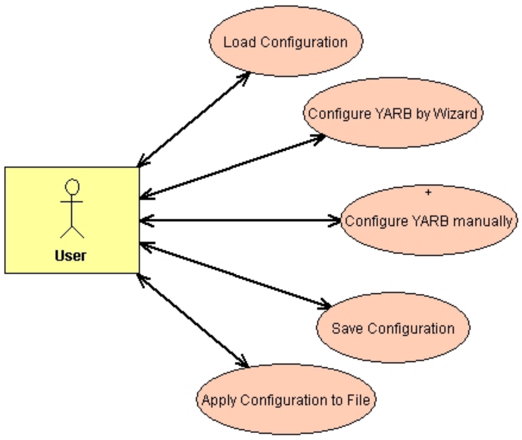

Figure 3.2: UCD: Run YARB manually

3.2.3 User interaction: configure YARB

But not only the main actions had to be figured out, also the sub actions of configuring YARB had to be identified. The result of this finest identification process can be seen in figure 3.3.

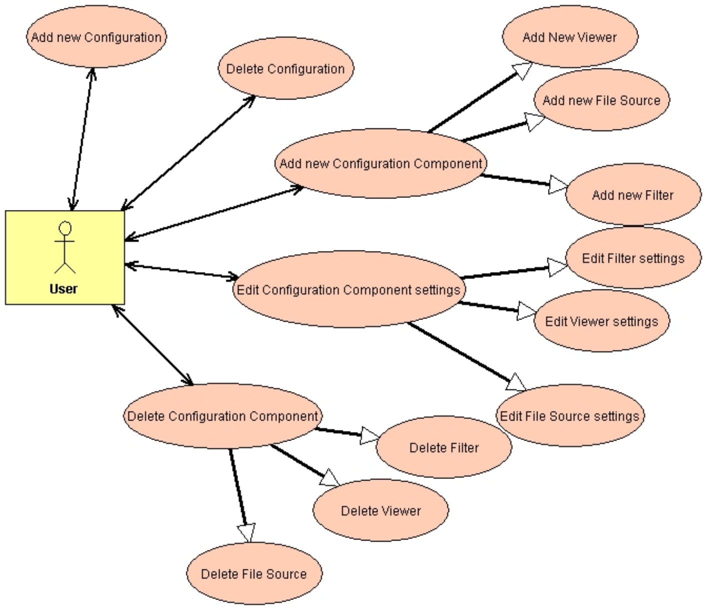

Figure 3.3: UCD: Configure YARB manually

3.2.4 YARB and RMI

Last, but not least the interaction of YARB and NetJen had to be designed. As shown in figure 3.4, this is the smallest use case, but not the easiest to implement.

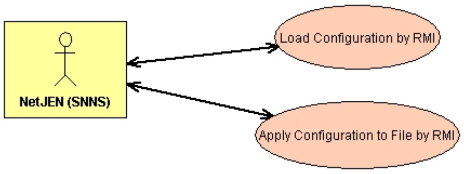

Figure 3.4: UCD: Run YARB by RMI

3.3 Designing the YARB structure

Out of the use cases we had to design the basic structure of YARB. This was not a trivial task and therefore here is the description of our main design decisions.

3.3.1 Design of the data processing flow

One of the big design aim was to provide flexible data processing for different kinds of visualization. As one of he design pattern that best fitted was the "Pipes and Filters" architecture pattern. A brief description of this pattern can be found in appendix A.1.

We splitted the data processing flow to different components that works an the YARB data processing pipeline. The pipeline input data (data source) is provided by the file sources. The data transformation task is done by various filters that may be linked together in a chain. The data viewers are responsible for the visualization task (data sink). The communication and pipeline infrastructure is provided by a so called configuration container.

Error Handling

The weak spot of the pipes and filters pattern is the error handling. As there can be seen on Unix pipes (a mutant of this design pattern), error handling and detection easily becomes an arduously task.

In the YARB architecture the pipeline structure is provided by the configuration container. This structure was designed in a way that it is possible to implement a two-way error handling system:

The first way of handling errors is, like in most Java applications, to deal with *exceptions*. In the YARB interface IDataProvider (this interface is described in section 3.4.1) is the declaration of an exception class called DataProcessingError. This is the general exception class for critical errors that occur during the data processing task in the YARB pipeline. This exception will be first caught by the configuration container, and therefore the process will be abandoned. This error handling is used for critical errors like file not found, division by null, etc.

When there are minor errors, there is the guideline to give an *return value* of *false* and send an error description to the console output (via System.out.println). Each process component (file source, filter, viewer) will check the return values of its predecessor, and if it is false, it will return false too. But not immediately. First, it finishes its own data processing procedure. The configuration container will check the resulting return value, and in an error case, warns the user that there went something wrong.

3.3.2 Design of the Visualization process

The visualization process is done by the data viewers. To get the most flexibility out of the design we encapsulated the visualization process in two extra components, the viewer frame and the data holder. Both are created by the main viewer and work together in a "Document-View architecture". This is a simplified version of the famous "Model-View-Controller architecture".

3.3.3 Design of the user interface

It took a long time to figure out which user interface is most comfortable when working with YARB. We decided to build in the standard menu- and toolbar driven command processing UI, and to implement an additional tree view for managing the configurations.

But as it is ever when using different classes for doing the user interface, the UI-classes and their relations are very interwoven. The maintenance and extension of such classes will become a burdensome and fault-prone task. Therefore it was the logical consequence to look for something that helps doing it better. This was the time we looked for a design pattern and where we decided to choose the "Mediator". Appendix A.2 contains a brief description.

Now the interaction between the main frame (command user interface) and the navigator frame (configurations tree view) is encapsulated to a third object, the global navigation director.

3.3.4 Naming conventions

As in every project where more than one developer is working on, YARB needs naming conventions, too. But we designed our style guide to be not too limiting (because most programmers don't like to be told how to do their work). Although naming should be done by general good software engineering practices, the most common naming conventions are given here.

Interface Names

In the YARB design all interface classes have the prefix *I*. Example: *IDataProvider*.

YARB class names

All concrete class names that are used have the prefix *Y*. For some class families there are further prefixes:

File Sources: Prefix: *YFS* (e. g. YFSSNNS)

Data Filters: Prefix: *YDF* (e. g. YDFArtih)

Data Viewers: Prefix: *YDV* (e. g. YDVBarChart)

Property methods

In addition to the classes provided by the JDK all property methods have set-, get-, is-, or update prefix followed by the name of the controlled property. (e. g. getName or setParent).

Variables

Our variable naming convention is nearly the same as the famous *hungarian notation* in the Windows programming world. There, all variables have a lower prefix with their corresponding type, which can be one of the following:

integer: Prefix: *n* (e. g. nIndex)

float: Prefix: *f* (e. g. fAverage)

String: Prefix: *str* (e. g. strFilePath)

boolean: Prefix: *b* (e. g. bResult)

If the type is none of these, you should give the variable a really speaking name. For member variable it is mandatory to put the prefix m_ to it (e. g. m_nArrayCounter).
NOTE: This naming conventions are not mandatory for loop variables.

3.4 Class diagrams

As defined in the design process, the main classes had to be extracted from the use cases. After doing this we recognized that there will be a big class hierarchy. So therefore it was a big necessity to design a clear interface model.
A big advantage of our interface model is the flexibility. This flexibility makes it is possible to extend YARB on runtime.

3.4.1 Interface diagram

The resulting interface diagram is shown in figure 3.5.

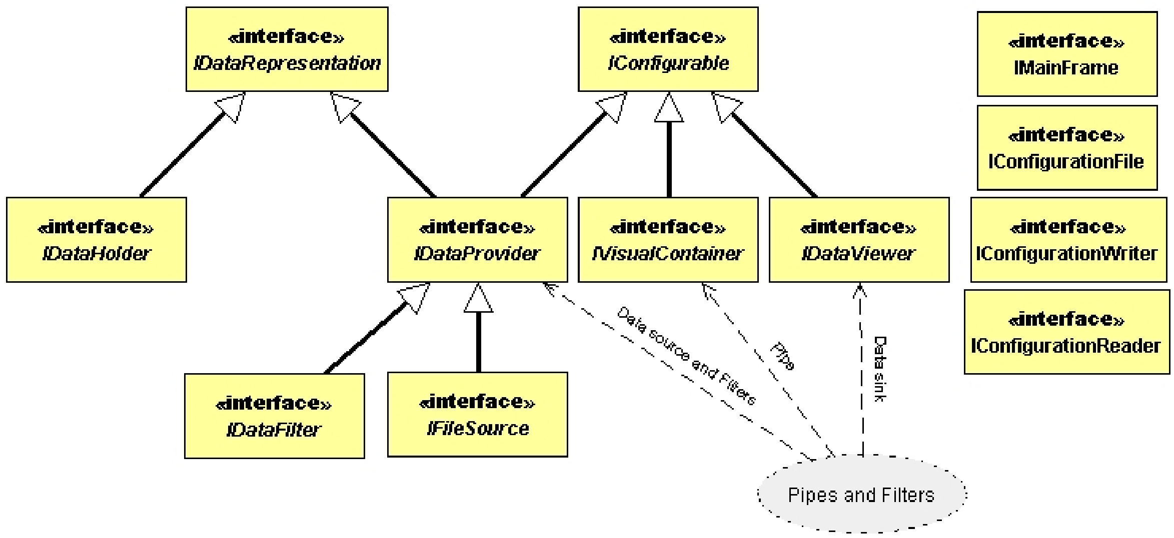

Figure 3.5: CD: YARB Interfaces

IDataRepresentation

IDataRepresentation is the main interface for classes that provide pattern data to other classes.

IDataHolder

The IDataHolder interface was designed for objects that mainly act as data containers. This is mainly true for the data holder of an visualization object.

IConfigurable

Every object that allows the user to configure it interactive has to implement this interface. IConfigurable also provides the functional interface for saving and loading configuration data.

IDataProvider

The classes that are built to serve as data sources for the visualization data pipeline have to implement IDataProvider. This also is the interface for the data source and the filters of the visualization data pipeline.

IDataFilter

This is a specialization of the IDataProvider interface that also allows it to use the data of a predecessor.

IFileSource

The IFileSource interface enhances IDataProvider with methods for specifying the file to read the data from.

IVisualContainer

The IVisualContainer acts as the pipe for the visualization data pipeline. It also is the main access point for the other objects of the pipeline.

IDataViewer

A data viewer acts as a data sink of the visualization data pipeline and does, as the name already says, the visualization.

IMainFrame

The IMainFrame is the interface for the main application frame and responsible for abstraction of additional frame operations.

IConfigurationFile

This interface provides the necessary functionality for working with configuration files.

IConfigurationWriter

The classes that are responsible for saving configuration data to a configuration file have to implement the IConfigurationWriter interface.

IConfigurationReader

Analog to the IConfiguratonWriter interface the IConfigurationReader is for classes that read configuration data.

3.4.2 Main class diagram

The main class diagram of YARB is shown in figure 3.6.
NOTE: In the main class diagram all the YConcreteXXX-classes are only place holders for the real used working classes that are provided by YARB. All these classes are documented in chapter 4.

YConcreteFileSource

Concrete Implementation of IFileSource.

YConcreteFilter

Concrete Implementation of IDataFilter.

YDefaultVisualContainer

Concrete Implementation of IVisualContainer. It also provides the functionality to work with the associated viewer, file source and filters.

YConcreteViewer

Concrete Implementation of IDataViewer. It has to create a new data holder for the data of his predecessors, than create a new viewer frame which uses the data holder for visualization.

YConcreteViewerFrame

It's task is to visualize the data from it's corresponding data holder.

YDefaultDataHolder

Concrete Implementation of IDataHolder.

YYARBMain

YYARBMain is the application loader class.

YMainFrame

The main application window is provided by this class, it also implements the interface IMainFrame. YMainFrame provides the user interface for command input.

YVNavigator

User interface visualization is provided by this class.

YGlobalNavigationDirector

It acts as a mediator (see A.2) , therfore it encapsulates the interaction between YVNavigator and YMainFrame. The idiom used for implementation is Singleton (see A.3).

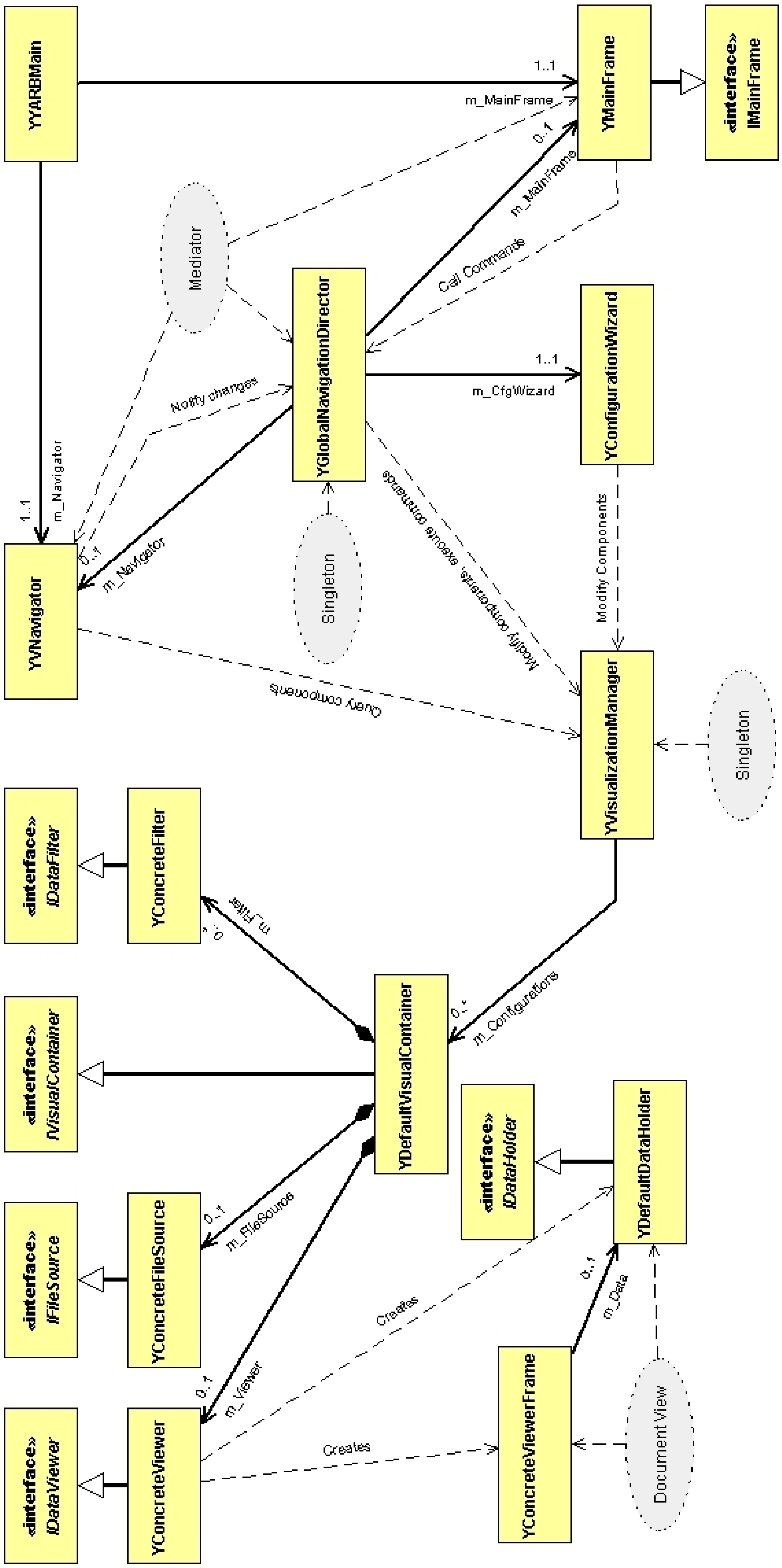

Figure 3.6: CD: Main class diagramm

YVisualizationManager

YVisualizationManager is the internal command center for working with configurations. The idiom used for implementation is Singleton (see A.3).

YConfigurationWizard

Yet not implemented. Class that's responsible for guiding the user through the configuration process.

3.5 Sequence diagrams

Sequence diagram 3.7 explains the pipelining process and the succeeding process of visualization.
The visual container holds all configurations and launches the pipeline process. In this first step the DoVisualization()-method of the data viewer is called. This call forces the data holder to start the data processing, by calling ProcessData() of the given data provider. Here the pipelining starts. The first thing every pipeline component does, is to route the call of ProcessData() through its predecessor, until the file source level is reached. First, the file source is forced to read the data from the file by calling its ProcessData()-method. The call returns, and then every filter continues to read the data from its predecessor into its own data buffer (using the methods provided by the interface IDataRepresentation), calculates the data, and calls FreeData() of the data provider. This is the main process for routing data through the pipe. When the data passed the pipe, we are back in the data viewer. The data viewer creates a new data holder object (in most cases this will be the YDefaultDataHolder), gives a reference of its corresponding data provider (whether a file source or a data filter) by the setDataProvider()-method, which will copy all the data to the data holder buffer. Then the viewer will release the data by calling FreeData() of the data provider.

The data holder now contains all the data, and we are ready to create a new viewer frame and force it to visualize the data in the data holder.

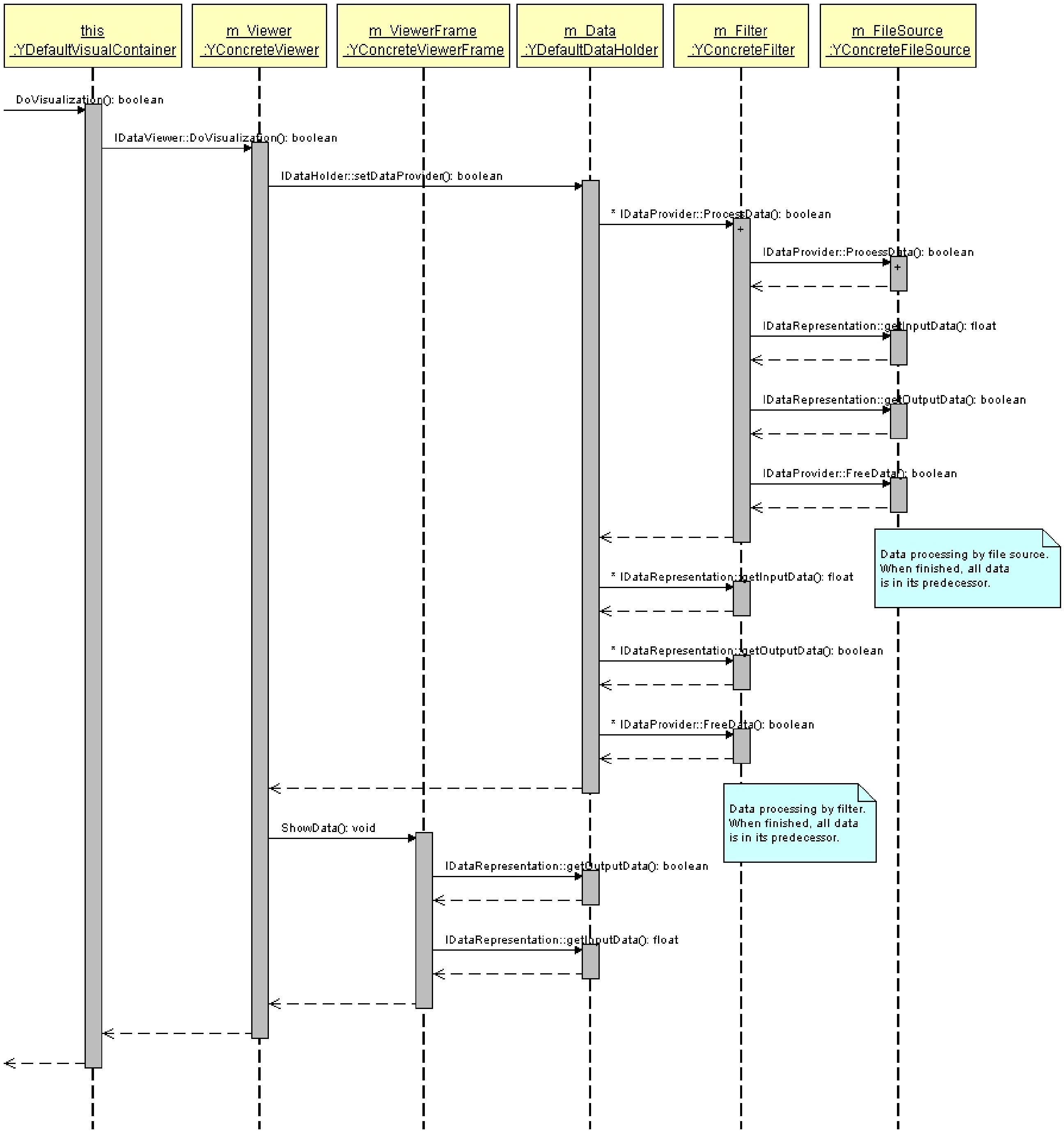

Figure 3.7: SD: Pipeline processing data flow

Chapter 4

Implementation

This chapter describes the implementation of the interfaces in section 3.4.1.

4.1 File Sources

4.1.1 SNNS pattern file source: *YARB.FileSources.YFSSNNSPatternFiles*

This class is capable of parsing existing SNNS pattern files.
It validates the grammar of the whole file and if it is within the convention of the SNNS User Manual V 4.1 ([11],[17]), the input file is parsed. After this it provides the data from within the result or pattern file.
To perform this task every byte of the input file has to be checked, that is why it takes a while to parse bigger files. Every Terminal Symbol and Grammar definition had to be programmed properly to guarantee that the pattern or result file complies with the definition of the SNNS kernel file interface.

4.1.2 SNNS result file source: *YARB.FileSources.YFSSNNSResultFiles*

This class is capable of parsing existing SNNS result files.
The problem was that I didn't found any existing Grammar or Terminal Symbols definition for this file type. So I took the definition for pattern files and changed the definition of the header, the body remains as it is.

- Added Terminal Symbols

```
TEACHING_OUTPUT_INCLUDED : "teaching output included"
END_PATTERN     : "endpattern {WHITE}* ":"
START_PATTERN   : "startpattern {WHITE}* ":"
```

- Added Header Definition

```
start_pattern ::= [Comment] START_PATTERN NUMBER
end_pattern ::= [Comment] END_PATTERN NUMBER
```

- New Header Definition

```
header ::=  VERSION_HEADER V_NUMBER GENERATED_AT
            NO_OF_PATTERN NUMBER
            i_head [o_head] [vi_head] [vo_head]
            [start_pattern] [end_pattern] [TEACHING_OUTPUT_INCLUDED]
```

The value of version number is ignored by the parser, because it's definition couldn't be found.

4.2 Data Filters

Filters do the data transformation task in the YARB data processing pipeline.

4.2.1 Arithmetic Filter: *YARB.Filter.YDFArithmetic*

The Arithmetic Filter provides the following basic arithmetic operations:

- Add
- Subtract
- Multiply
- Divide
- Power

As the second operand the user is able to choose one of the following items:

- Constant Value (user defined)
- Average (calculated for each pattern separately)
- Maximum Value (calculated for each pattern separately)
- Minimum Value (calculated for each pattern separately)
- Sum (calculated for each pattern separately)

4.2.2 Average Filter: *YARB.Filter.YDFAverage*

The Average Filter calculates the average of each pattern. As result the filter always gives one Unit to the next pipeline component.

4.2.3 "The Winner Takes it All" Filter: *YARB.Filter.YDFWTA*

The "The Winner Takes it All" filter identifies the greatest value and gives the unit number of this unit as output. Therefore, if there are any units as filter input, the filter output is always one unit.
Example:
Before the WTA filter the data looks like:

```
INPUT PATTERN 0:
1.0  0.5  0.4  0.3  0.2  0.1

OUTPUT PATTERN 0:
1.0  2.0  3.0  4.0  5.0  4.0  3.0  2.0  1.0
```

After processing by the WTA filter, it looks like:

```
INPUT PATTERN 0:
0.0

OUTPUT PATTERN 0:
4.0
```

4.2.4 "Concatenate pattern" Filter: *YARB.Filter.YDFPatternConcatenation*

This filter concatenates all Patterns in one huge pattern, whereby input/output can be selected.

4.2.5 Vector Filter: *YARB.Filter.YDFVector*

The vector filter is able to transform x,y coordinates into r, φ and vice versa. The user has to choose which direction is performed and select between input or output. This filter was specially implemented for the "Agent" problem.

4.3 Data Viewers

The Viewers are used to visualize the data provided by the last data provider. The concrete implementation of the IDataViewer puts this data in and IDataHolder so that the Frame is able to access the data every time it needs it for updating the screen.

4.3.1 Plain Text Viewer: *YARB.Viewer.StdViewer.YDVPlainText*

The Plain Text viewer prints out the data of the pipeline directly to the screen.

4.3.2 Line Chart Viewer: *YARB.Viewer.StdViewer.YDVLineChart*

The Line Chart viewer uses a xy-axes system. On the x-axis are the input/output units and on the y-axis is the percentage difference between the input/output units drawn. These originated coordinates are connected by a line and printed to the screen.

4.3.3 Bar Chart Viewer: *YARB.Viewer.StdViewer.YDVBarChart*

The Bar Chart viewer uses also a xy-axes system. On the x-axis are the input/output units and on the y-axis is the percentage difference between the input/output units drawn. These originated coordinates are used for the height of the drawn rectangles.

4.3.4 Pie Chart Viewer: *YARB.Viewer.StdViewer.YDVPieChart*

This viewer displays the data in percentage form with slices in a circle.

4.3.5 Agent Viewer: *YARB.Viewer.Special.YDVAgent*

The agent viewer uses a xy-axes system. It needs to get x,y values and information about the direction (left, right, up, down). In the pattern file the input values are stored as absolute values and angels and the output values describe the direction ("the greatest of four numbers wins"). With the help of the "Vector filter" this values are converted into xy - coordinates. A "WTA filter" determines the direction which is drawn as little icon on the screen.
In the configuration dialog of this viewer it is possible to change:

- the labelling of the axes
- the name of the viewer window
- the title of the view:
  the standard name of a view is the file name and the path name of the used file. if the title is changed and you wand to get back file and path name it is possible to write the strings: [FILENAME] or/and [PATH] into the dialog. In the new view they will be changed to a complete filename and path.
- the density of the values on an axis:
  you either can choose to show no values or a special ratio like 1:10. This means 10 values are drawn.

4.3.6 Image Viewer: *YARB.Viewer.Special.YDVImage*

The image viewer is able to show jpeg and gif images on the screen. This viewer is very useful for comparison original images with classified images.

4.3.7 Image Viewer ext: *YARB.Viewer.Special.YDVImageex*

This image viewer support the functionality as the one above but is better. It only works with JDK 1.2.2 and above.

Chapter 5

Configuration Files

Saving different configurations in YARB is very important for the user, because else you would have to configure your specific problem again and again. So we implemented the ability to save the configurations which can be seen in the Navigator of YARB.
The format of the files is in XML, which can be easily understood by anyone who is familiar with HTML.
It is possible to the edit the files and change the parameters of the configuration, but if you want to do so you have to keep up the right format for XML and for the specified Data Table Definition.

5.0.8 Data Table Definition

Here you can see the content of the DTD file

```
<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT YARBConfiguration (Title,File,FileSources, DataFilter*, DataViewer)>
<!ELEMENT DataFilter (Title, Class, boolean*, int*, float*, String*)>
<!ELEMENT FileSources (Title, Class,boolean*, int*, float*, String* )>
<!ELEMENT DataViewer (Title, Class, boolean*, int*, float*, String*)>
<!ELEMENT boolean (#PCDATA) >
<!ELEMENT int (#PCDATA) >
<!ELEMENT float (#PCDATA) >
<!ELEMENT String (#PCDATA) >
<!ELEMENT File (#PCDATA) >
<!ELEMENT Class (#PCDATA) >
<!ELEMENT Title (#PCDATA) >
```

- YARBConfiguration¿ s the root tag of the XML file.
- DataFilter¿is the tag for data filters.
- FileSources is the tag for file sources.
- DataViewer is the tag for data filters.
- boolean, int, float, String are tags for the status variables.

- Title specifies the title.
- File¿specifies the filename to load.
- Class specifies the class.

5.0.9 Example Configuration File

Here we describe how to create or change your own configuration file.

- Specify the header for an XML file (don't change):

```
<?xml version='1.0' encoding='UTF-8'?>
```

- Specify the DTD file, which has to be in the same: directory.

```
<!DOCTYPE YARBConfiguration SYSTEM "YARBConfiguration.dtd" [] >
```

- The root tag for an XML:

```
<YARBConfiguration>
</YARBConfiguration>
```

  All the other tags in the XML have to be inside of this one.

- Main attributes for the configuration:

```
<Title>Config 1</Title>
<int>1</int>
<File>D:\AgentConfigs\AgentExample\gen0agent0.pat</File>
```

- Specify file source:

```
<FileSources>
    <Class>YARB.FileSources.YFSSNNSPatternFiles</Class>
    <Title>File 1 (SNNS PAT)</Title>
</FileSources>
```

- Specify data filters:

```
<DataFilter>
    <Class>YARB.Filter.YDFWTA</Class>
    <Title>Filter 3(WTA)</Title>
    <boolean>false</boolean>
    <boolean>true</boolean>
</DataFilter>
```

- Specify data viewer:

```
<DataViewer>
    <Class>YARB.Viewer.SpecialViewer.YDVAgent</Class>
    <Title>Viewer 2 (Agent)</Title>
    <String>x Axis</String>
    <String>y Axis</String>
</DataViewer>
```

- The complete file:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE YARBConfiguration SYSTEM "YARBConfiguration.dtd" [] >
<YARBConfiguration>
    <Title>Config 1</Title>
    <int>1</int>
    <File>D:\AgentConfigs\AgentExample\gen0agent0.pat</File>

    <FileSources>
        <Class>YARB.FileSources.YFSSNNSPatternFiles</Class>
        <Title>File 1 (SNNS PAT)</Title>
    </FileSources>
    <DataFilter>
        <Class>YARB.Filter.YDFWTA</Class>
        <Title>Filter 3 (WTA)</Title>
        <boolean>false</boolean>
        <boolean>true</boolean>
    </DataFilter>
    <DataViewer>
        <Class>YARB.Viewer.SpecialViewer.YDVAgent</Class>
        <Title>Viewer 2 (Agent)</Title>
        <String>x Axis</String>
        <String>y Axis</String>
    </DataViewer>
</YARBConfiguration>
```

Chapter 6

Expanding YARB

This chapter describes how to expand YARB to match the requirements of your own problem domain. The expansion may consist of writing your own file sources, data filters or data viewers. Because there are already file sources available for the common used SNNS file formats (pattern and result files) and for picture elements, the main job will be to write your own data filter and to adapt one of the existing data viewers.

When expanding YARB we recommend to study the code of one of the existing components well.

6.1 Dynamically Class Loading

We added the ability to load the classes for File Sources, Data Filters and Data Viewers dynamically over an existing XML file.
This file is located at /YARB/Configuration/DynamicClasses.XML, and can be edited.
It is used to tell YARB which classes are available next to the standard components.
The convention of this file can easily be read out of it.
Some parts of this file follow:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE YARBConfigurationDTD SYSTEM "YARBConfiguration.dtd" [] >
<YARBConfiguration>

  <Title>YARB dynamic class loader configuration file.</Title>

  <FileSources>
    <Class>_none_</Class>

    <Title>The following integer contains the number of loadable file sources:
    </Title>
    <int>1</int>

    <String>This integer contains the number of the following strings. Should
            always be 3.(Full class qualifier, Short description, Description
    </String>
```

```
      <int>3</int>
      <String>YARB.FileSources.YFSSNNSPatternFiles</String>
      <String>SNNS PAT</String>
      <String>File source for SNNS pattern files.</String>

  </FileSources>

  <DataFilter>
    <Class>_none_</Class>

    <Title>The following integer contains the number of loadable data filter:
    </Title>
    <int>1</int>

    <String>This integer contains the number of the following strings. Should
            always be 3.(Full class qualifier, Short description, Description
    </String>
      <int>3</int>
      <String>YARB.Filter.YDFOne2One</String>
      <String>1:1</String>
      <String>1:1 data filter. Does nothing.</String>

  </DataFilter>

  <DataViewer>
    <Class>_none_</Class>

    <Title>The following integer contains the number of loadable data viewer:
    </Title>
    <int>1</int>

    <String>This integer contains the number of the following strings. Should
            always be 3.(Full class qualifier, Short description, Description
    </String>
      <int>3</int>
      <String>YARB.Viewer.StdViewer.YDVPlainText</String>
      <String>Plain text</String>
      <String>Plain text viewer.</String>

  </DataViewer>

</YARBConfiguration>
```

6.2 File Sources

On the following pages you can see what has to be done to create your own File Source for YARB.

6.2.1 Classes and Interfaces to import and implement

- Package : YARB.FilesSources;
- Needed imports :
  - YARB.Interfaces.*;
  - YARB.Dialogs.*;
- Interfaces to implement :
  - IFileSource (main Interface)
  - IDataProvider (implied by IFileSource)
  - IDataRepresentation (implied by IDataProvider)
  - IConfigurable (implied by IDataProvider)

6.2.2 Methods to implement

Most of the code inside the methods was created as an example. What you have to do is to adapt the data of your File Source into the interfaces .

- This function forces the object to save its configuration data to a file

```
public boolean SaveConfiguration(IConfigurationWriter Cfg) throws
IConfigurationWriter.ECfgWriterError {
    if(Cfg==null) return false;

    Cfg.addTitle(getName());
    Cfg.addBoolean(bool1);
    Cfg.addBoolean(bool2);
    Cfg.addString(string);

    return true;
}
```

  In this function you have to save the status variables of your class. It's important to keep up the right order, because you've to restore them in the order as you saved them. Take a look at the interface IConfigurationWriter to see what methods are supported.

- This function forces the object to read a configuration from a file

```
public boolean LoadConfiguration(IConfigurationReader Cfg) throws
IConfigurationReader.ECfgReaderError {
    if(Cfg==null) return false;

    setName(Cfg.getTitle());
    bool1 = Cfg.getBoolean();
    bool2 = Cfg.getBoolean();
    string = Cfg.getString();
    return true;
 }
```

In this function you have to restore your status variables. As mentioned before you've too keep up the right order, or you'll get false values or even errors.

- This function determines if your file source is configurable or not, default should be set to true.

```
public boolean IsConfigurable() {
    return true;
}
```

- With this function you can implement a method to use a dialog to configure your File Source.

```
public boolean ShowConfigurationDialog(JFrame ParentFrame) {
    YStandardNameChooser dlg = new YStandardNameChooser(
      ParentFrame, "YARB Viewer Properties",true,getName());
    dlg.show();

    setName(dlg.getName());
    return true;
}
```

We designed our pipeline process to work with SNNS files, that's why we took the naming conventions from the pattern files. If you need further information of the naming conventions of the used methods go and take look at the SNNS users manual ([11]).

- This function has to return the number of patterns.

```
public int numbOfPatterns() {
    return 0;
}
```

- This function has to return the number of input units.

```
public int numbOfInputUnits() {
    return 0;
}
```

- This function has to return the number of Output units.

```
public int numbOfOutputUnits() {
    return 0;
}
```

- This function has to return the number of variable input dimensions.

```
public int numbOfVariableInputDimensions() {
    return 0;
}
```

- This function has to return the number of variable ouput dimensions.

```
public int numbOfVariableOuputDimensions() {
    return 0;
}
```

- This function has to return the value of one max input dimension.

```
public int getMaxInputDimesion(int nVarInputDimesion) {
    return 0;
}
```

- This function has to return the value of one max output dimension.

```
public int getMaxOutputDimesion(int nVarOutputDimesion) {
    return 0;
}
```

- This function has to return the value of one Outputdata. Where nPattern is the pattern and nUnit the needed unit.

```
public float getOutputData(int nPattern,int nUnit) {
    return (float)0;
}
```

- This function returns the value of one Inputdata. Where nPattern is the pattern and nUnit the needed unit.

```
public float getInputData(int nPattern,int nUnit) {
    return (float)0;
}
```

- This function has to trigger the file source to process its data.

```
public boolean ProcessData() throws
IDataProvider.DataProcessingError {
    return true;
}
```

- This function has to free all the resources of the file source.

```
public boolean FreeData() {
    return true;
}
```

- This function has to set the filename of the source.

```
public void setFileName(String strName) {
     m_FileName=strName;
}
```

6.3 Data Filters

On the following pages you can see what has to be done to create your own Data Filter for YARB.

6.3.1 Classes and Interfaces to import and implement

- Package : YARB.Filter;
- Needed imports :
  - YARB.Interfaces.*;
  - YARB.Filter.*;
- Interfaces to implement :

  - IDataFilter (main Interface)
  - IDataProvider (implied by IDataFilter, description see 6.2.2)
  - IDataRepresentation (implied by IDataProvider, description see 6.2.2)
  - IConfigurable (implied by IDataProvider, description see 6.2.2)

6.3.2 Methods to implement

Most of the code inside the methods was created as an example. What you have to do is to adapt the data of your Data Filter into the interfaces. A Filter has to process the data of its predecessor, safe the calculated data in a buffer and consigns the buffer to the successor.

- This method let me know which data provider to use to get my data for processing:

```
public void setPreviousProvider(IDataProvider DP) {
    m_DP=DP;
}
```

- After the data provider was set, the ProcessData() method will be called.

```
 public boolean ProcessData() throws IDataProvider.DataProcessingError
{
    if(m_DP==null) return false;
    m_DP.ProcessData();
```

  Now copy the data from your data provider with

```
    m_DP.getXXXData()
```

  to a buffer and do the data calculation that your filter should do:

```
    .   // Filter calculations
    .
    .
```

  Note: Don't forget to let your data provider free its internal buffers with:

```
    m_DP.FreeData();
}
```

- Now the data will be read by the next pipe component with the methods from the IDataRepresentation interface. For documentation see (see 6.2.2).
- The other methods which have to be implemented can be found in section File Sources (see 6.2.2).

6.4 Data Viewers

The following section describes how to create your own Data Viewer for YARB.
It is completely on your own how to design your viewer frame. The viewer frame is one of the less components that don't relay on the other YARB architecture. The standard viewers shipped with YARB are just some common examples for all the powerful jobs that can be done by using YARB.

6.4.1 Classes and Interfaces to import and implement

- Package : YARB.Viewer;
- Needed imports :
  - YARB.Interfaces.*;
  - YARB.Viewer.*;
- Interfaces to implement :
  - IDataViewer (main Interface)
  - IConfigurable (implied by IDataFilter, description see 6.2.2)

6.4.2 Methods to implement

Most of the code inside the methods was created as an example. What you have to do is to read the data from the IDataHolder and visualize it as you need it.

- The function **setCaption** sets the caption of a new Frame that will be displayed in the title bar:

```
void setCaption(String strTitle);
```

- Sets the data source, that provides the visualization data:

```
void setDataSource(IDataProvider DP);
```

- Sets the file name for the data file:

```
void setFileName( String Name);
```

- Start the visualization:

```
boolean DoVisualize(IMainFrame ParentFrame)
        throws DataProvider.DataProcessingError;
{
```

Check if there is any data do visualize:

```
    if(m_DP==null){
        return false;
    };
```

Create a new DataHolder:

```
    YDefaultDataHolder data=new YDefaultDataHolder();
```

Process the data from the DataProvider, store it in the DataHolder and free the the DataProvider:

```
    m_DP.ProcessData();
    data.setData(m_DP);
    m_DP.FreeData();
```

Set the title of the frame with the caption, the file name and the specified viewer name and create a new frame:

```
    String str = m_Caption + " (" + m_File + ")"
                                + " [" + m_strName + "]";
    YDVLineChartFrame txt = new YDVLineChartFrame(str,m_strX,m_strY);
    txt.setDataHolder(data);
```

Then set the location of the new frame to the one that the parent frame gives back, add the frame to the parent frame and exit successfully:

```
    txt.setLocation(ParentFrame.GetNextCreationPoint());
    ParentFrame.AddFrame(txt);

    return true;
}
```

6.4.3 Already implemented "real world" example to expand YARB

- *"The Agent Problem"*

An object - called the agent - moves around, and every object on a coordinate system points to the direction the agent would move to if it would be there.

The data is stored in a pattern file whereby the input values an absolute value (r) and angel (ϕ) describe. The output values are the direction (left, right, up, down). The output values are four numbers and the greatest number of them determines the direction. Therefore it was our job to create a filter that converts r,ϕ values into x,y coordinates and a filter that chooses the greatest number of an amount (the winner takes it all - principle).

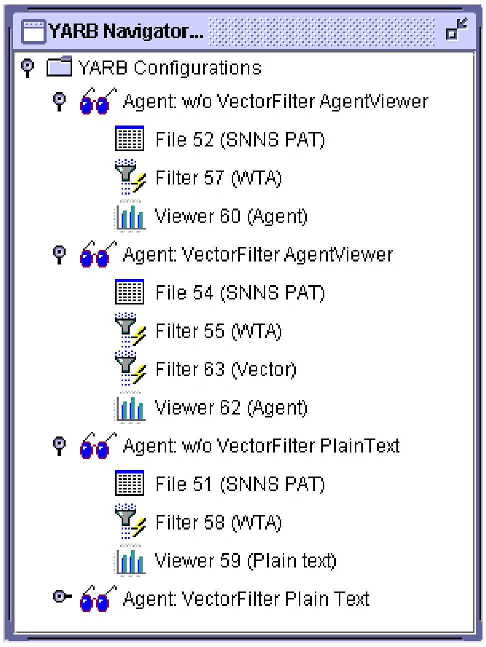

Figure 6.1: Configuration dialog for "Agent Problem"

Figure 6.1 shows the configuration dialog specially for the agent problem. The figure 6.2 is the resolution of the first configuration in this dialog. It is evident that only r,ϕ values are drawn. Figure 6.3 shows the result of the second configuration. Thus the vector filter is used the values are drawn on x,y coordinates. The direction of the icons in both images is reached through the use of the "WTA" filter.

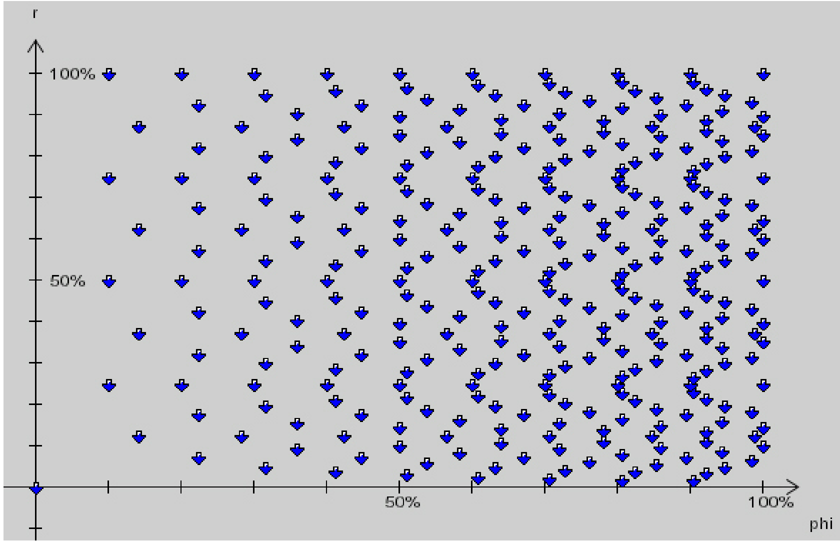

Figure 6.2: Agent Viewer: r/phi

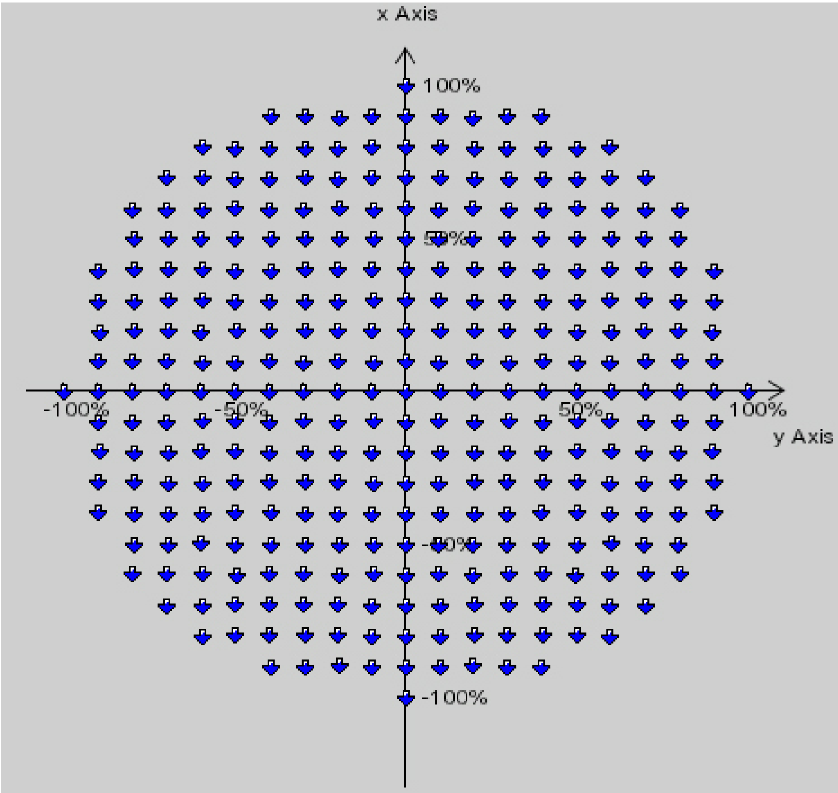

Figure 6.3: Agent Viewer: x/y

Appendix A

Used design pattern

When designing the YARB architecture we used many design pattern or one of their mutants. In this chapter a short overview of these pattern will be given. For further information about these design patter we recommend reading of [14] (or [15]) and [16]. For the description of pattern we use the same terminology as the books above used.

A.1 Pipes and Filters (Architecture pattern)

The pipes and filters pattern provides a structure for systems that processes data streams. Each processing step is encapsulated in an extra filter. The data ist routed through pipes from filter to filter. It is easy to build families of different systems by adding new filters and arranging filters in different order.

Context

This pattern works in the context of processing data streams.

Problem

A system for processing data streams has to be built and the implementation as a single component in not able for reasons like : Different developing teams, the natural of the system is divided too, the requirements are subject to change, ...

Solution

The pipes and filters pattern splits the job into different, sequently tasks.

Structure

The base structure of the pipes and filters pattern is shown in figure A.1.
Filters are the working components of the pipeline. A Filters completes, transforms of refines it's input data.
Pipes are the links between the filters. It is possible to use active or passive filters.
A *Data source* is the provider of the input data for the system.
Data sinks are the data customer, that collect the data at the end of the pipeline.

Figure A.1: Base structure of the pipes and filters pattern

Consequences

The pipes and filters pattern has the following advantages:

- Temporary files are not necessary, but possible
- Flexibility by exchanging filters
- Flexibility by changing the filter order
- Reuse of filters
- Rapid prototyping of pipeline prototypes
- Efficiency when using parallel pipe processing

Using the pipes and filters pattern also brings disadvantages:

- Shared use of state information is expensive and not flexible
- More efficiency by parallel processing often is an illusion
- Unnecessarily data conversion
- Error handling

Literature

A detailed description of the pipes and filters pattern can be found in [16, pages 54-71].

A.2 Mediator (Design pattern)

The Mediator design pattern defines a object that encapsulates the interaction between several other objects. Mediators are good for loose coupling by keeping its collaborators away form calling them explicitly. Therefore the interaction between is colleagues can be changed independently from them.

Context

Mediators are used when several object interact in complex ways and the interaction should be displaced to an extra object.

Problem

In object oriented design the distribution of interactions between objects is supported. But the result of such a design is an object structure with many relations between the objects. In the worst case, every object knows each other.

Solution

The Mediator pattern encapsulates the interaction between objects and as the result the number of object relation decreases. The Objects only know the mediator, and the mediator is responsible for doing the interaction with the other objects.

Structure

The structure of the Mediator pattern is best documented in its class diagram, like in figure A.2.

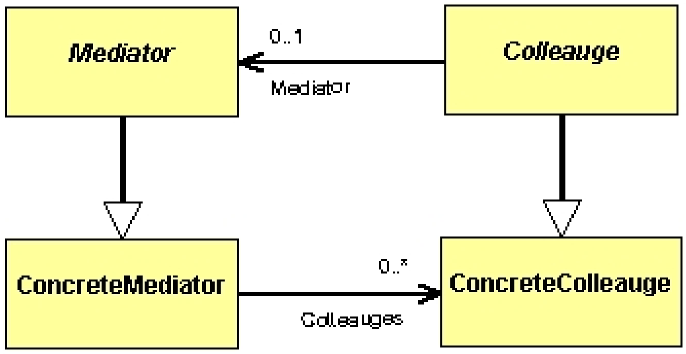

Figure A.2: Class diagram of the Mediator pattern

The *Mediator* defines the interface of the interaction with its colleagues.
A *ConcreteMediator* implements the behavior of the system by coordinating its colleague objects. It also knows and manages its colleagues.
ConcreteColleagues knows their mediator object and works together with it, but never does anything with an other colleague.

Consequences

The mediator pattern has the following advantages:

- Limitation of subclassing
- Decouples colleague objects
- It simplifies the protocol of objects
- It abstracts the object interaction
- It centralizes the control

Literature

For a detailed description of the Mediator design pattern see [15, pages 345-355].

A.3 Singleton (Idiom)

The Singleton pattern ensures that there is exactly one instance of a class per time. It also provides the global access point for this one and only object.

Context

In some cases it is very important hat there is only one instance of a special class. For example there may be several printers in a system, but only one print spooler, or a digital filter uses only one A/D converter.

Problem

How to make sure that there is no other instance of this class. A global variable may be the access point for such an object, but it doesn't prevent from creating several instances. And in some programming languages, e. g. java, there are no global variables.

Solution

Therefore it is better to set the responsibility for creating the one and only instance to the class itself. The class can catch all attempts of creating other instances. This is the task of the Singleton pattern.

Structure

First the class diagram in figure A.3 is discussed.

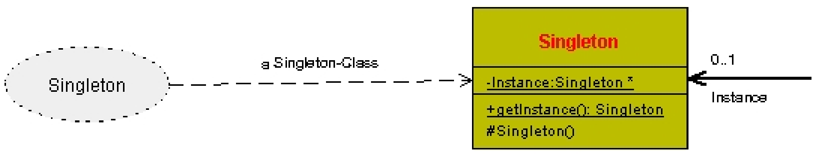

Figure A.3: Class diagram of the Singleton pattern

Singleton

- Defines a class operation that allows clients to access its one and only instance.
- Is responsible for creation of its one and only instance.

Now let's take a look to figure A.4, where the process of getting access the one and only instance is shown:

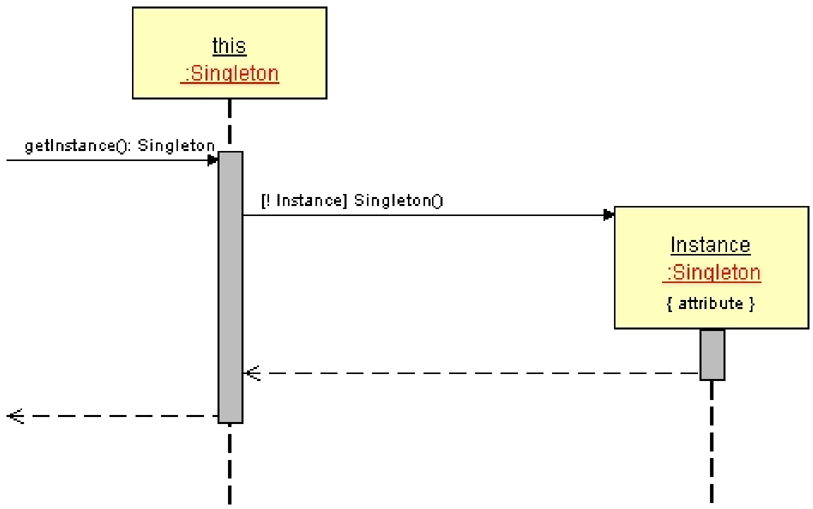

Figure A.4: Sequence diagram of the Singleton pattern

Consequences

The consequences when using the Singleton pattern are the following:

- Access control to the instance
- Restricted name space
- Refining of operations and representation
- Eventually variable number of instances
- It's more flexible than class methods

Literature

For a detailed description of the Singleton pattern see [15, pages 139-146].

Appendix B

Reports

This chapter contains our reports and results of the week.

B.1 1st. Protocol March 15, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.00 - 16.30
- *Chief of protocol:* Erich Eichberger

General project definition

Theme: Visualization of netJEN problem domains and results

The project is going to be a graphical tool, written in java, for the visualization of netJEN problem domains. The aim is to show the pattern file[1] before and after netJEN evaluated it (they are called learning and result pattern). The pattern files are in the pattern file format of the SNNS [2]. Thus the program has to be able to read this files and display the information contained in a manner that depends on the problem domain. This, perhaps, could be an image from a satellite that should be classified or prognosis of tax courses.

Topics talked about

- Administrative details
- Sources for Information related to the topic
- Presentation of our group's web site
- The Neural Network Simulator netJEN and SNNS
- Problem domains, kinds of visualization

[1]The data neural network simulators work with is called pattern
[2]Stuttgart Neural Network Simulator

Task list

- Pictures of group-members (for web site)
- Find problem domains for NNs [3]
- Find NN-applications and problems that are solved with NNs

[3] NN... short form of Neural Network

B.2 Results of the week (March 15th to March 22nd)

The goal of this week was to find the applications and problem domains of NNs. First we discuss the NN-Applications we found, and then we list the problem domains we found within these applications.

Applications for NNs

Manufacturing

- process optimizing
- process monitoring and controlling
- determining operational policies
- predictive control of quality
- flexible manufacturing scheduling

Chemical Industry

- sensor analysis
- spectroscopy
- molecular modeling
- data rectification for dynamic processes
- process modeling for fault detection
- statistical analysis

Medicine

- diagnosis of coronary artery disease
- structure analysis of virus proteins
- diagnosis of epilepsy
- rehabilitation of persons with incomplete spinal cord injury
- analysis and classification of EEG signals

Nuclear engineering

- spectral analysis
- signal prediction in nuclear power plants
- classification of energy levels
- alarm handling in nuclear power plants

Miscellaneous applications

- classification of satellite pictures
- sensor analysis
- optical character recognition
- voice recognition
- stock rate prognosis
- texture recognition
- temperature regulation
- sunspot evaluation
- business prognosis and optimization
- sale prognosis

Problem domains

- classification
- time series prediction
- parameter determination
- signal recognition
- signal analysis and prediction
- rule extraction
- optimization
- modeling

Ideas for possible visualization

- picture data as bitmap
- classification as regions (graphs)
- bar chart
- line chart
- plain pattern text
- graphs
- text

B.3 2nd. Protocol March 22nd, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.00 - 16.30
- *Chief of protocol:* Erich Eichberger

Topics talked about

- Results form last week
- Possible visualization of problem domains
- Possible names of problem domains (like classifikation, ...)

Task list

- Adding bibliography information
- Finding problem domains, their possible visualization and names
- Pictures of possible visualization
- Grouping NN-Applications to problem domains
- Work into ArgoUML

B.4 Results of the week (March 22nd to March 28th)

Adding bibliography information

- Added BibTeX file
- Added all our Internet links

Finding problem domains, their possible visualization and names

- Analyzed the application for ANNs and classified the problem domains
- Added the corresponding problem domains to all applications
- Collected sample pictures of possible visualization, added cross references to all applications
- Got names for problem domains

B.5 3rd. Protocol March 29th, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.00 - 16.30
- *Chief of protocol:* Erich Eichberger

Topics talked about

- Results form last week
- Structure of the project paper
- File concepts of NetJEN
- File formats

Task list

- Restructuring of the project paper
- Research the format of the used files
- Add introduction to project paper
- Take a look at UML
- Think about possible realization

B.6 Results of the week (March 29th to April 12th)

Restructuring the project paper

- Changed the layout to book
- Added introduction chapter

Research the format of the used files

- Took a look at the SNNS book
- Took a look at the samples provided by Mr. Schwaiger

Think about possible realization

- Reconfigured the analyzing phase
- Thought about possible use of design patterns
- Introduced the concept of the configuration files
- Basic approach to use cases

B.7 4th. Protocol April 12th, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.30 - 17.00
- *Chief of protocol:* Erich Eichberger

Topics talked about

- Results form last week
- Use cases
- Class diagrams
- Handling of the different problem domains
- Configuration file with XML format

Task list

- Design use case diagrams
- Design class diagrams
- Develop a prototype GUI
- Search for existing XML parsers
- Create a prototype configuration file

B.8 Results of the weeks (April 12th to May 2nd)

Design use case diagrams

- Interaction between User, NetJEN and YARB
- Run YARB manually
- Configure YARB manually
- Run YARB by RMI

Design class diagrams

- YARB Interfaces
- Main class diagram

Develop a prototype GUI

The Test-GUI for playing with configurations, file sources, data filters and data viewers is almost finished.

XML parsers

We decided to use the new XML format to represent our configuration files. After a while I decided to use Sun's XML parser, also called JAXP. I had to figure out how XML is defined, afterwards I wrote some exmaple files. I used the Document Object Model to parse my files, because you get a tree like result,and displayed it in a Java Swing Tree.

Task management

After finishing the design phase we had to split the project implementation into different tasks and to fix the task distribution.

- **Aschauer Thomas:**

  Software architecture These are the Use Cases, Class Diagrams, persistence model

  Interface Design Includes all interface classes

  Main application logic Implementation of the main frame, the YARB Navigator, ...

- **Eichberger Erich:**

  Data Viewer Implementation and design of the different viewer classes. See 1.5.2 on page 14 for details.

- **Engl Thorsten:**

  XML - parse and persistence engine Implementation of an existing XML parser, like Suns JAXP, to read and store different configurations.

  File Sources Parser for existing SNNS files.

- **Zödl Christian:**

  Data Filter Identify all needed filters, design and implement them. A filter that executes the external LASSPTools filters is planned too.

B.9 5th. Protocol May 2nd, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.30 - 17.00
- *Chief of protocol:* Erich Eichberger

Topics talked about

- Results form last week
- Use cases
- Class diagrams
- Documentation
- YARB Architecture design

Task list

- Update documentation
- Continue programming

B.10 Results of the weeks (May 2nd to May 24th)

Update documentation

- Document the Use Case diagrams
- Document the Class diagrams
- Added appendix A (Short overview of used design pattern)
- Added documentation of file sources, filters and viewers

Programming tasks completed

- Updated user interface
- Added file source for Pattern files
- Added average-, "the winner takes it all"- and general purpose arithmetic- filter
- Added viewers for plain text, bar chart and line chart

B.11 6th. Protocol May 24th, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.30 - 17.00
- *Chief of protocol:* Erich Eichberger

Topics talked about

- Results form last week
- Documentation
- The "Agent" Problem

Task list

- Update documentation: How to implement for example a new viewer, filter, ...
- Implement the "Agent" problem
- Continue programming

B.12 Results of the weeks (May 24th to June 07th)

Update documentation

- Added documentation of already implemented filters and viewers.
- Added sequence diagram for better understanding the pipeline processing data flow.
- How to expand YARB with your own file sources, filters and viewers

Programming tasks completed

- Added capability of saving and loading configuration in XML format.
- Updated interface.
- Added file source for Result files.
- Added patter concatenate filter and vector filter for the "Agent" problem.
- Added viewers for pie chart and for the "Agent" problem.

B.13 7th. Protocol June 07th, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.30 - 17.00
- *Chief of protocol:* Erich Eichberger

Topics talked about

- Results form last week
- Documentation
- The "Agent" Problem - "Problem"

Task list

- Update documentation: How to implement for example a new viewer, filter, ...
- Finish the "Agent" problem
- Find remaining "features" `:-)`.

B.14 Results of the weeks (June 07th to June 21st)

Update documentation

- Documented how to expand YARB with your own file sources, filters and viewers
- Added instructive case: "Extending YARB for the Agent example"

Programming tasks completed

- Added capability of saving and loading configuration in XML format: now special characters are handled correctly
- Finished Agent viewer
- Finished Vector filter
- Added dynamically class loading with a XML file

B.15 8th. Protocol June 21st, 2000

General

- *Present persons:*
  - DI Mag. Roland Schwaiger
  - Erich Eichberger
  - Thomas Aschauer
  - Thorsten Engl
  - Christian Zödl
- *Place:* DI Mag. Schwaigers Office
- *Time:* 16.30 - 17.00
- *Chief of protocol:* Erich Eichberger

Topics talked about

- Results form last week
- Documentation
- The "Agent" Problem - "Problem"

Task list

- Update documentation about the Configfile
- Make several changes on the Agent Viewer

B.16 Results of the weeks (June 21st to July 05th)

Update documentation

- Added a description of the structure of a config file
- Added a short abstract of the project

Programming tasks completed

- Added and completed several problems at the agent viewer
  - Added a possibility to change the title of a view.
  - Changed colors of the arrow bitmaps.
  - All views are now stretched to a full window view
  - Changed axis labelling

Bibliography

[1] The centre for neural computing applications. http://www.brunel.ac.uk/research/cnca/links/related.html.

[2] Einfuehrung in den stuttgarter neuronale netze simulator, 1996. http://www-i6.Informatik.RWTH-Aachen.de/Colleagues/HeKonn96/abstract/snns.html.

[3] Neural network applications in manufacturing. http://www.emsl.pnl.gov:2080/proj /neuron/neural/bib/manufacturing.html.

[4] Neural network applications in chemistry. http://www.emsl.pnl.gov:2080/proj /neuron/neural/bib/chemistry.html.

[5] Neural network applications in medicine. http://www.emsl.pnl.gov:2080/proj /neuron/neural/bib/medicine.html.

[6] Neural network applications in power plants. http://www.emsl.pnl.gov:2080/proj /neuron/neural/bib/power.html.

[7] Nn-at gruppenprofile (8-apr-1997). http://www.ai.univie.ac.at/nn-at-97/gruppen.html.

[8] Problem domain. http://www.cis.hut.fi/ jhollmen/dippa/node36.html.

[9] Neural network applications in molecular modeling. http://www.emsl.pnl.gov:2080/proj /neuron/neural/bib/molecular.html.

[10] Neural network applications in process control. http://www.emsl.pnl.gov:2080/proj /neuron/neural/bib/molecular.html.

[11] Andreas Zell; Günter Maimer; Michael Vogt; Ralh Hübner; Sven Döring; Kai-Uwe Herrmann; Tobias Soyez; Michael Schmalzl; Tilman Sommer; Artemis Hatzigeorgiou; Dietmar Posselt; Tobias Schreiner; Bernward Kett; Gianfranco Clemente; Jens Wieland; Niels Mache. *SNNS Stuttgart Neural Network Simulator User Manual.* 1.4th edition, 1995. Report No. 6/95.

[12] Bjarne Stroustrup. *Die C++ Programmiersprache.* Addison-Wesley, 3rd edition, 1998. ISBN 3-8273-1296-5.

[13] Rainer Burkhardt. *UML - Unified Modeling Language. Objektorienntierte Modellierung für die Praxis.* Addison-Wesley, 2nd edition, 1999. ISBN 3-8273-1407-0.

[14] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Design Patterns - Elements of reusable object oriented software.* Addison-Wesley, 1st edition, 1995. ISBN 0-201-63361-2.

[15] Erich Gamma; Richard Helm; Ralph Johnson; John Vlissides. *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, 1st edition, 1996. ISBN 3-89319-950-0.

[16] Frank Buschmann; Regine Meunier; Hans Rohnert; Peter Sommerlad; Michael Stal. *Pattern-orientierte Software-Architektur. Ein Pattern-System.* Addison-Wesley, 1st edition, 1998. ISBN 3-8273-1282-5.

[17] Stuttgart neural network simulator, 1997. http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html.