Single Honours Computing Project

2011/2012

Temporal Discovery Workbench (TDWB)

Dissertation and manuals

Daniel Blasco Calzada

d.blascocalzada.11@aberdeen.ac.uk

Department of Computing Science

University of Aberdeen

Aberdeen AB24 3UE, UK

I. Acknowledgements

I would like to thank my project supervisors, Professor Derek Sleeman and Dr Wamberto Vasconcelos for all their advice and assistance during the development of this project, and also for providing such an interesting and rewarding topic for the basis of my Honours project.

I would also like to thank my family and friends for their words of wisdom and advice throughout the project's duration.

II. Declaration

I declare that this document and the accompanying code have been composed by myself, and describe my own work, unless otherwise acknowledged in the text. It has not been accepted in any previous application for a degree. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed: …….. Date: ……

Daniel Blasco Calzada

III. Abstract

The necessity of providing an agile tool to researchers, who want to describe the data series behaviour before a special event, has encouraged me to carry out this project. This is a software engineering project that provides a solution to a real problem.

Before producing a software program, we outlined a possible solution for this problem. In order to implement the designed solution, two prototypes have been developed. These prototypes have been evaluated by domain-knowledgeable analysts. As a result of this feedback, I have changed the specifications for the program's final version.

The final version of Temporal Discovery Workbench (TDWB) offers the needed functionalities to solve the main goal and the secondary goal of this project. Proposed further work and improvements are described at the end of this document.

IV. Index of contents

V. Index of figures

VI. Index of tables

1 Introduction

This sections provides an overview of the problem to be solved, the goals, the objectives, and the document structure

1.1 Overview

To detect myocardial damage (MD) in intensive care unit (ICU) patients, a test is made every 72 hours. This test checks if the troponin level in the patient's blood is high (i.e. over a pre-defined threshold). This test is expensive and is only performed infrequently. So it would be convenient to find a cheaper and faster system to detect MD.

The experts know that after a MD the patient's physiological parameters change. For example, as we can see in Figure 1-1 – Special event example we suggest the hypothesis that a high troponin value is detected after three high HR values. To check if the hypothesis is valid, we could match this pattern against hundreds of similar data sets where half of them contain raised troponin values and the other half report zero or low troponin values. If the pattern matches all- or almost all- of the positive data set and doesn't match any- or almost any- of negative data set then we could say that there is a correlation between the HR parameter and the troponin values.

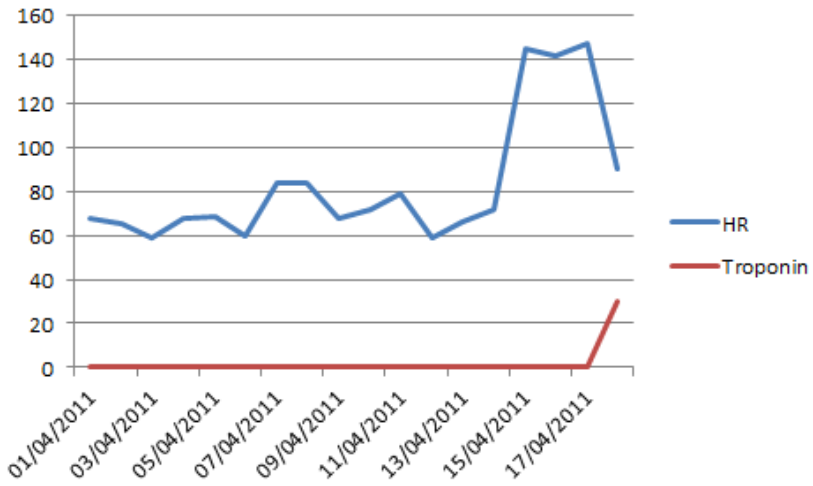

Figure 1-1 – Special event example

Sleeman et al [DS 11] argue that a software system could provide domain experts support when attempting to detect such events. With such support it would be easier for experts to detect a MD before the 72 hours period. Also, this new detection system would be cheaper than the old troponin-based detection test.

The experts have determined some of these patterns. But these patterns happen in multiple parameters that frequently change, and are different depending on the type of MD. Hence these patterns are not always precise. As a result, we need an intuitive and agile tool to help the domain expert to formulate, test, and improve such patterns.

1.2 Objectives

The high-level objective is to provide a tool which enables the domain experts understand the data series patterns before a special event occurs. This will involve providing the domain expert with statistical information about the data series and by matching the produced patterns against these data series.

1.3 Primary goals

The main goal of this software engineering project is to design and build a workbench which the myocardial damage experts can use to develop potential explanations for troponins rises.

1.4 Secondary goals

The secondary goal is to generalise the workbench to be useful in a variety of domains, like weather prediction, ecology and other medical domains. This could be a demanding yet interesting goal.

1.5 Structure of Document

Chapter 1: Overview of the problem addressed, the project's goals and objectives.

Chapter 2: Describes the background and the required specifications of the needed software system.

Chapter 3: Initial design of a possible solution to this problem.

Chapter 4: The first prototype of Temporal Discovery Workbench (TDWB).

Chapter 5: The second prototype of TDWB.

Chapter 6: The final version of TDWB.

Chapter 7: Conclusions and further work.

Chapter 8: References.

Chapter 0: Appendices.

2 Background

The software engineering process begins with the software requirements specification, which is the description of the system functionalities.

2.1 Specifications

The specifications of this program are given by the section 4.1.5 (Temporal Discovery Workbench) from the Sleeman et al [DS 11] paper, which describes: "To date in this series of projects we have accepted an initial rule-set from a domain expert (as we believed that machine learning algorithms sometimes failed to incorporate domain-important concepts /rules). However, as the domain and the task (prediction in temporal datasets) gets more complex we feel it is appropriate to develop systems which are genuinely collaborative i.e. where both the system and the expert suggest features (to explain specific temporal events), the system creates from these composite features, and these

are evaluated against datasets. The expert then decides on the basis of “coverage” statistics and his knowledge of the domain which patterns should be retained and developed further. The ground breaking APRIORI algorithm has recently been developed to handle temporal datasets and patterns; we plan to use this later algorithm as a central component of this collaborative workbench.”

This is a general guideline about the software system to be developed. Interviews with Professor Derek Sleeman will outline a more precise requirements list.

2.2 APRIORI algorithm

Analysing the data is a key step in guiding the domain expert to produce more precise patterns.

The APRIORI algorithm concept, as described in [LAX 06], can be used as a guide for designing the analysis modules of our Workbench. Basically, the APRIORI algorithm principle states that more complex patterns should only be created if their elementary components have sufficient support. The elementary patterns for a domain are the variable name and one of its ranges. For example, the normal values and the very high values of a patient’s heart rate can be two distinct ranges. If for all the data sets, the system provides the number of values that are in each range, maybe the domain expert can determine if there are more high values in data sets of type A than in the data sets of type B. This feedback helps the user decide which elementary patterns could be used to build more complex patterns.

The variables are changing along the time and these changes are also important to understand better the causes of the problem, so the system should provide also a report about these changes.

2.3 Initial requirements

This is the list of the requirements initially agreed with the customer representative. These requirements are going to be updated with the user's feedback after each prototype.

2.3.1 Functional

These are requirements that will become implemented functionalities of the program.

2.3.1.1 Data management

- Load patient's data series from a CSV file.
- Feature: Read from Excel and other popular formats.
- Feature: Load data from multiple files.
- Define the data samples period. This is the time between each data sample. This is used to smooth the data and for the analysis and pattern discovery process.
- Define positive special events (PSE) and negative special events (NSE). A positive special event is the event to be predicted; the negative special event is when nothing has to be predicted.

2.3.1.2 Variables

- Select the variables to be analysed.
- The variables are numeric.
- Define ranges of values for each variable; this is to discretize the values.
- The ranges cannot overlap.

2.3.1.3 Data display

- Display the data in a graph and in a table.
- Display the continuous values and the discrete values.
- Select colours for the ranges of each variable.
- The colours of the ranges are shown in the graph as a visual help.
- Display the special event in the graphs.

2.3.1.4 Data analysis

- Implement analysis modules to report statistical information from the data.
- One of these analysis modules will report the occurrences of the elementary patterns.

2.3.1.5 Pattern discovery

- Hardcoded / pre-defined pattern types.
- The user can define the patterns to be matched against the datasets.
- The user can match the patterns against the several data series.
- The system generates a match report.

2.3.2 Non functional

These are requirements that will not become functionalities, are requirements which impose constraints on the design or implementation.

2.3.2.1 Documentation

- Would be available a user manual from the main menu.

2.3.2.2 Extensibility

- The program will be general enough so that it can be applied to time series data sets from various domains.

2.3.2.3 Platform compatibility

- The program will be compatible with several operating systems and platforms.

2.3.2.4 Portability

- The program logic will be abstracted so different GUIs can be implemented easily.

2.3.2.5 Response times

- The response time of the system will depend on the size of the datasets and the quantity of patterns to be matched. However, for small datasets we expect a rapid response time.

2.3.2.6 Usability

- Friendly user interface.
- Easy visualization of the data.
- Feedback provided for most values provided by the User.
- Confirmation dialogs for irreversible actions.
- Form data validation.

2.4 Similar approaches

The following are some of the software systems (or solutions) with similar functionalities to TDWB:

- APRIORI algorithm is very useful to analyse the data.
- Semantic Web-Enabled Exploration of Temporal Information (SWEETInfo), Stanford. That uses ontologies to analyse time-stamped data sets. See: http://bmir.stanford.edu/projects/view.php/sweetinfo
- ChronoMiner, Rashmi Raj, Martin J. O'Connor, Amar K. Das, Stanford. This system, as SWEETInfo, uses ontologies to extract information from data sets.
- Temporal Pattern Discovery System (TEMPADIS), Ramirez & Cook. Used to discover patterns in temporal data.

2.5 Constraints

The main constraint is the time limit of around four months to develop the system. Ideally we should ask the medical experts to evaluate the system, although if I don't have access to these experts, then my supervisors and other research staff, who work directly with the domain experts, will be able to provide feedback on the system.

2.6 Choosing a Java graphic time series library

I am going to need a graph library to display the data sets in a time series graph. This could be a big effort so I have to decide if I am going to write my own graph library or to use a third party library.

As seen in Figure 2-1 I have implemented a very simple library to display time series graphs in a swing component.

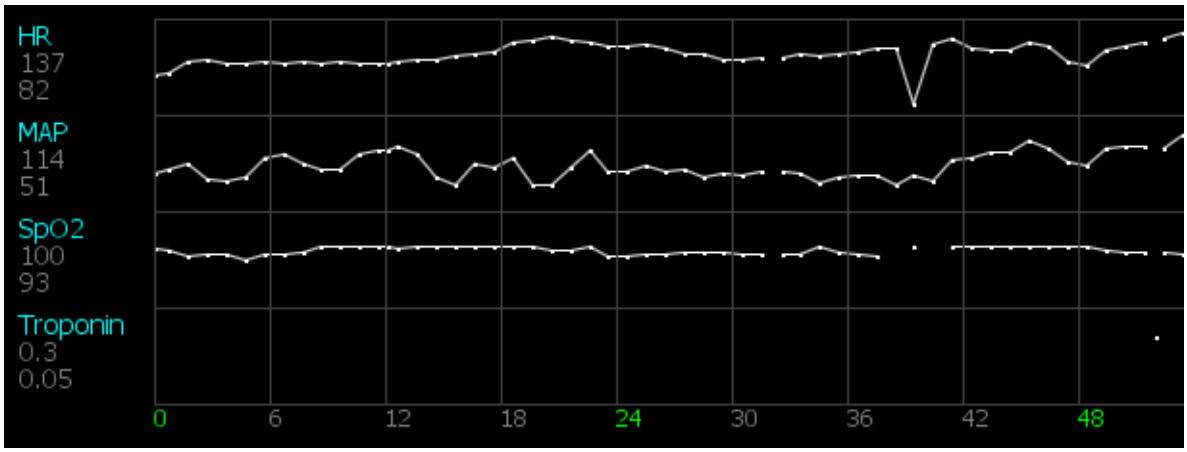

Figure 2-1 – Custom graph library

This library has not all the functionalities that I need. To develop a library with all the needed functionalities can be a large amount of work, so it is better if I try to find an existing graphs library and use it for this project.

To make use of a specialized graphics library could be very useful. Not only because the amount of code to write the program would be minor, but because I could benefit from the library's extra functionalities to easily add new features to the program.

2.6.1 Requirements needed

The library must satisfy various requirements. The library not only has to satisfy functional requirements, has to satisfy non-functional requirements too. The overall requirements depend on the requirements of the program.

2.6.1.1 Functional requirements

Draw time series charts

The data series analysed in the program are time series, so the library must have functions to manage and display time series.

Draw discontinued time series

Sometimes there are missing values in the data series to analyse. The chart library must deal with missing values.

Export graph as an image

It is not essential, but it is nevertheless desirable that the user can save graphs as bitmap images.

2.6.1.2 Non-functional requirements

Java compatible

As seen in the section 3.7.2, the chosen programing language is Java. So the library must be Java compatible.

Swing compatible

As seen in the section 3.7.3, the chosen IDE is NetBeans which has a Swing UI builder. So the library must be compatible with Swing or easily embeddable in a Swing component.

Documentation

I need documentation to know how to use the library.

Tutorials

Some simple tutorials that explain how to set up my program to use the library functions would be very useful. If someone has written tutorials about advanced uses of the library that would also be helpful.

Active community

Sometimes the libraries have some bugs, have some tricky functionality or the documentation is not clear. Most of these bugs and functionalities are discussed in forums or blogs.

Free

The library must be free for educational purposes.

Ease of use

I will use a library because the amount of work would be considerably less than if I have to write a whole graphics library myself. If the library is extremely difficult to use then it would not be sensible to use it.

Stable

If I am going to use a third party library then I need than it always works correctly.

Performance

Maybe the program is going to manage large amounts of data. So I need an efficient library.

Open source

If there is some bug in the library’s code, or if I need to tweak its performance then the library has to be open source so I could improve it.

2.6.2 Reviewed libraries

2.6.2.1 Chart2D

Has active developers, documentation, some tutorials but is not very extended, has not a big community and has not time series support.

URL: http://sourceforge.net/projects/chart2d/

Figure 2-2 – Chart2d

2.6.2.2 ChartDirector

License needed, so it is discarded.

URL: http://www.advsofteng.com/cdjava.html

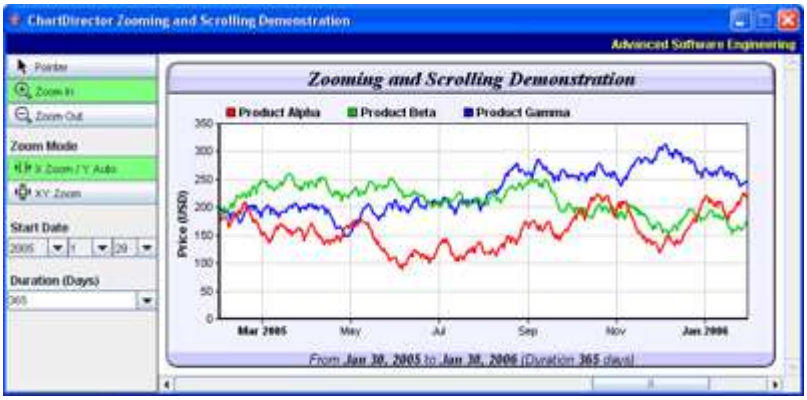

Figure 2-3 – ChartDirector

2.6.2.3 G

The last version was in December 2009 and has not time series support.

URL: http://geosoft.no/graphics/

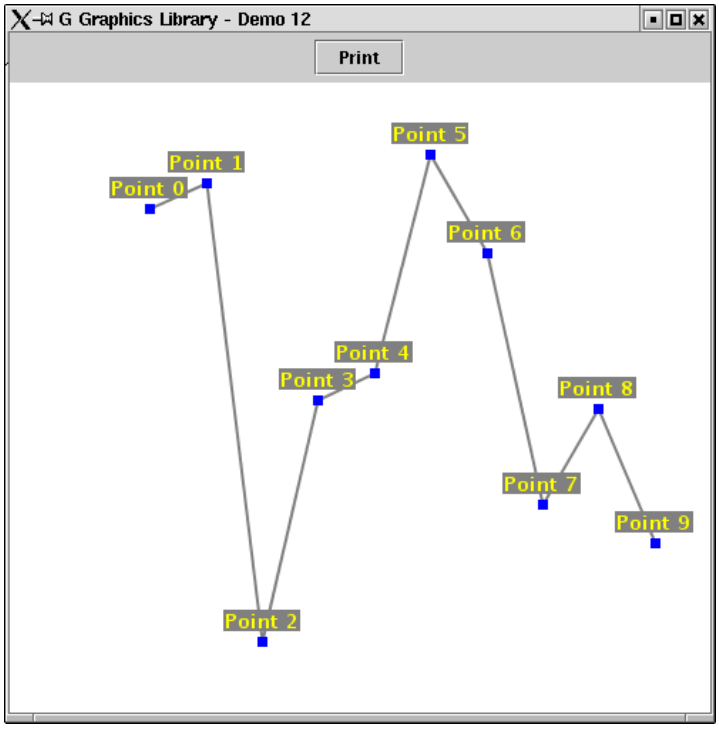

Figure 2-4 – G

2.6.2.4 JCCKit

Has not an active community, doesn't support time series and the last version was in 2004.

URL: http://jcckit.sourceforge.net/

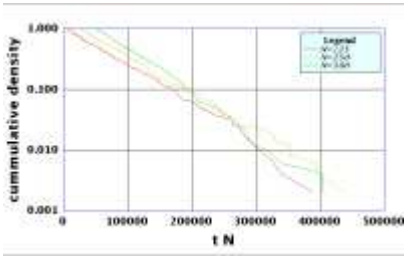

Figure 2-5 – JCCKit

2.6.2.5 JChart2D

It is very complete, have lots of functionalities and an active community.

URL: http://jchart2d.sourceforge.net/index.shtml

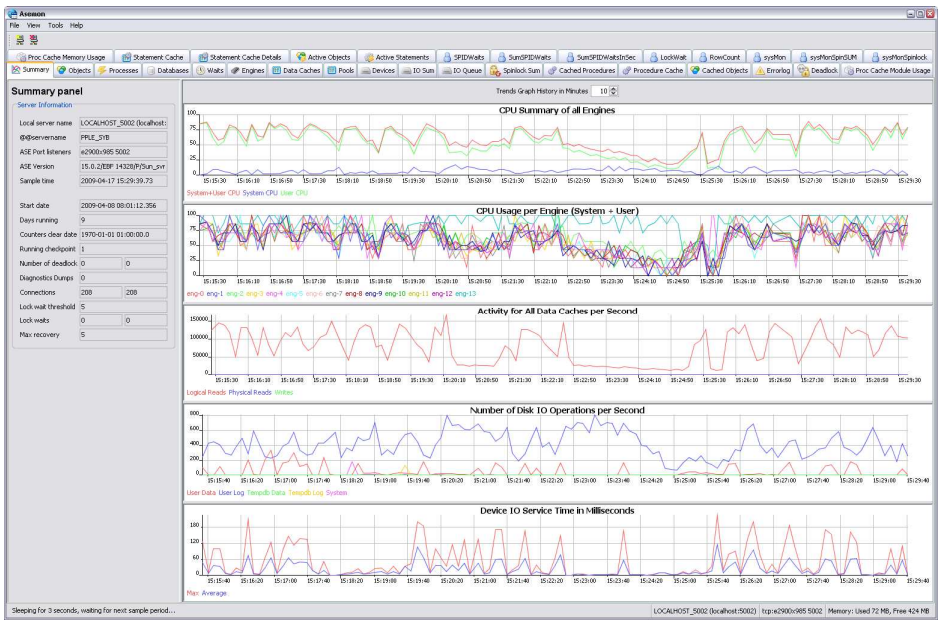

Figure 2-6 – Jchart2D

2.6.2.6 JChartLib

Has a big community and tutorials but in its web site there are not much examples and the documentation is not free.

URL: http://sourceforge.net/projects/jchartlib/

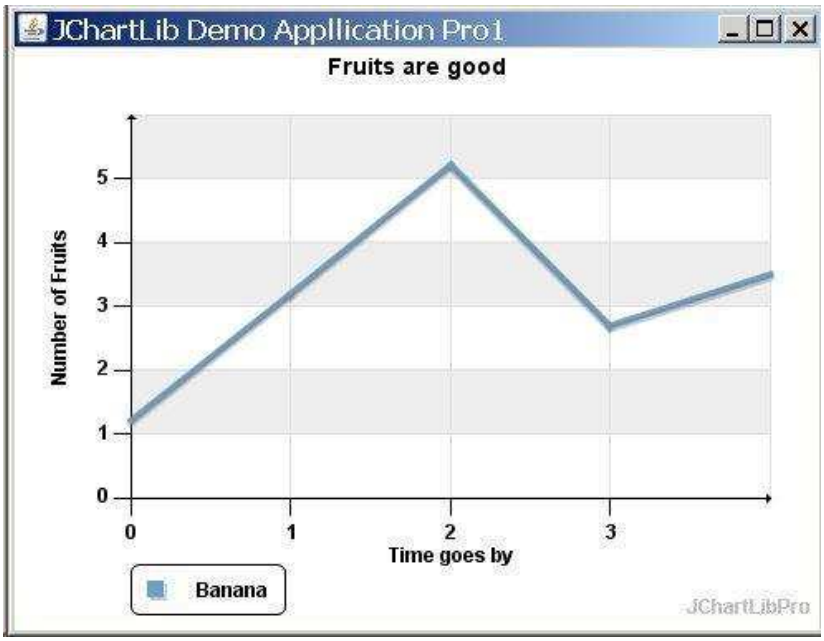

Figure 2-7 – JchartLib

2.6.2.7 jCharts

Has basic functionalities and the last version is from 2003.

URL: http://jcharts.sourceforge.net/

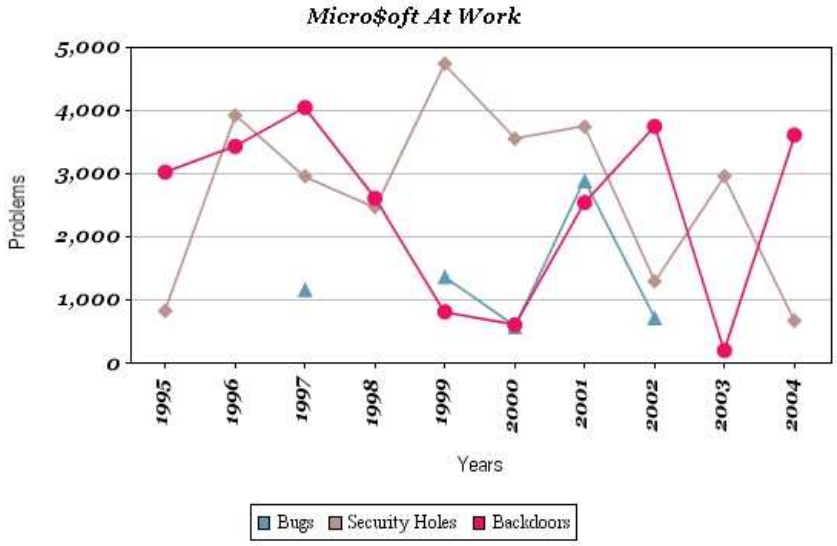

Figure 2-8 – jCharts

2.6.2.8 JfreeChart

It has lots of functionalities, lots of examples, lots of tutorials, good documentation, and good community.

URL: http://www.jfree.org/jfreechart/

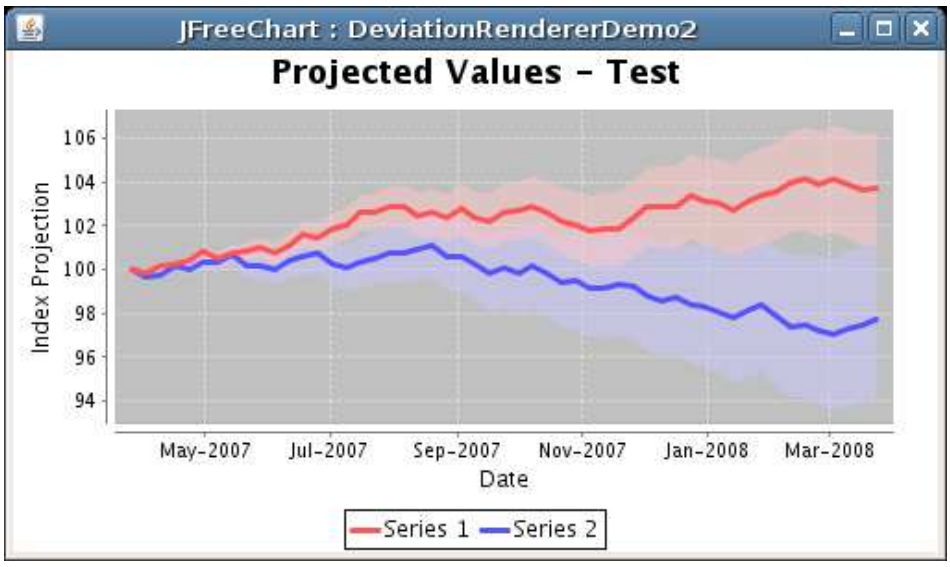

Figure 2-9 – JfreeChart

2.6.2.9 JOpenChart

The last version is from 2002. The community and the tutorials are very poor.

URL: http://jopenchart.sourceforge.net/

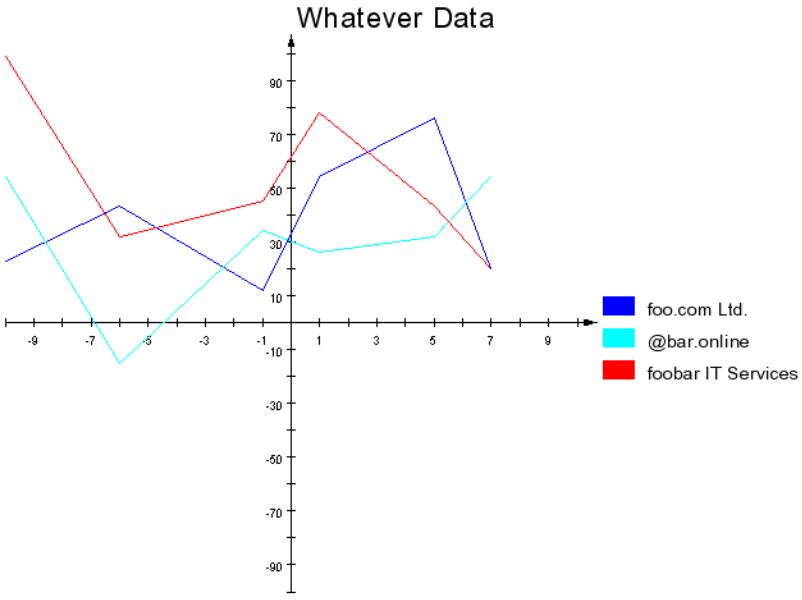

Figure 2-10 – JopenChart

2.6.2.10 Ptplot

It is developed for drawing functions and not time series.

URL: http://ptolemy.eecs.berkeley.edu/java/ptplot/

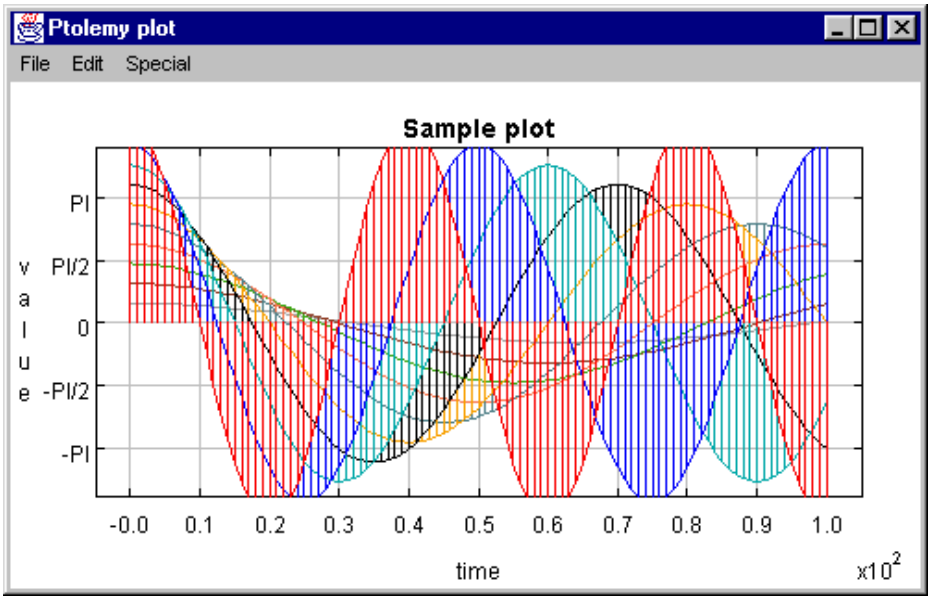

Figure 2-11 – Ptplot

2.6.3 Comparative

All the compared libraries are Java, Swing compatible, open source and free for academic educational purposes. These are mandatory requirements.

	Chart2D	G	JCCKit	JChart2D
Time series	No	No	No	Yes
Discontinued TS	No	No	No	?
Export as image	?	?	?	Yes
Documentation	Yes	Yes	Yes	Yes
Tutorials	Poor	Good	Poor	Good
Community	Poor	Poor	Poor	Good
Ease of use	?	?	?	?
Stable	Yes	?	?	Yes
Performance	Good	Good	?	Good

	JCHartLib	JCharts	JFreeChart	JOpenChart	Ptplot
Time series	Yes	Yes	Yes	No	No
Discontinued TS	?	Yes	Yes	No	?
Export as image	Yes	Yes	Yes	?	?
Documentation	Not free	Yes	Yes	Yes	Yes
Tutorials	Good	Poor	Good	Poor	Poor
Community	Good	Poor	Good	Poor	Poor
Ease of use	?	?	Medium	?	?
Stable	Yes	?	Yes	?	?
Performance	?	?	Good	?	?

2.6.4 Chosen library

JFreeChart and JChart2D are the only one that offers all the requirements. JFreeChart has better documentation, better examples and a more active community. I have tried it and the results are very good, so I have chosen JFreeChart as the graph library used by TDWB.

I tried to draw a similar graph to the generated graph by my own custom library and the result, as seen in Figure 2-12, is much better than the given by my custom library.

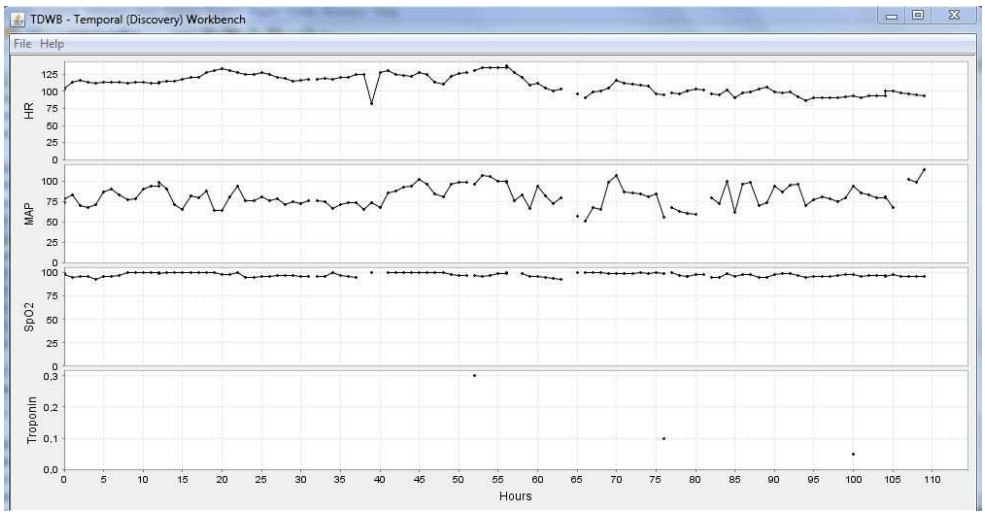

Figure 2-12 – JfreeChart test

2.6.5 Extra features

Plus, JFreeChart offers these interesting extra features.

Statistical functions, like mean, medium and correlation

Are functionalities not required for the current specifications but could be useful for further work.

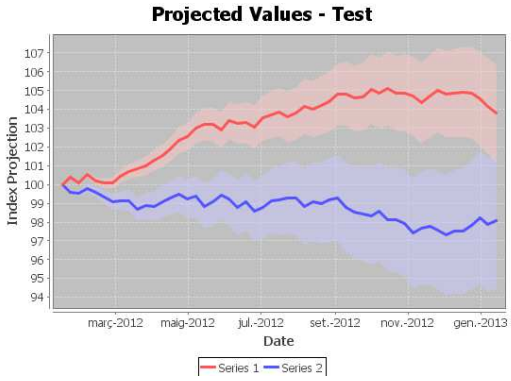

Figure 2-13 – Deviation demo

Annotations

This could be good to mark determined values.

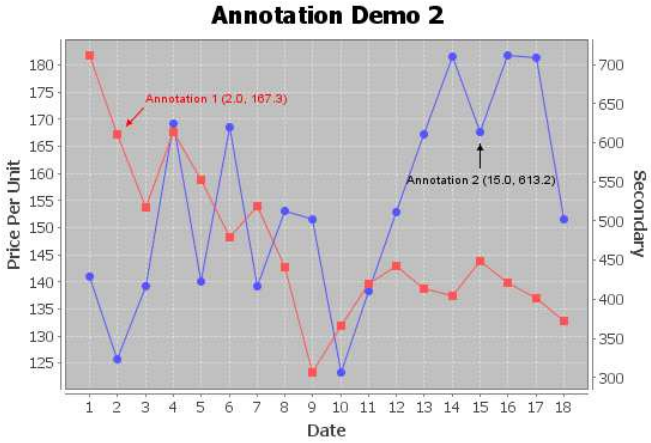

Figure 2-14 – Annotations

Box annotation

This is useful for discrete graphs colouring.

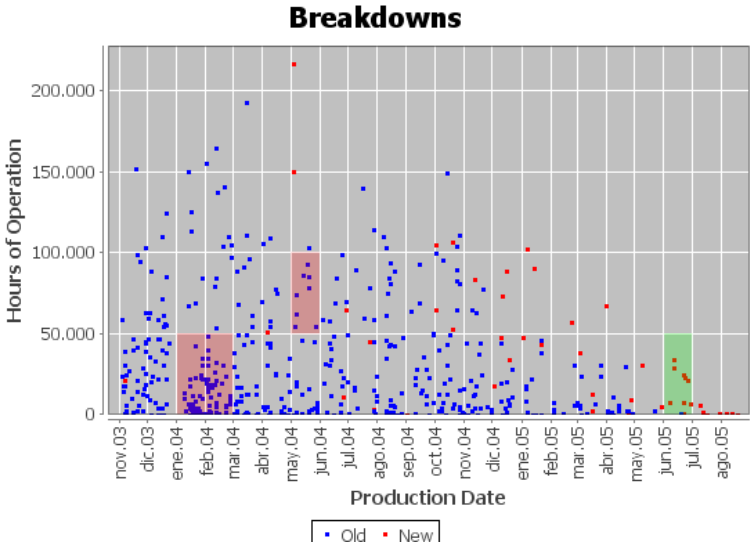

Figure 2-15 – Box annotation

Marker

It is useful for mark the special events.

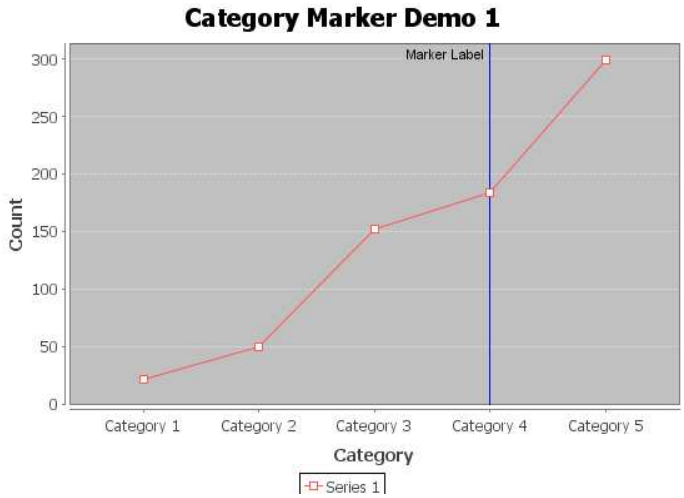

Figure 2-16 – Marker demo

2.6.5.1 Support

There is a support forum and the API Documentation is free. But on the other hand, the developer documentation is not free. However, I found some tutorials:

http://www.vogella.de/articles/JFreeChart/article.html

The developer documentation of an older version:

http://ktipsntricks.com/data/ebooks/java/jfreechart-0.9.1-US-v1.pdf

Some demos:

http://www.java2s.com/Code/Java/Chart/Time-Series-Chart.htm

http://www.java2s.com/Code/Java/Chart/JFreeChartTimeSeriesDemo8.htm

3 Outline design

This chapter describes an initial design of a possible solution to this problem.

3.1 General use workflow

Figure 3-1 shows the typical stages of the program/analysis. First load data, then select the variables to be included in this analysis, specify their discrete ranges, analyse the data, and finally search for patterns that match the data.

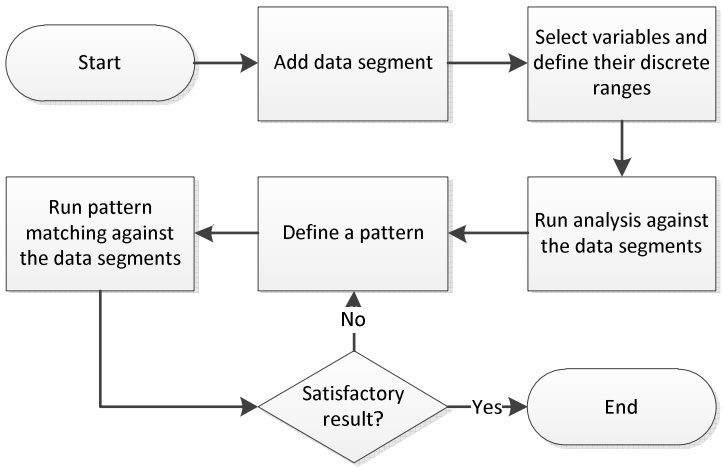

Figure 3-1 – General use workflow

3.2 Data management

Data segments

The user must load the data to be analysed. The system requires that files be formatted to comply with particular conventions.

TDWB requires one specific variable in each data file to load them, it is the **Time** variable. This variable is the time stamp for each data record. The formats accepted by this program for the time stamps are:

```
dd-MM-yyyy kk:mm:ss.SSS
dd-MM-yyyy kk:mm
dd-MM-yyyy kk:mm:ss
yyyy-MM-dd kk:mm
yyyy-MM-dd kk:mm:ss
yyyy-MM-dd kk:mm:ss.SSS
dd/MM/yyyy kk:mm
dd/MM/yyyy kk:mm:ss
dd/MM/yyyy kk:mm:ss.SSS
yyyy/MM/dd kk:mm
yyyy/MM/dd kk:mm:ss
yyyy/MM/dd kk:mm:ss.SSS
yyyy.MM.dd G 'at' HH:mm:ss z
h:mm a
yyyyy.MMMMM.dd GGG hh:mm aaa
yyMMddHHmmssZ
yyyy-MM-dd'T'HH:mm:ss.SSSZ
```

Where, the pattern letters are described in Figure 3-2:

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Figure 3-2 – Date time pattern letters

Special event

The user has to define the time of the special events to be predicted. The data before each special event constitutes a data segment. If a file has multiple special events, then the file also has multiple data segments.

The special events can be POSITIVE or NEGATIVE. If it is positive, means that a special event happened; if it is negative, means that there is not special event. This is useful for the training data segments.

Continuous values

The “Continuous values” are the result of applying a smoothing process to the original values. First, the system generates new time stamps at a regular period. Then, the values are reallocated to the new time points. If more than

one value is in the same period, the mean of these variables is used as the value of that time period. Figure 3-3 illustrates this process.

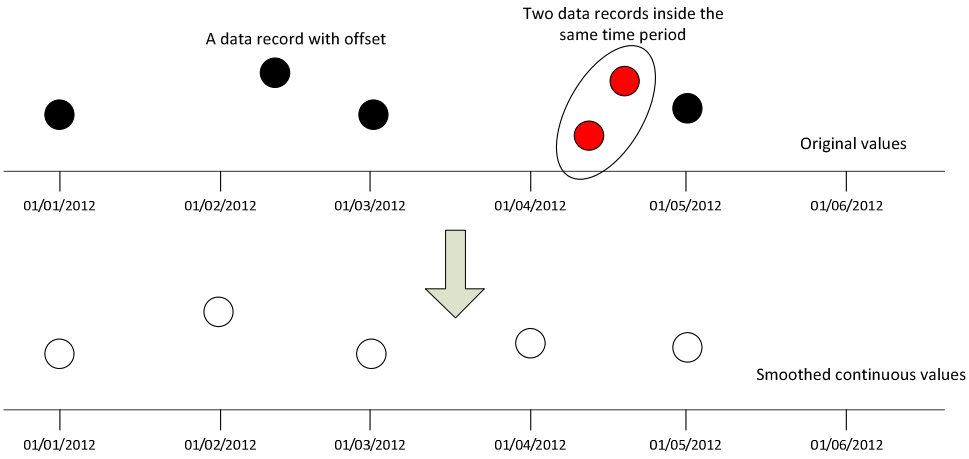

Figure 3-3 – Data smoothing

Discrete values

After generating the smoothed continuous values, the system has to generate the discretized version of these values. This process is done by determining the value range to each smoothed continuous value. The Figure 3-4 illustrates this process.

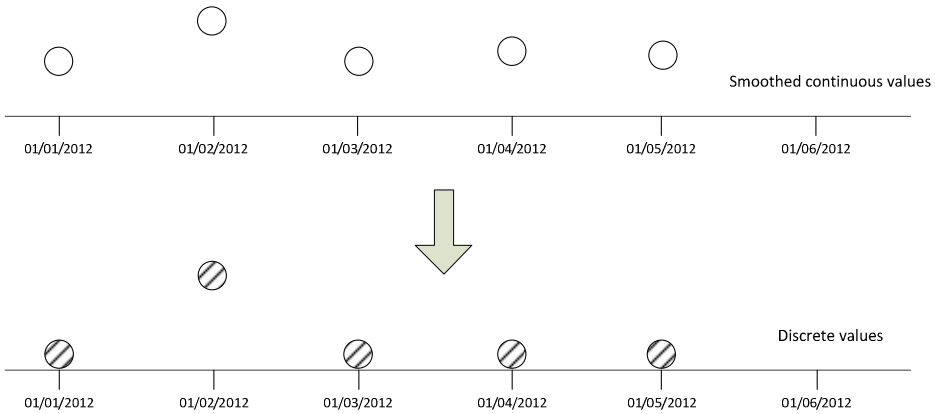

Figure 3-4 – Data discretizing

3.3 Variables

The user selects the variables to be analysed. These variables have to be numeric values, like integer or decimal values.

Discrete ranges

It is very difficult to analyse continuous variables, and so we made a decision to only analyse data which had been assigned to ranges. Classifying the values in ranges is a method to discretize the data. For example, the user could classify the heart rate values as very low (VL), normal (N) and very high (VH). These discrete ranges for each variable are the elementary patterns.

These ranges cannot overlap, because if two ranges overlap and a value belongs to the two ranges then the system has to deal with an ambiguity.

The system has to provide an intuitive interface to determine the value ranges of each variable.

3.4 Data display

The system has to display the original variables values, the continuous and the discrete versions of the values. JfreeChart is going to be used to display the graphs, and a JTable Swing component to display the data values.

For the discrete graph, each value is going to be coloured. These colours describe the value ranges and are used as a visual aid.

The special events have to be marked in the graph to visually see when a data segment finishes.

3.4.1 Chart library interface Class

I am going to use JfreeChart library as the graph library to display the data time series. Maybe this library has some major bug or a lack of functionality that now

I don’t know, or maybe a new version is distributed. Then, then library has to be easy of change by other library that is not necessarily a new version of JfreeChart. To resolve this I am going to write an interface class as a proxy to use the chart library functions. Then, to change the library the proxy class is the only class to be modified.

3.5 Data analysis

The system provides algorithms to statistically analyse the data segments. This information should be useful for the analyst when determining the kind of patterns to develop.

As stated in section 2.2, I am going to implement analysis modules to report the occurrences and the changes of the discrete values.

Should be easy to add or remove analysis module, so I am going to make this modifications easy to achieve by writing an interface class- that the analysis modules must implement- and a proxy class that will interact between the analysis modules and the system.

3.6 Pattern discovery

After loading the data files, selecting the variables, defining their discrete ranges and reviewing the analysis report, the user can evaluate the patterns developed against the data segments. All the data segments end with a positive- or a negative- special event. Each pattern is matched against the discrete values of the variables of each data segment. The workbench then reports which “positive” and which “negative” data sets are matched against the pattern. A typical report is shown in Figure 3-5.

Pattern matching result

	SE	NSE
SE Match	25/33: 75.8% True +	16/20: 80% False +
NSE Match	8/33: 24.2% False -	4/20: 20% True -

Figure 3-5 – Pattern match result

The categories used in Figure 3-2 are defined below:

- True Positive, if the pattern matches a data segment which contains a positive special event.
- True Negative, if the pattern doesn't match a data segment with a negative special event.
- False Positive, if the pattern doesn't match a data segment with a positive special event.
- False Negative, if the pattern matches a data segment with a negative special event.

A perfect pattern will match only the data segments with a positive special event and none of the data segments with a negative special event (NSE).

In the next sections different patterns defined for the myocardial damage domain are described.

3.6.1 Single time point pattern

The elementary components of the patterns are the duplets in a format “Variable name [Discrete range]”. For example, for the variable with name ‘X’ with discrete ranges [V1, V2, V3, V4], a possible tuple would be “X[V2]”, or “X[V1]”. The single time point patterns are checked at each single time point of every data segment.

For example, the pattern “X [V3]”, for the sequence:

```
X: V1 V1 V2 V3 V3
```

Reports positive matches:

```
X: V1 V1 V2 V3 V3
```

3.6.2 Single time point composite pattern

The system matches patterns at the same time for more than one variable. For example for the variable ‘X’ with discrete ranges [V1, V2, V3, V4] and the variable ‘Y’ with discrete ranges [V1, V2, V3] the user can check if “X[V2] AND Y[V1]” happen at the same time point.

For example, the pattern “X[V3] AND Y[V2]”, for the sequence:

```
X: V1 V1 V2 V3 V3

Y: V1 V1 V1 V1 V2
```

Reports a positive match:

```
X: V1 V1 V2 V3 V3

Y: V1 V1 V1 V1 V2
```

3.6.3 Temporal pattern

The user can specify a composition of elementary patterns with a specific time offset. The time offset parameter is added to the elementary pattern which is a triplet with the format “T+ Time offset: Variable Name[Discrete range]”.

For example, the pattern “T+0:X[V3] AND T+1:X[V3]”, for the sequence:

```
X: V1 V1 V2 V3 V3
```

Reports positive matches:

```
X: V1 V1 V2 V3 V3
```

3.6.4 Temporal composite pattern

The user can combine temporal patterns of a variable with the temporal patterns of other variables.

For example, the pattern “T+0:X[V3] AND T+1:Y[V3]”, for the sequences:

```
X: V1 V1 V3 V3 V3
Y: V1 V1 V3 V2 V3
```

Reports positive matches:

```
X: V1 V1 V3 V3 V3
Y: V1 V1 V3 V2 V3
```

3.6.5 Combinatorial pattern

The system can look for different combinations of single time point and single time point composite patterns. This is denoted with commas, like: “A,B,C” or “A,A,A,B”.

For example, for the combinatory pattern “A,B,C”, where A=X[V1], B=X[V2], C=X[V3] AND Y[V2], the system expands to:

```
T+0:A AND T+1:B AND T+2:C OR
T+0:A AND T+1:C AND T+2:B OR
T+0:B AND T+1:A AND T+2:C OR
T+0:B AND T+1:C AND T+2:A OR
T+0:C AND T+1:A AND T+2:B OR
T+0:C AND T+1:B AND T+2:A
```

3.6.6 Combinatorial M-out-of-N pattern

This is similar to the previous pattern, but here interleaved gaps are added. The gaps are denoted by a ‘G’.

For example, the pattern “A, B, G” is expanded to the pattern:

T+0:A AND T+2:B (i.e. G occurs at T+1)

The pattern “A, B, C, G” is expanded to the pattern:

T+0:A AND T+1:B AND T+2:C OR

T+0:A AND T+1:B AND T+3:C OR

T+0:A AND T+2:B AND T+3:C OR

T+0:A AND T+1:C AND T+2:B OR

T+0:A AND T+1:C AND T+3:B OR

T+0:A AND T+2:C AND T+3:B OR

T+0:B AND T+1:A AND T+2:C OR

T+0:B AND T+1:A AND T+3:C OR

T+0:B AND T+2:A AND T+3:C OR

T+0:B AND T+1:C AND T+2:A OR

T+0:B AND T+1:C AND T+3:A OR

T+0:B AND T+2:C AND T+3:A OR

T+0:C AND T+1:A AND T+2:B OR

T+0:C AND T+1:A AND T+3:B OR

T+0:C AND T+2:A AND T+3:B OR

T+0:C AND T+1:B AND T+2:A OR

T+0:C AND T+1:B AND T+3:A OR

T+0:C AND T+2:B AND T+3:A

3.7 Methodology and technologies

3.7.1 Software development method

The traditional way of software development is a linear sequence:

Requirements - Design – Development – Testing – Feedback

But in this case there are some important factors which make the “linear” approach less viable:

- I have just a general idea about the problem being addressed. So it is better to do a quick/early prototype and then produce an enhanced version which incorporates changes proposed by the expert-users.
- I know which kind of users are going to use the program but I don’t initially know the kind of interface that is appropriate. In any event I am planning to separate the logic from the UI; the latter can then be a simple interface or it could be a web interface.
- It is acceptable for me to use libraries or tools at my disposal; but of course their use must be acknowledged. Some of these libraries are going to be used in the initial UI, as I am not sure about the UI design. When I have more detailed information then I will better appreciate the services I need from a third party library.
- I should generalize the program, to solve a wide range of problems. Also, I plan to morph the workbench into a framework. But first I need a working program for the specific problem and then I will generalize it

I am exploring a novel and slightly under-specified task. It would be a colossal amount of work to modify or redo all the documentation and processes involved in a traditional development method like Rational Unified Process (RUP) or the Waterfall model. The philosophy that fits this situation is one like Agile Software Development [AG 1] [AG 2] which could be implemented using a methodology such as Extreme Programming (XP) [XP 1] [XP 2]. However, I am a team of one, so I will follow this method: I will build a fast and simple prototype, show it to the customer representative (tutors) and to the stakeholders (experts). Then, take notes of the required/suggested changes, and then modify or rebuild the prototype. This process will end when all the parties/partners agree about the functionalities offered by the final prototype or we are close to the end date. A proposed method:

Initial Requirements –Design – Coding+Test – Doc–Feedback – Design – Coding+Test – Optimization - Doc

The last part, after the prototyping cycle, is to tidy up the code and add some extra functionalities if needed.

3.7.1.1 Design

I am going to use standard flow charts and UML diagrams. But at first I will only develop a simple version because they will certainly be modified at later stages. I am planning to use UML Diagrams because these are a standard for Object-Oriented Programming (OOP) and I am familiarized with them.

I am going to use Microsoft Visio to create and maintain the UML diagrams, because I have used it in the past and I know that it is easy to use.

3.7.1.2 Prototyping

At first, the program doesn't need a "heavy" data management facility as, at least initially, only CSV files are to be used for entry data. This will simplify the code considerably.

An IDE with a graphical GUI designer will be a great help.

I am working alone so for the prototypes I can mix the code a little bit by not splitting it in a typical architectural pattern, like Model-View-Presenter (MVP). And only comment the important parts for my future revision of the code.

I am a team of one, so I cannot assist team meetings as XP recommends. But my tutors will play the role of co-workers and will share points of view at our weekly meetings.

3.7.1.3 Testing

Every class needs a Unit Test. XP method recommends writing the Unit Tests before the code. Essentially, I will write the properties and the headers of the methods, then the tests and, finally, the body of the methods. This will give me a clear perspective.

The GUI testing will be carried out by the users at the end of each prototyping cycle.

3.7.2 Programming Language

I have background in widely used OOP languages like C++, Java and Python; additionally I have experience of web technologies. Initially, the users are going to be only a specific group of experts, so a PC-based system is acceptable.

The program is probably going to use some libraries written in Java, like graph libraries. There are some IDEs with a graphical GUI designer for Java. If the program is growing and if it is written in Java, then it can be reformulated as a web service. For these reasons, Java is the language that I choose as the logic

and the UI components of the systems mentioned earlier can be implemented in it. Java's great strengths are that it allows libraries to be used (these libraries are now very extensive) and it is platform independent. This approach will allow me to have, for instance, a number of UIs including one to the WWW, if required.

3.7.3 Integrated Development Environment (IDE)

An IDE is a great tool for developing a program. It has a lot of useful functionalities to assist the programmer and to maximize the productivity. I need an IDE with the following functionalities and requirements:

- Supports Java
- Free license
- Supports control version
- Visual GUI builder
- Refactoring
- Debugging
- Unit Test

So, after check the list:
http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#Java

I short-listed Eclipse and NetBeans. I have used both IDEs before and both are good enough, but as I have never used the GUI builder of Eclipse, I decided to use NetBeans.

3.7.4 Repositories and Backups

A good way to keep track of the changes and to share the code and the documents with my tutors is a revision system. I have tried Subversion (SVN)

and Git. However, Git has more functionality and is better at merging changed files than SVN. SVN is easier to use and I don't need the extra functionalities of GIT, so I decided to use SVN.

Also, SVN can be a backup system for the project. But sometimes, for certain mistakes or malfunctions, the repository gets corrupted or lost. So I decided to use an auxiliary tool for the backups. DropBox is a web service to store files in the cloud. The files are uploaded automatically every time they are saved to disk. Also, with DropBox, I can get access to the files from the web, share them and have control version. (Additionally, I will use the UoA user space to backup weekly copies of all the PFC material.)

The subversion project is structured in two folders, 'trunk' and 'tags'. In the trunk folder will be the working copy of the NetBeans project and in the tags will be the finished prototypes that I produce.

3.7.5 Program versions

The prototypes are numbered with the convention 0.X.Y where 'X' means the number of the prototype and 'Y' means the versions of the prototype.

The beta versions for the final program are numbered with the convention 1.0bX where 'X' means the version of the beta.

The final and definitive version for this project is the 1.0.

4 Prototype 1 (TDWB 0.1)

After outlining the initial design, I produced the first prototype, in which all the functionalities were not implemented. However, it provides a general idea about the final program. Only one analysis module has been implemented, the "One variable, one change" test, which is the simplest.

With the development of this prototype, the main infrastructure is set up; this was a considerable amount of work. So, the further development phases are expected to be shorter than this one.

4.1 Design

4.1.1 Use workflow

The pattern discovery process is not implemented in this version.

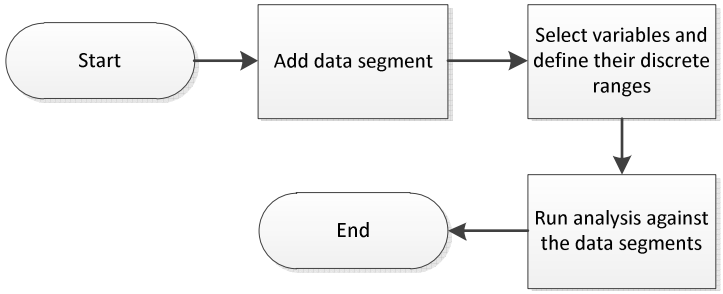

Figure 4-1 – Use workflow

4.1.2 Functionalities list

This is the list of the implemented functionalities.

4.1.2.1 Data management

Load data from a CSV file

¡Error! No se encuentra el origen de la referencia. shows how to load data in to the program from a file. The button “Load Data” starts the dialog to load the data. “Load Test Data” loads automatically the variables HR, MAP, SpO2 and Troponin from an existing CSV file. This button is only for testing purposes and is not going to be present in the final version of the program.

The program allows the user to choose between different CSV formats, like semicolon or colon separated columns. The user can decide whether the first row of the CSV file provides names of the variables or whether it is part of the data set.

After previewing the data, the user can decide which variables to load. The user can change the variable names, as blank names and repeated names are not allowed. It is necessary to select at least one variable, before a data file is loaded.

The program recognizes which variables are time variables and allows the user to pick one of them as the main time variable. A variable is a valid time variable if all its values are not blank, and are in a correct time format. The currently accepted time formats are hardcoded. (A **possible feature** would be to allow the user to add more time formats.) At least one time variable is required to load the data.

The loaded data replaces the previously loaded data in the program's main memory.

Data granularity

As seen in the chapter 3.2, the data is smoothed and then discretized. To smooth the data, the user must determine the period to generate the new time stamps. This period is the data granularity that determines the smoothing factor, as less granularity value then the data is less smoothed. By default the granularity is 1h (hour). The program analyses the time points and suggests a granularity value that is automatically selected.

Special events

The user can set the time point which corresponds to a Special Event (SE) event. There is a button in the interface to set the time of this special event to the last time point.

For now, the user can pick one and only one special event. There isn't the option of mark the event as "Not special event" (NSE).

4.1.2.2 Variables

Discrete ranges

The user can add, delete and modify the ranges for the values of each variable. Also, the user can change the colour of each range. In the **initial requirements** we decided that the ranges don't overlap so when the user applies any changes to the discrete ranges, the program checks if the ranges overlap.

4.1.2.3 Data display

The data is displayed in a table on the bottom left and as a graph in the top. The JfreeChart graphs can be zoomed, printed and saved as a bitmap image.

The user can choose how to view the data; the options being: "Original values", "Analysis continuous values" and "Analysis discrete values". The graphs and the data table change when the user selects one of these options.

To draw a time series using the JfreeChart library is easy, but to change the style of the graphs- like the colours and the shapes of the lines- were not trivial.

4.1.2.4 Data analysis

Analysis modules

In the analysis panel, as shown in Figure 4-8, we can activate the analysis module. In this version, the "One variable, one change" module is the only analysis module implemented.

Report

After the analysis, the system displays the result on the screen.

4.1.3 UI

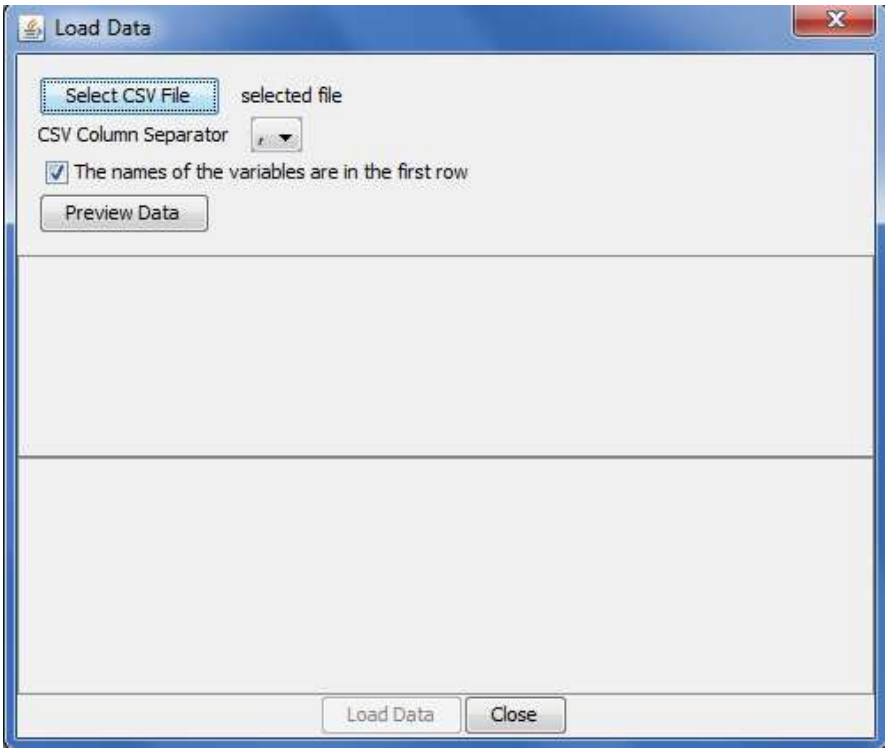

Figure 4-2 – Load data dialog

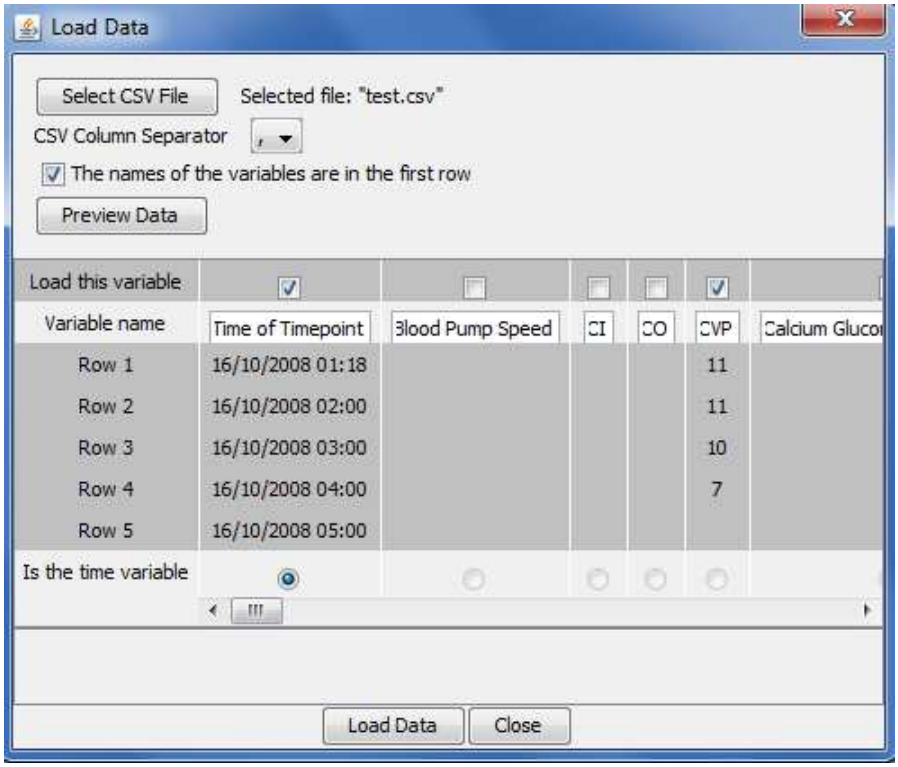

Figure 4-3 – Load data dialog with data preview

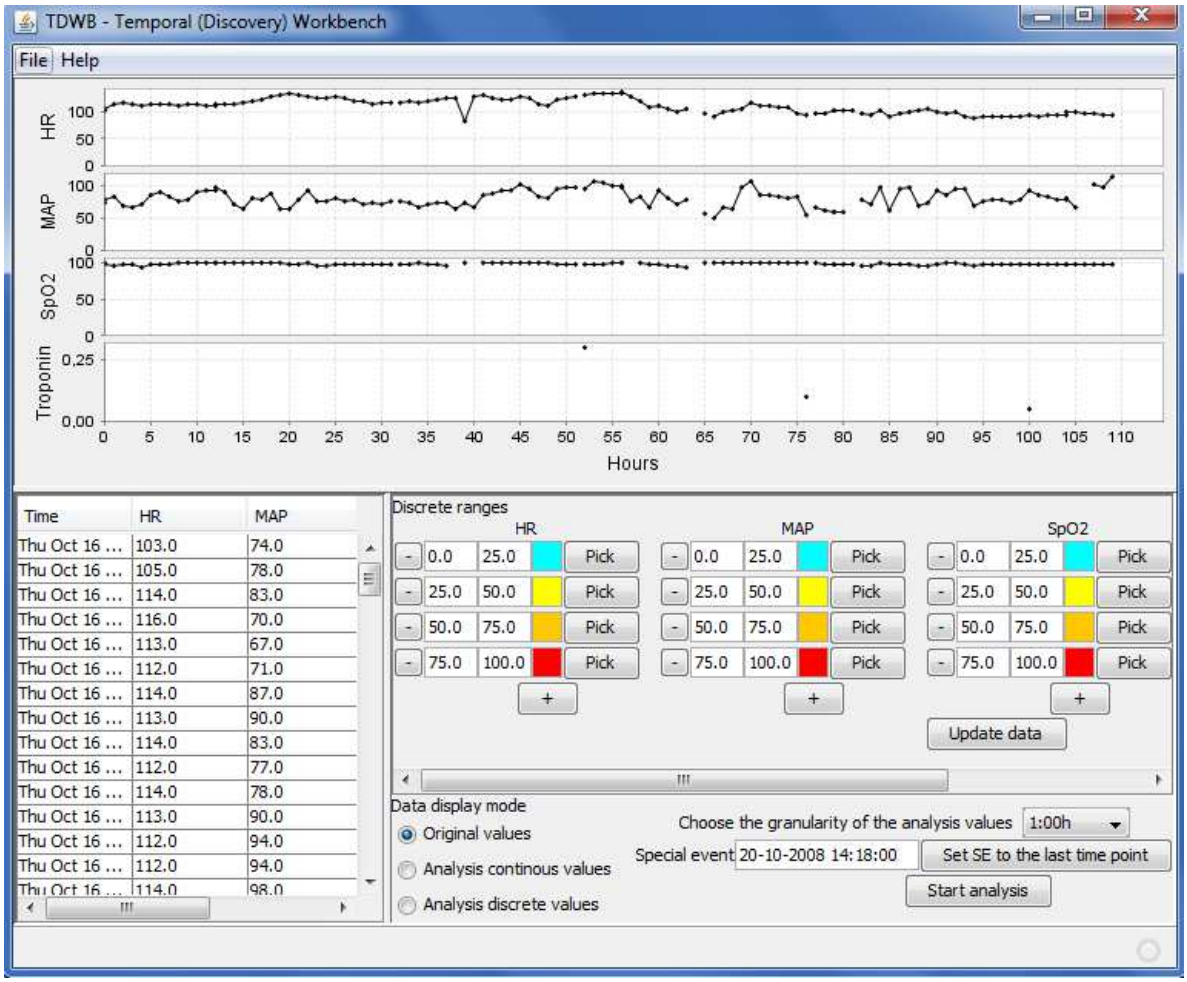

Figure 4-4 – Data panel

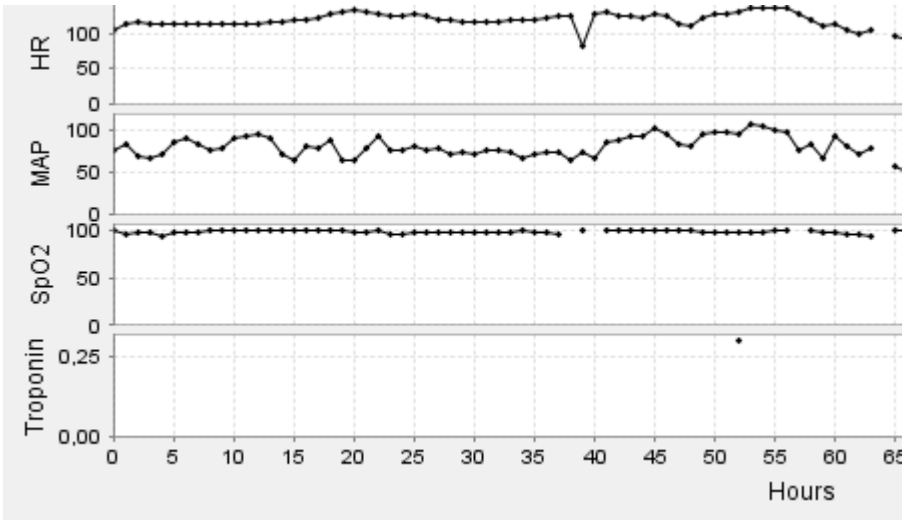

Figure 4-5 – Smoothed continuous values

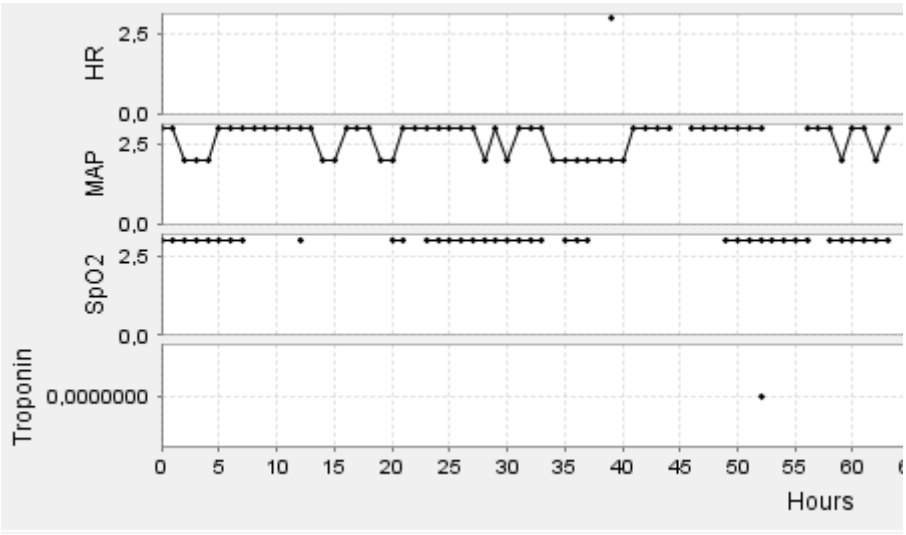

Figure 4-6 – Discrete values

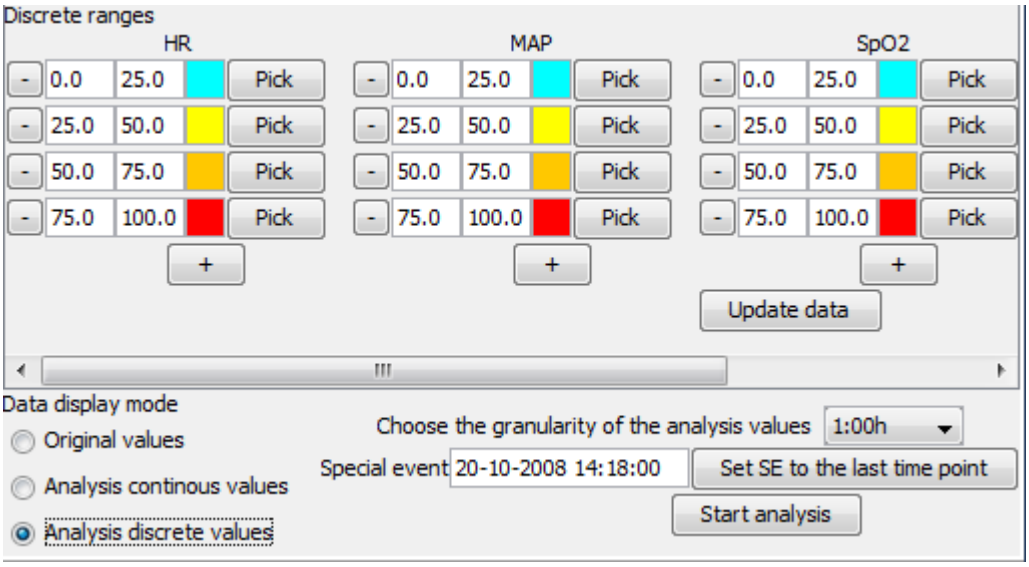

Figure 4-7 – Analysis options

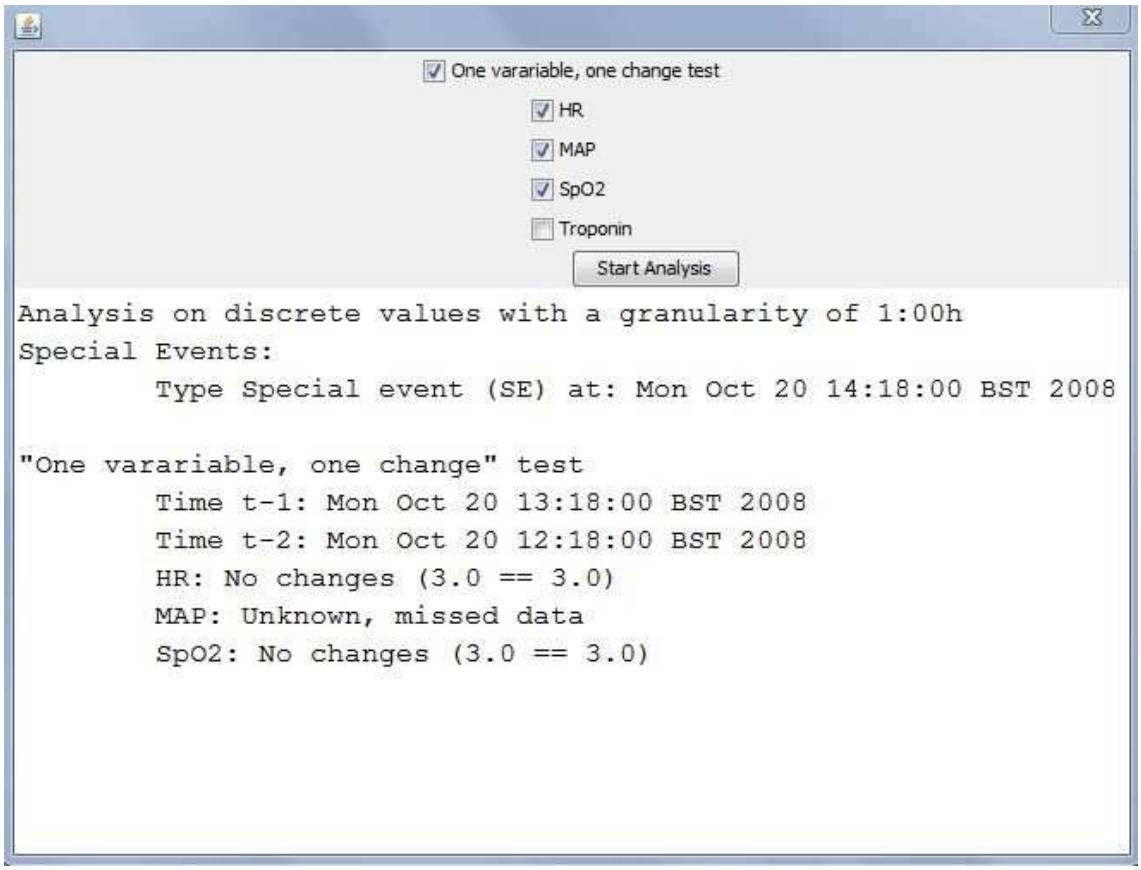

Figure 4-8 – Analysis report

4.1.4 UML

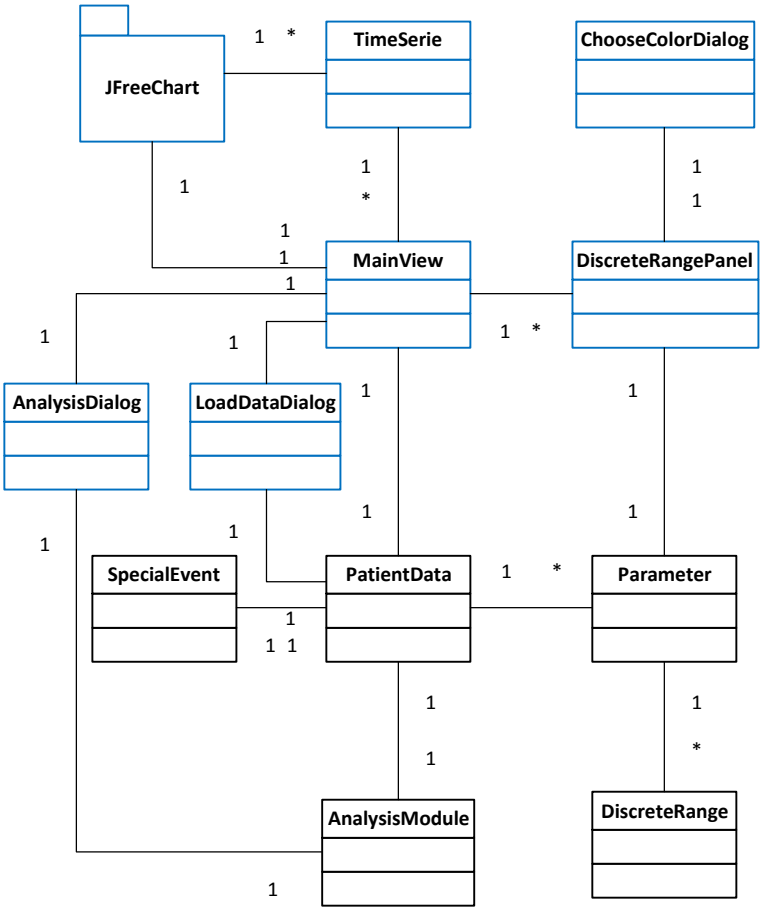

Figure 4-9 – UML

4.2 Users evaluation

4.2.1 Proposed changes

This is the list of changes proposed by Derek Sleeman and Wamberto Vasconcelos after evaluate the system on 16 November 2011. A detailed discussion of the actual changes applied can be found in the functionalities section 5.1.2 of the next version of the program.

4.2.1.1 Data management

- Change the name of the parameter “Data granularity” to “Time point period”.

4.2.1.2 Variables

At “Discrete ranges” section:

- Change text of the button “Update data” to “Apply changes”.
- Check that the min value of a range is less than or equal to the max value.

5 Prototype 2 (TDWB 0.2)

For this version, multiple files can be loaded into the system. Also, the “project” concept and the pattern matching process have been implemented. The model layer has been completely changed and a radical redesign of the GUI has been undertaken, therefore I have rewritten almost all the source code.

5.1 Design

5.1.1 Use workflow

The user can now load more than one file in the system. Also, the pattern discovery process has been implemented. But, to simplify the code, the analysis module has been removed.

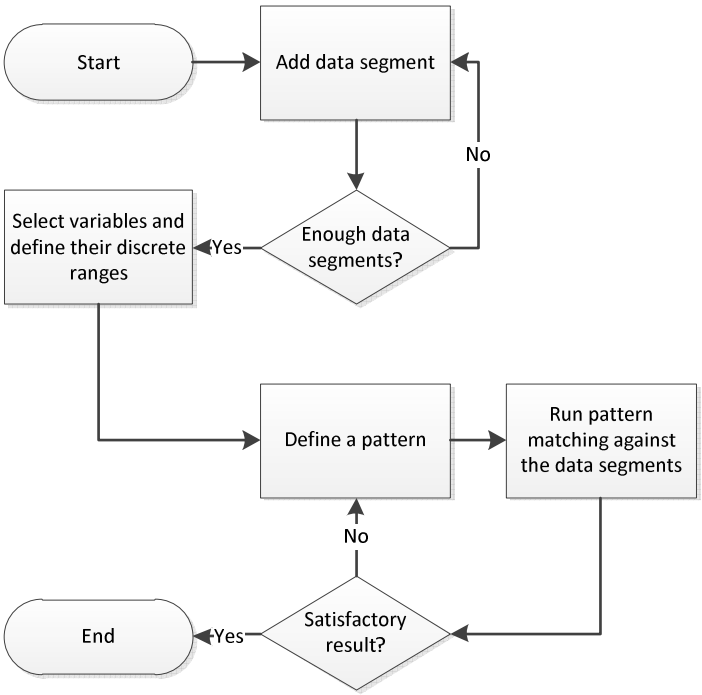

Figure 5-1 – Use workflow

5.1.2 Functionalities list

5.1.2.1 Project

The concept of “project” has been implemented. A project is a conjunction of data files, selected variables, discrete ranges and patterns.

Save/Load the project into/from files

These are simple but important functionalities because the users need to save their work for further purposes. Also, the saved project files can be distributed.

5.1.2.2 Data management

Removed the selection of variables when data loading

To simplify the data loading all the variables in the CSV files are loaded into the system. After data loading, the system provides a process to select which

variables are to be used in the analysis and the pattern discovery processes. The system now loads multiple data files faster than before

Also, this adds flexibility because if the user wants to analyse new variables, he does not have to reload all the CSV files as in the last prototype.

Temporal limits

In the last prototype, all the data was analysed, but sometimes not all the data has to be analysed. In this version I have implemented the “Time frame before the SE” parameter that determines which is the first value to be analysed. This is also used in the pattern matching process.

Data granularity and time point period parameters

In the last version I have used the name “Data granularity” to describe the time period used for smoothing the continuous values. That was unclear for the users, so I have changed the name of that parameter to “Time point period”.

In the last prototype, the time point period (time granularity) options were only some values from 0:15h to 8h. In this prototype I have added more flexibility, letting the user choose between different time scales - from milliseconds to years - and a wide range of values. This is useful for pattern matching.

Positive and negative special events

The special events are of two types, namely: positive or negative. Negative means that a special event did not happen. This is useful for the matching pattern process. A perfect pattern will match only the data segments with a positive special event and will not match any of the data segments with a negative special event.

Selection of the special event time from the time variable

A combo box with all the time stamps allows to the user to determine the time of the special event, this functionality helps the user to determine the time of the special event faster and with precision.

Other functionalities

- The user can remove the data files from the current project.
- Null time values accepted
- The file can have a special event variable
- If the file has more than one possible time variable, then the system asks the user which one to use as the time variable for the analysis

5.1.2.3 Variables

All the variables are displayed in a panel as can be seen in Figure 5-4. There the user can select which variables are to be analysed. Also, the user can define the ranges for the variables

5.1.2.4 Data display

Only the data within the analysis time is showed

It is not necessary to show data before the analysis time frame neither after the special event.

Select which data segment display

As the system allows the loading of more than one data file, the user must be able to select which data file to display.

Draw a marker for the special event

To mark where a special event is, a vertical line is drawn in the graph. At the top of this marker, information is displayed which indicates whether this is a positive or negative special event.

Discrete ranges colours

The colours of the discrete ranges are painted on the graph. As seen in Figure 5-6 , the colours are painted in the area of every range. This allows the user to visually check if any value is outside the defined ranges.

In the discrete graph, the colours are painted in each time point as seen in Figure 5-7. This is useful to visually check the value changes.

To add the markers and the range colours to the JfreeChart graphs were quite easy.

5.1.2.5 Data Analysis

Analysis module removed

It is now necessary to implement the pattern discovery process. This means that a huge amount of work has to be done in a short time period. To simplify the code, I decided not to include the data analysis module.

5.1.2.6 Pattern discovery

The patterns described in chapter 3.6 are single time points, single time point composite, temporal, temporal composite, combinatorial and combinatorial M-out-of-N patterns. This is an abstract description of the different patterns needed for this domain, but that classification can be simplified as:

- PatternNode, which is the single time point and a temporal pattern
- AndPattern, which is the single time point composite and a temporal composite pattern.
- CombinatoryPattern, which is the combinatorial and a combinatorial M-out-of-N pattern.

The PatternNode is the simplest of the above patterns.

UI panel to introduce patterns

Figure 5-8 shows the panel to add new patterns;

Figure 5-9 and Figure 5-10 show the dialogs to edit these patterns; and Figure 5-11 show an example of a pattern matching report.

5.1.3 UI

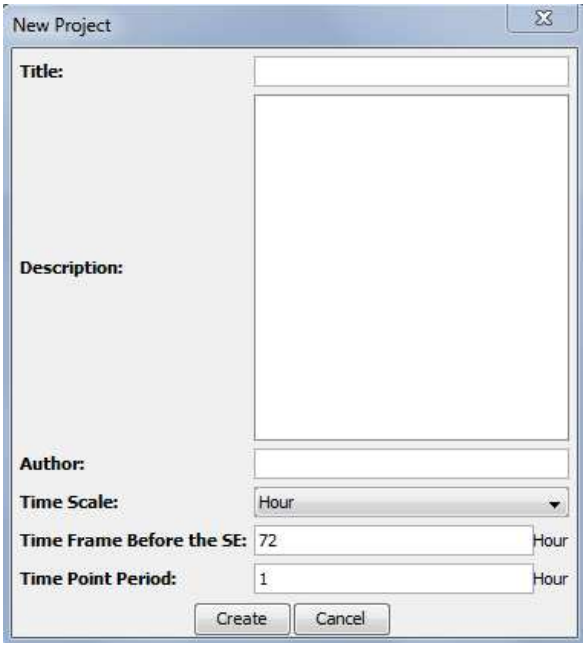

Figure 5-2 – New Project

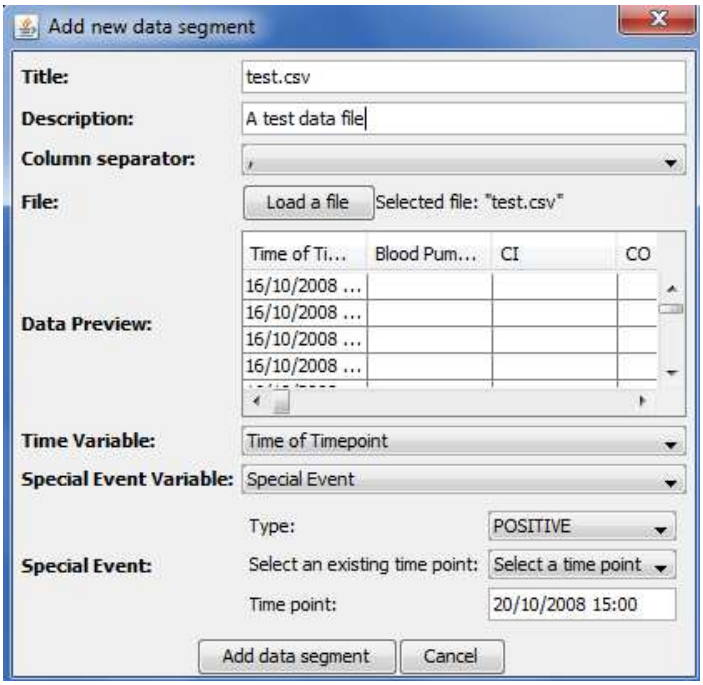

Figure 5-3 – Add data segment dialog

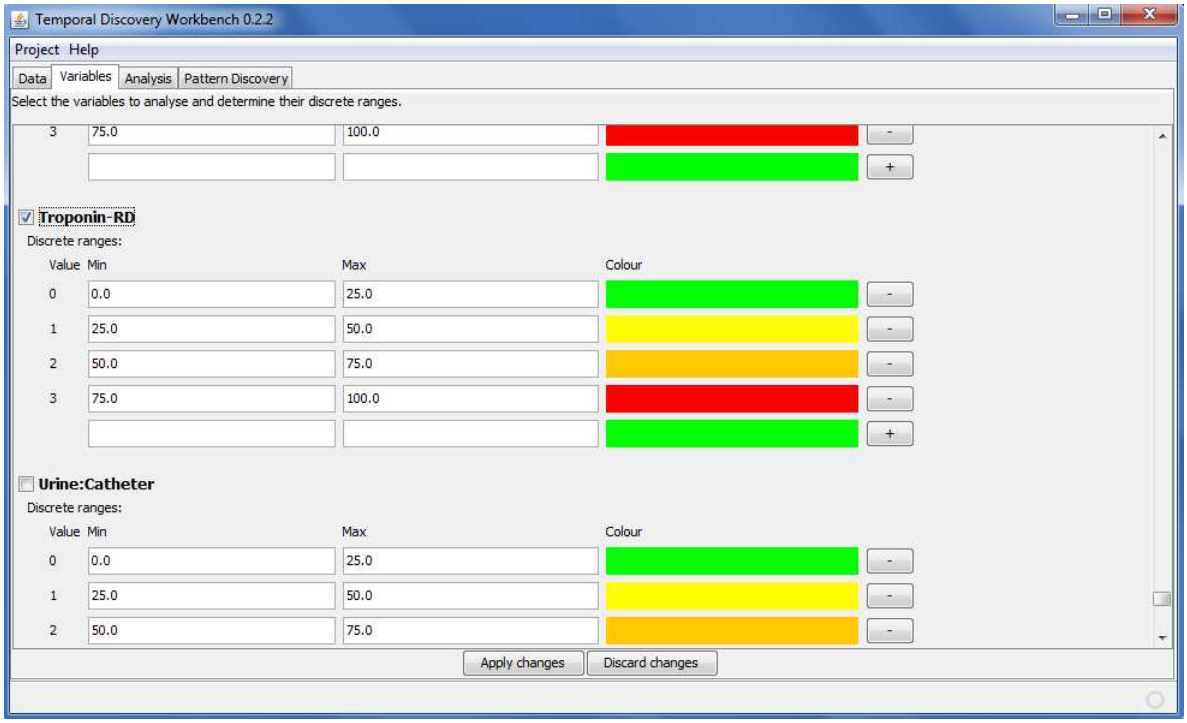

Figure 5-4 – Variables panel

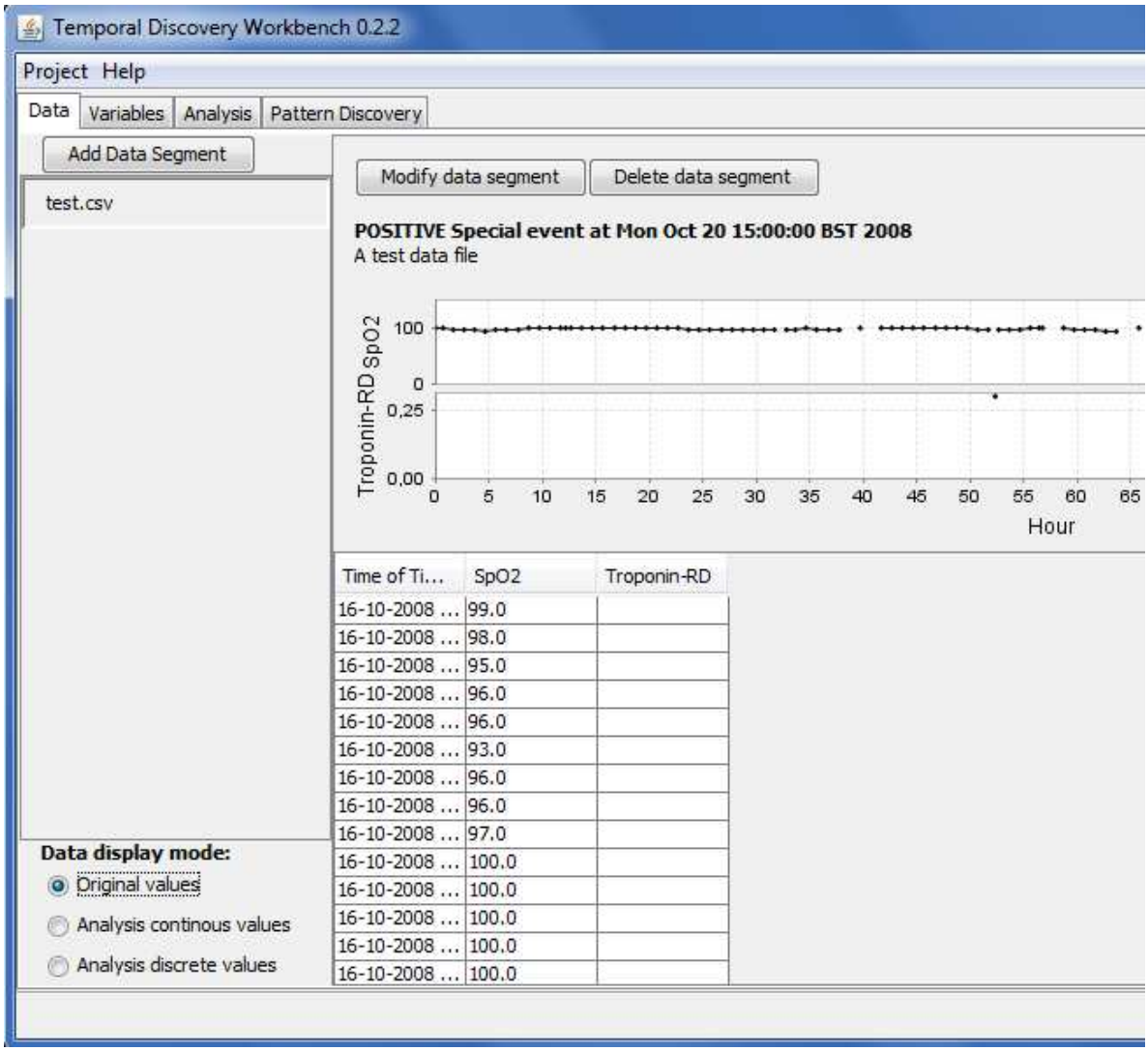

Figure 5-5 – Data panel

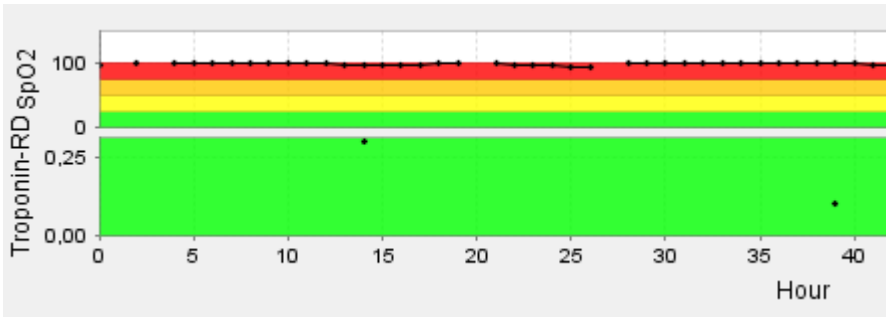

Figure 5-6 – Continuous graph

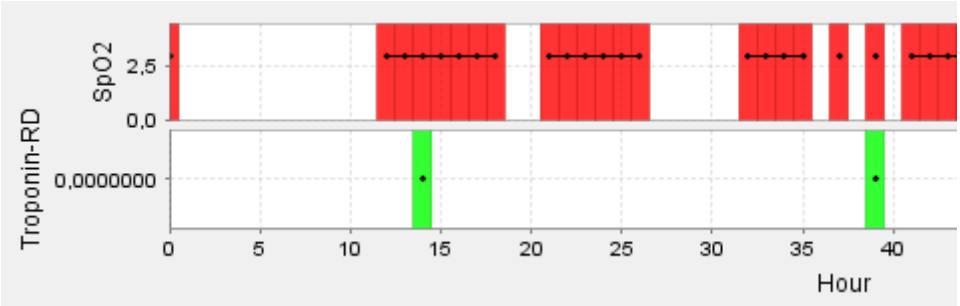

Figure 5-7 – Discrete graph

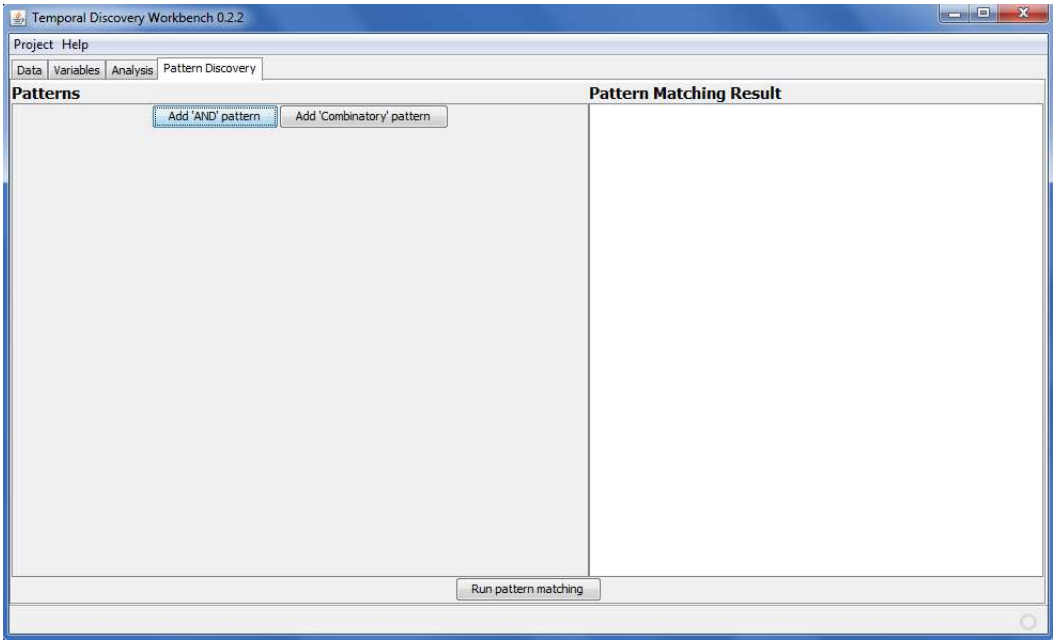

Figure 5-8 – Pattern discovery panel

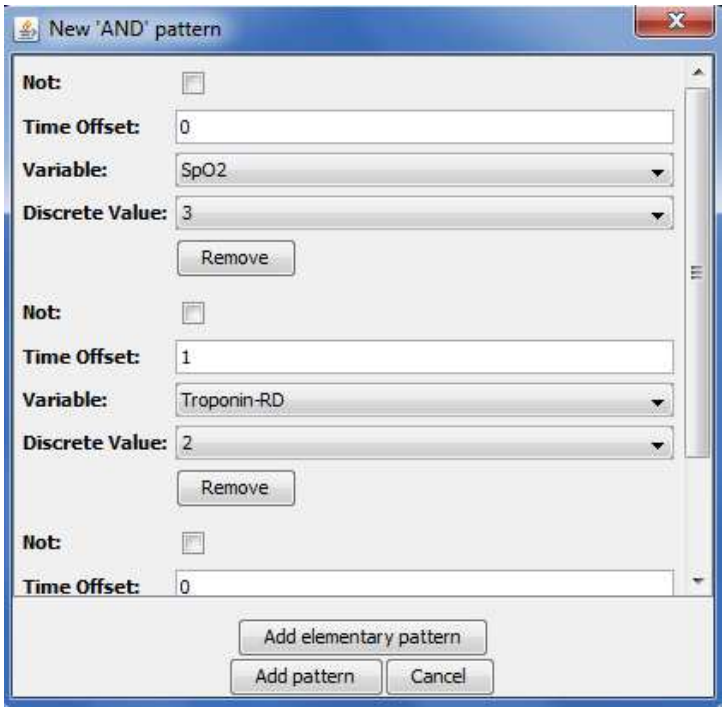

Figure 5-9 – And pattern edit dialog

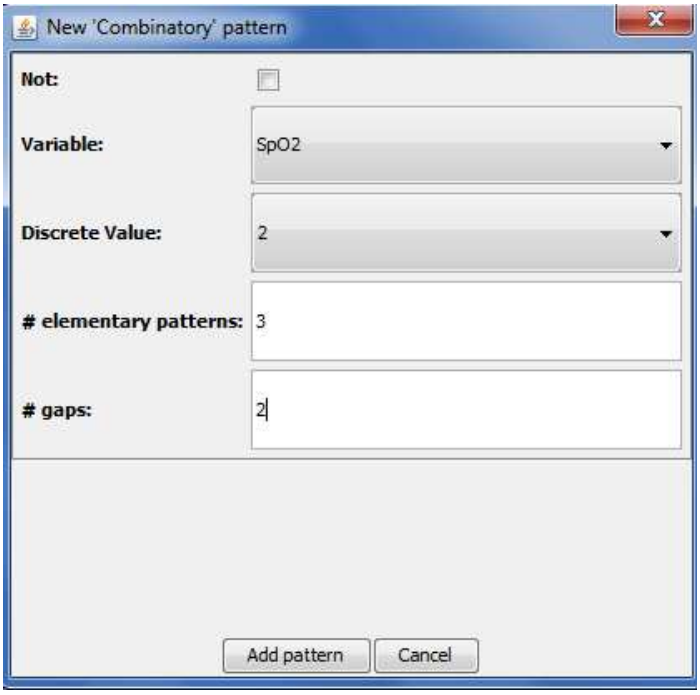

Figure 5-10 – Combinatory pattern dialog

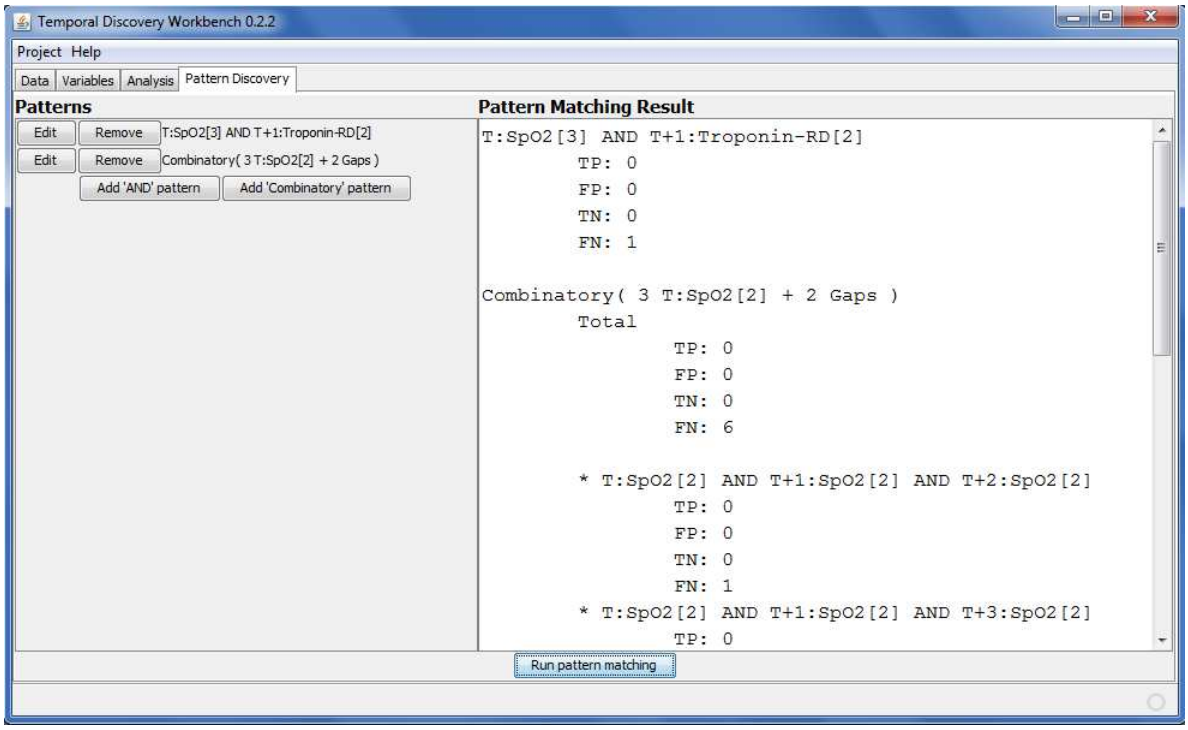

Figure 5-11 – pattern matching report

5.1.4 UML

5.1.4.1 Model UML

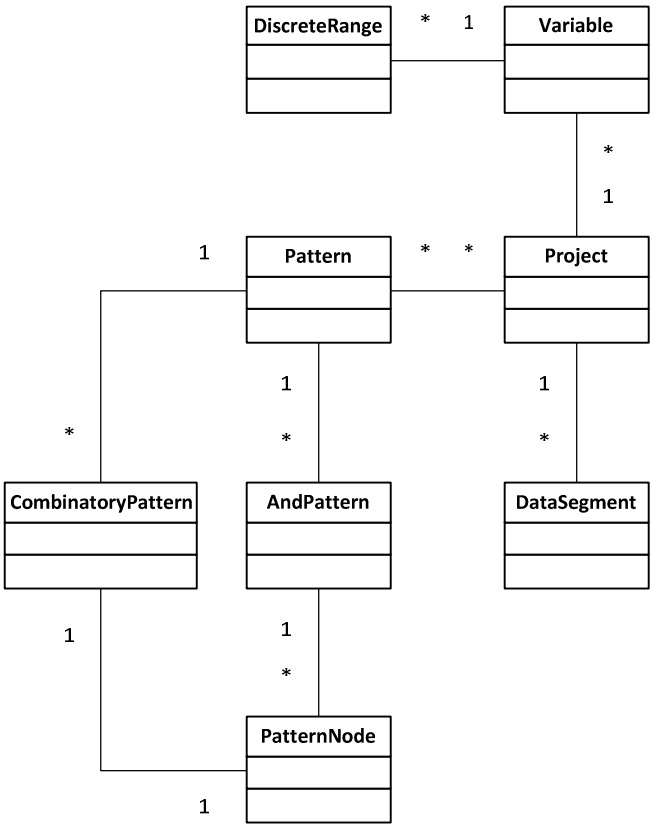

Figure 5-12 – Model UML

5.2 Users evaluation

User evaluation realized on 16 December 2011.

5.2.1 Proposed changes

This is the list of changes proposed by Derek Sleeman and Wamberto Vasconcelos after evaluate the system on 16 December 2011. A detailed discussion of the actual changes applied can be found in the functionalities section 6.1.2 of the next version of the program.

5.2.1.1 Data management

- Allow multiple special events per file.
- Step by step wizard to load files and select variables to be used in the analyses & pattern generation

5.2.1.2 Variables

- Edit variables in a dialog, not in a tab panel.
- Make the Variables edit system easier to use.
- Enhance discrete ranges labelling system.
- Grey colour for new discrete ranges.

5.2.1.3 Pattern discovery

- Pattern matching thresholds
- More sophisticated patterns

6 Final version (TDWB 1.0)

After developing several prototypes, this is the final, and most complete, version of the system. There are some major changes to the system like a Model-View-Controller structural pattern, functionality to load multiple Special Events from a Data file. Additionally there is the abstraction of the graph library, of the analysis modules and of the patterns. A more flexible pattern creation and a more sophisticated pattern evaluation are also included.

Also, a stylish GUI, a wizard for data loading; and confirmation, warning and error dialogs are included in this version.

The analysis module removed from the last prototype is included in this version and a new analysis module has been created; two analysis modules are also included.

6.1 Design

6.1.1 Use workflow

This version implements all the functionalities, so the use workflow includes the analysis and the pattern discovery processes, as seen in Figure 6-1 – Use workflow.

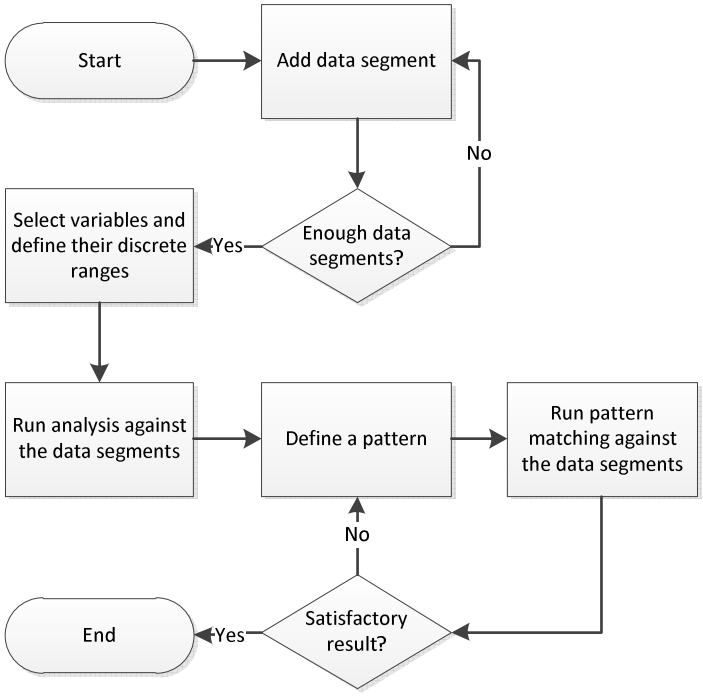

Figure 6-1 – Use workflow

6.1.2 Functionalities list

This is a list of the new and the modified functionalities.

6.1.2.1 Data management

Title for Data Files

It is not necessary to define manually a title or a name for the Data Files. The Data File name is used as the title. This change simplifies and speeds up the Data Files loading process.

Multiple Data Segments per Data File

To make the system more flexible, it is now possible to load multiple Special Events from a Data File. Each Special Event determines the end of a Data Segment and each Data Segment is as long as determined by the parameter "**Analysis time frame**". That is defined when creating a new project. After the project creation, this parameter can be modified in the project properties dialog that can be accessed from "Project -> Project properties" in the menu.

As can be seen in Figure 6-2, two Special Events are loaded from a Data File. The analysis time frame parameter is 72h. Then, two 72h Data Segments are used for the data analysis and the pattern discovery process.

Currently, the Data Segments can overlap. Functionality to allow the user to determine if the Data Segments overlap or do not overlap could be developed as part of further work. Currently, if the user wants non-overlapping Data Segments, the user has to split the Data File into multiple Data Files, and load them separately.

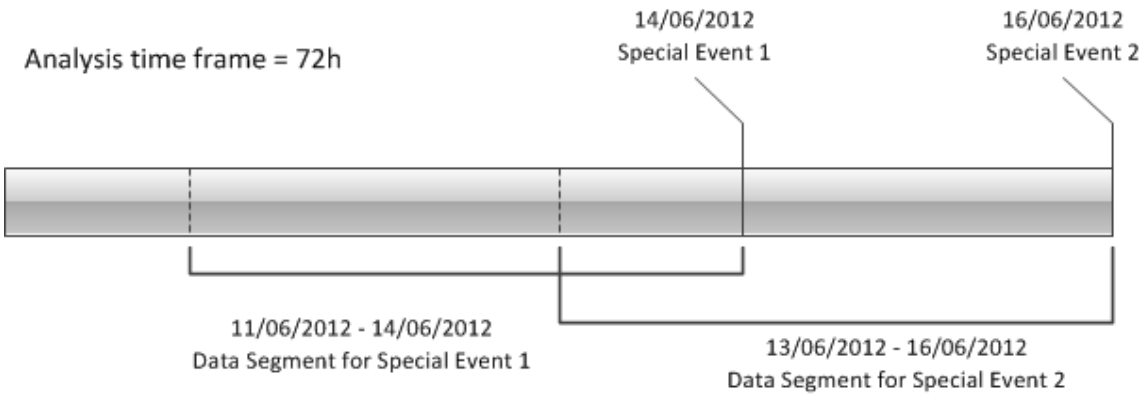

Figure 6-2 – Data segment overlapping

Editing Special Events

To simplify the system, the Special Events cannot be edited anymore from the program. The type and the time are loaded from the Data File and are permanent in the system. To modify the Special Events, the Data File must be removed from the program, the original CSV file edited, and then re-loaded into the program.

A functionality to edit the Special Events could be part of further work.

Show an error if the Time variable or the Special Event variable are not detected

Those variables are necessary. So if they are not in the Data File then the system does not load the file

Check time format errors

A data record is ignored if the format of its time variable is not valid.

Show a warning message if empty variable names

The system show a warning message when it detects empty variable names in a Data File. If the user wants to load that Data File, the variables with empty names will not be loaded.

Show an error message if duplicated variable names

The variable names must be unique. If the system detects duplicate variable names, then it displays an error and the Data File is not loaded. The variables names cannot be duplicated because then the system cannot differentiate between them.

Step by step wizard for loading files and selecting variables

To add agility and to help the user to use the program, the system provides a step by step wizard.

After adding a Data File into the system, the system asks the user what to do next: add more Data Files, set the variables perform the analysis or close the wizard.

6.1.2.2 Variables

Edit variables in a Dialog, not in the tab panel

It is not intuitive for the user to edit the variables in the main tab panel. With the tab panel, after editing the variables and pressing the “**Apply**” button, the system did not confirm the changes to the user.

The solution for this problem is to edit the variables in a dialog. When the user presses the “**Apply**” button, the dialog disappears and then the user, intuitively, realises that the changes are confirmed.

New mechanism to edit variables

When the system is managing lots of variables, is not convenient for the user to see all of them (the selected and the unselected variables) in the same scroll panel.

A better way is to only display in detail the selected variables. To achieve this, the system uses a divided panel. The left panel shows a compact list of all the variables where each variable has a check box to select it. The selected variables are shown in detail in the right panel where the user can edit their discrete ranges. The new variables dialog can be seen in Figure 6-7.

Discrete ranges labelling system

A unique ID is necessary to preserve consistency between the discrete ranges and the patterns. The unique ID is an alphanumeric label that the user must define. In addition, a label is a good mnemonic. For example, for the low values the label could be “L” and for very low values the label could be “VL”. Figure 6-7 shows the UI to define discrete range labels.

Descriptions in discrete ranges

As aide memoire for the user, a description field is included for each discrete range, as can be seen in Figure 6-7. This field is not mandatory.

6.1.2.3 Data display

Default grey colour for new discrete ranges

In the last prototype, the default colour for new discrete ranges was the green. But the colour green was also one of the colours used for the default discrete ranges. And that is confusing. So it has been decided to use a grey colour as the default colour for new discrete ranges.

Labels in Graphs

As the identifying system for the discrete ranges now uses alphanumeric labels, the discrete graph axis must now show these range labels instead of the integer values.

Size of the Graphs

In the last version the graph size was fixed to the dimensions of its containing panel. That was a problem when large Data Files were loaded into the system, because the graphs were compressed to fit the panel. Now, the graph’s dimensions are not fixed to the dimensions of its containing panel and grow

according to the amount of loaded data and the selected variables. Scroll bars are provided to allow visualization of the entire graph.

The JfreeChart graphs are designed to fill its containing swing component and it is very difficult to customize their dimensions. Now, when large files are loaded, the separation between time stamps is- more or less- the same than when loading small files. But when large files are loaded, the graph labels deform. I have been looking for a straight forward solution but I haven't found any one. Maybe should be a good idea to use other chart library like JChart2D.

6.1.2.4 Data analysis

In this version, data analysis modules are included. They are two modules, the "Number of Elementary Patterns" module and the "Value Changes" module.

Number of Elementary Patterns

This module reports the frequency of all the elementary patterns, splitting the result for the positive data segments and the negative data segments.

Value Changes

This module reports all the variables value changes, splitting the result for the positive data segments and the negative data segments.

6.1.2.5 Pattern discovery

The pattern's name AndPattern, used in the last prototype, had not been understood by the users, so I decided to change it to CompositePattern.

Also, I have renamed the PatternNode to ElementaryPattern. Because the name 'PatternNode' now refers to the abstract class which all the patterns implements.

A report to compare the patterns and their results

After a pattern matching analysis, a dialog with a text report is shown with the number of the Data Segment matches for each pattern. All the patterns are checked against all the positive and negative data segments. The program shows a report similar to Figure 6-16.

When a pattern is matched against a data segment, there are four possible results:

- TP: True Positive, if the pattern matches a data segment with a positive special event.
- TN: True Negative, if the pattern doesn’t match a data segment with a negative special event.
- FP: False Positive, if the pattern doesn’t match a data segment with a positive special event.
- FN: False Negative, if the pattern matches a data segment with a negative special event.

At the end of the report there is an extra row which provides the combined result of all the patterns.

A perfect pattern only matches data segments with a positive special event, and none of the data segments with a negative special event.

Discrete labels UI input

In the last prototype, to define a discrete range in a pattern, a combo box was used. If the user removes or changes the label of a discrete range used in a pattern, then there is an inconsistency. Because the combo box, when it is initialized again is not going to provide the option for the old value.

The implemented solution is to use an input text field instead of a combo box to define the discrete ranges. To show the user the different options, an

informative text is displayed near the input text field. If the user introduces a non-existing label for a variable, a warning dialog is shown after pressing the add button.

Figure 6-12 shows the new UI to introduce elementary patterns.

Check patterns with some invalid variable name or some invalid discrete labels

If there is a pattern with an invalid variable or an invalid discrete range label, the system displays a warning panel in the pattern discovery tab. Also, a warning dialog is shown if the user presses the "**Run pattern matching**" button in the pattern discovery tab.

Pattern discovery thresholds

In the last prototype, the pattern matching algorithm stopped when found a single match in a Data Segment and reported a match. But a more flexible approach is to let the user define a threshold of matches per Data Segment to report a match.

The patterns are tested against all the possible time points of the data segments. For example, for the data set:

```
A:  N, N, N, L, L, L, L, N, N, N
B:  N, N, H, H, H, H, N, N, N, N
```

If the pattern to be matched is **A[L]**, then this results in **4** matches out of **10** possible matches:

```
A:  N, N, N, L, L, L, L, N, N, N
B:  N, N, H, H, H, H, N, N, N, N
```

We have 4/10·100=**40%** matches against this data segment.

If the pattern to match is **Composite(T+0: (A[L]), T+1: (B[H]))**, the result is **2** matches in **9** possible matches:

```
A:  N, N, N, L, L, L, L, N, N, N
B:  N, N, H, H, H, H, N, N, N, N
```

We have 2/9·100=**22.22%** matches against this data segment. Is 9 possible matches because the length of the data is 10, and the length of the pattern is 2, so 10-2+1 = 9 possible matches.

To determine if the pattern matches against the data segment, the user determines a threshold for the number of matches or for the percentage of matches in a data segment.

For the patterns P1, P2 and P3 if the user determines the thresholds as in Table 6-1.

Thresholds	%	#
P1	40.00	-
P2	--.--	1
P3	0.13	23

Table 6-1 Pattern thresholds example

The pattern matching algorithm will report a positive data segment match it:

- Finds 40% or more matches for the pattern P1; or
- Finds 1 or more matches for the pattern P2; or
- Finds 0.13% or more, or a number of 23 or more matches for the pattern P3.

These thresholds can be defined in the “**Pattern Discovery**” panel, like is displayed in Figure 6-15.

Recursive patterns

Now, the composite patterns and the combinatory patterns can be composed of not only elementary patterns, but also of all kinds of patterns. This adds more complexity to the possible patterns. That requires a recursive system to build patterns. A pattern with the shape of a tree is an abstract representation of the recursive pattern creation, where the elementary patterns are leafs and the composite and combinatory patterns are nodes of this tree, as seen in Figure 6-3.

This provides the user with more possibilities for design more complex patterns.

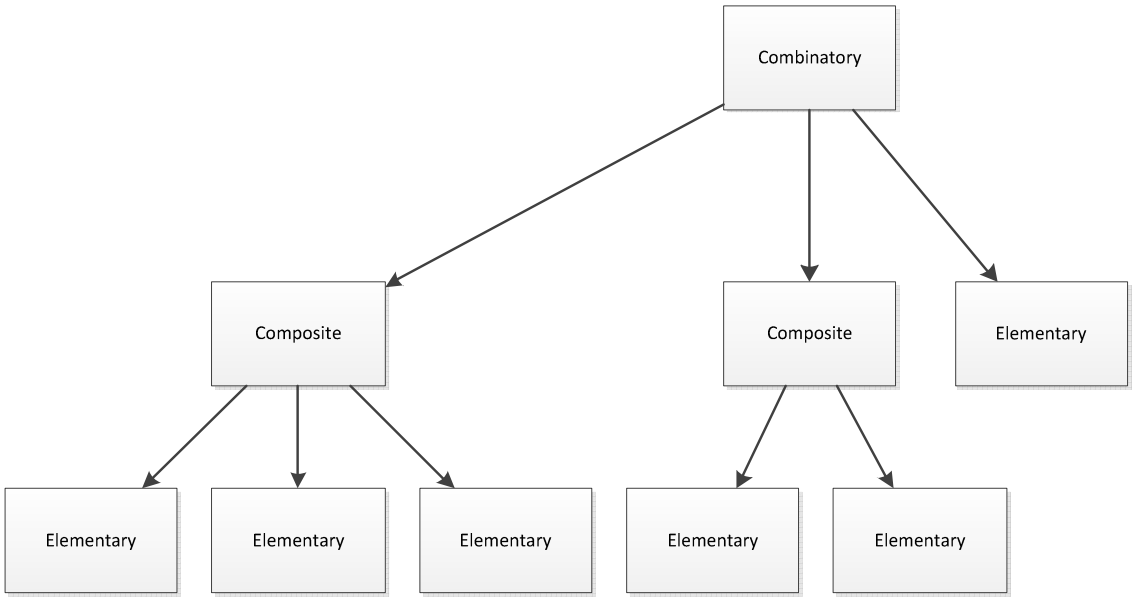

Figure 6-3 – Complex pattern

Select/deselect patterns to match

In the “pattern discovery” tab, the user can select (or deselect) the patterns which are matched against the data segments.

Export/Import the patterns into/from files

To reuse the patterns in various projects, the user can export the pattern from a project into a system file and then import them into other projects subsequently if needed.

6.1.2.6 User manual

The user manual can be found in the menu by clicking “Help -> User manual”.

6.1.3 UI

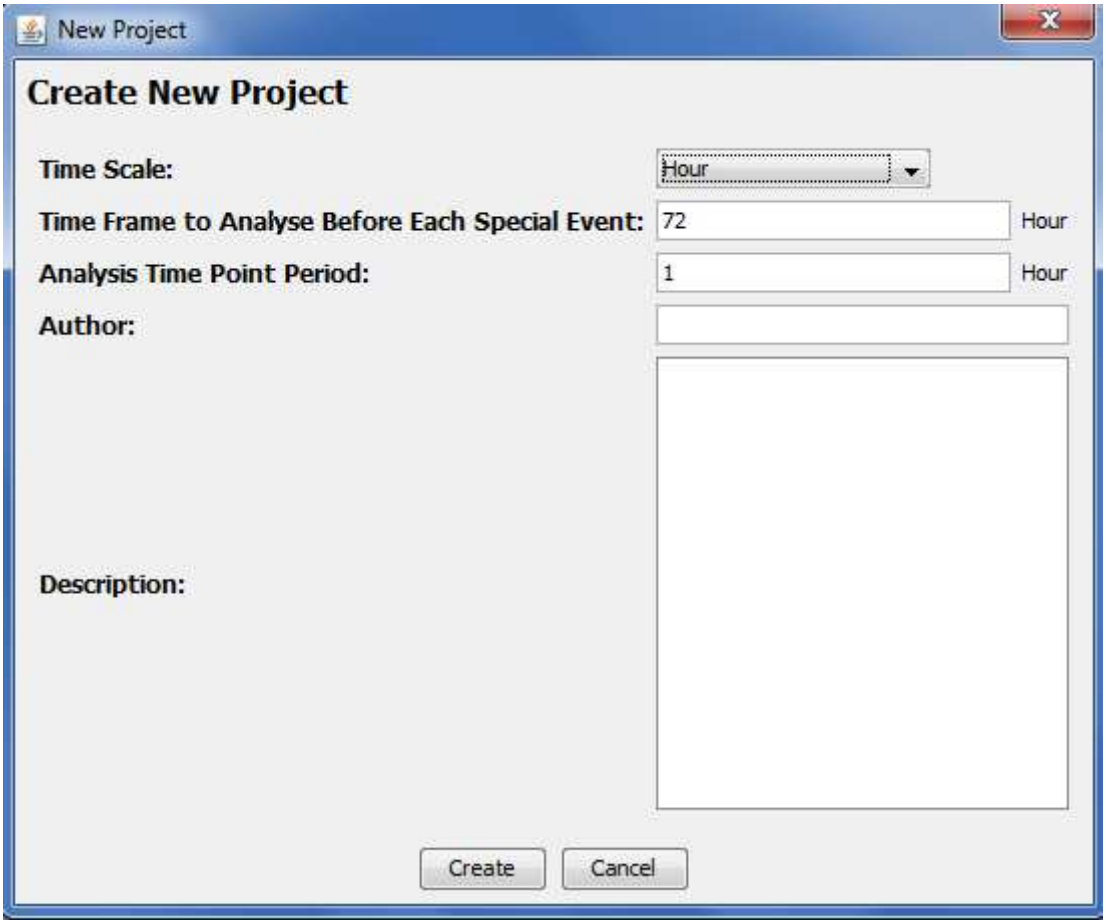

Figure 6-4 – Create a new Project/Project properties dialog

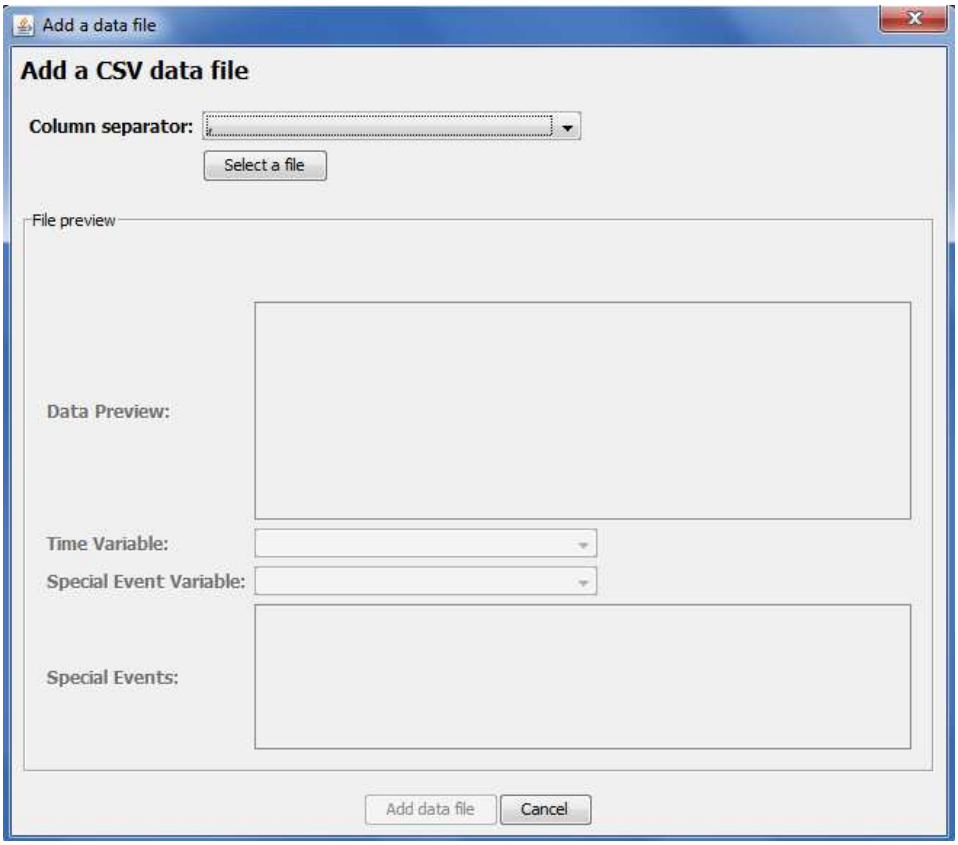

Figure 6-5 – Add a data file dialog

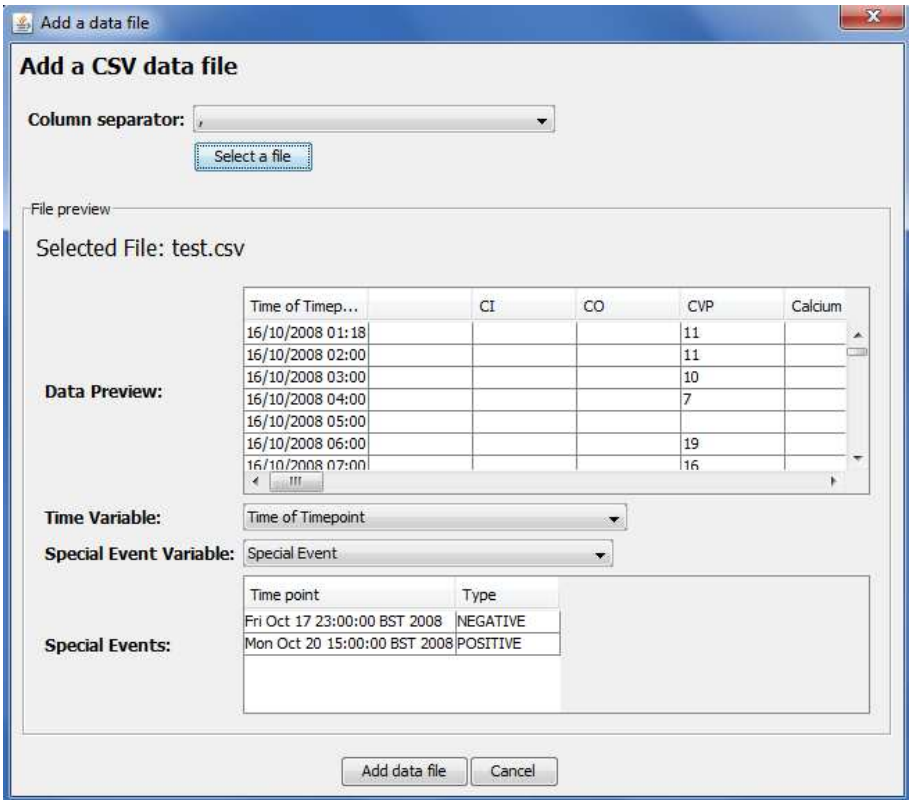

Figure 6-6 – Add a data file dialog with file preview

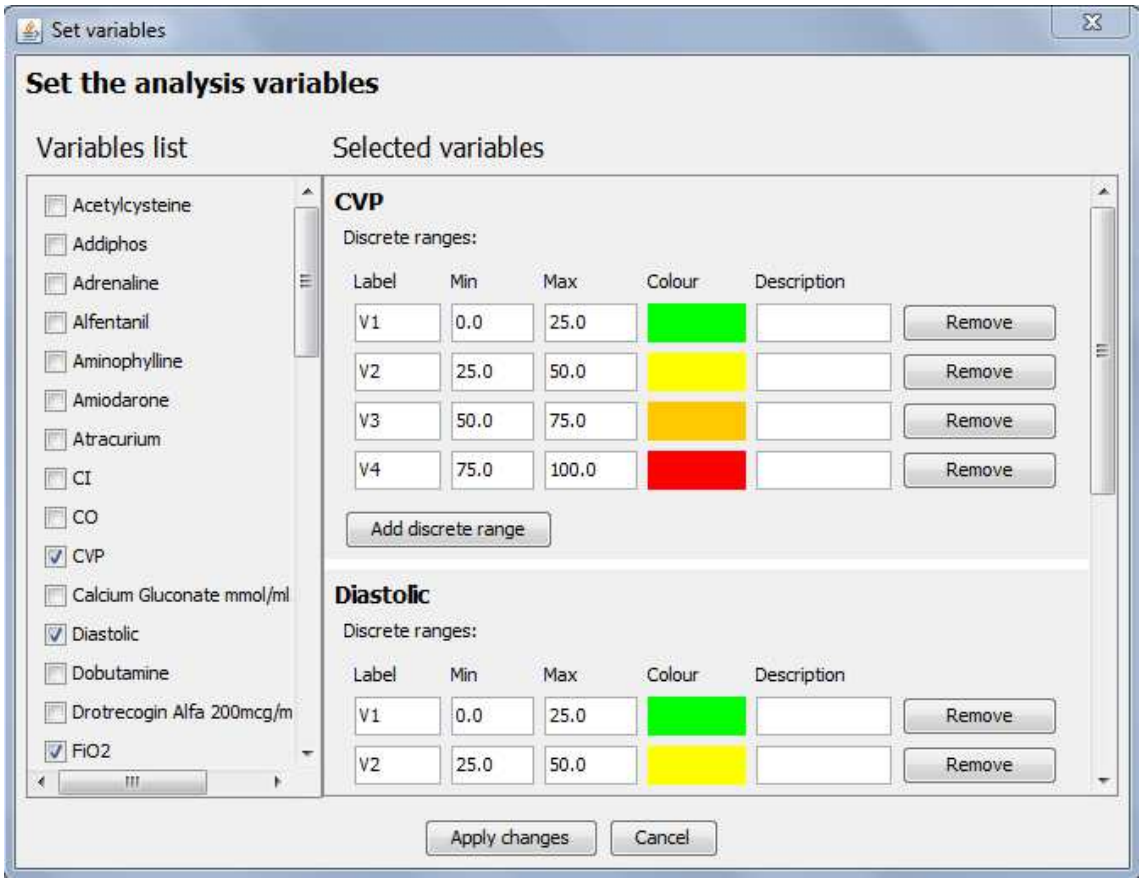

Figure 6-7 – Set variables dialog

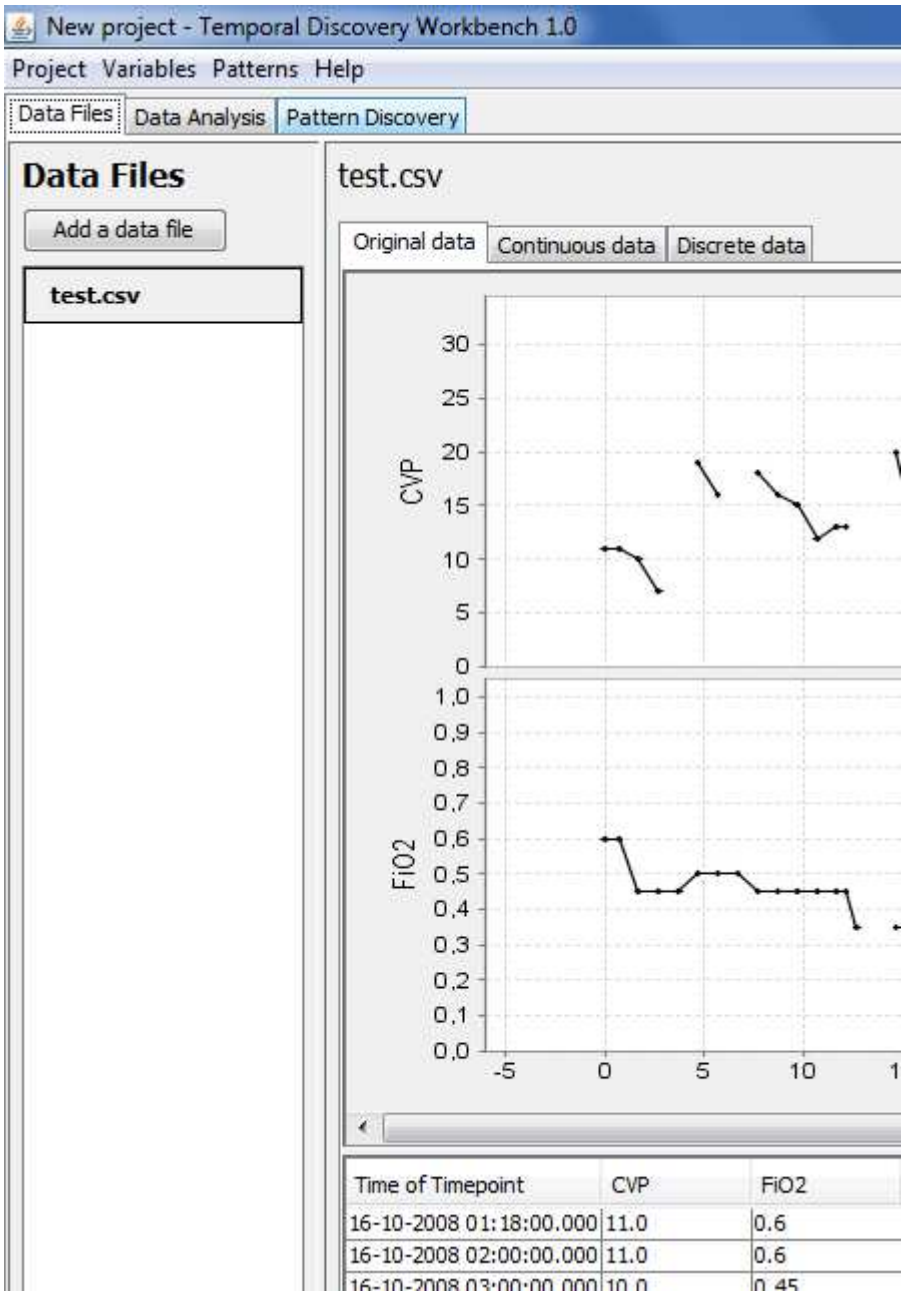

Figure 6-8 – Data files panel

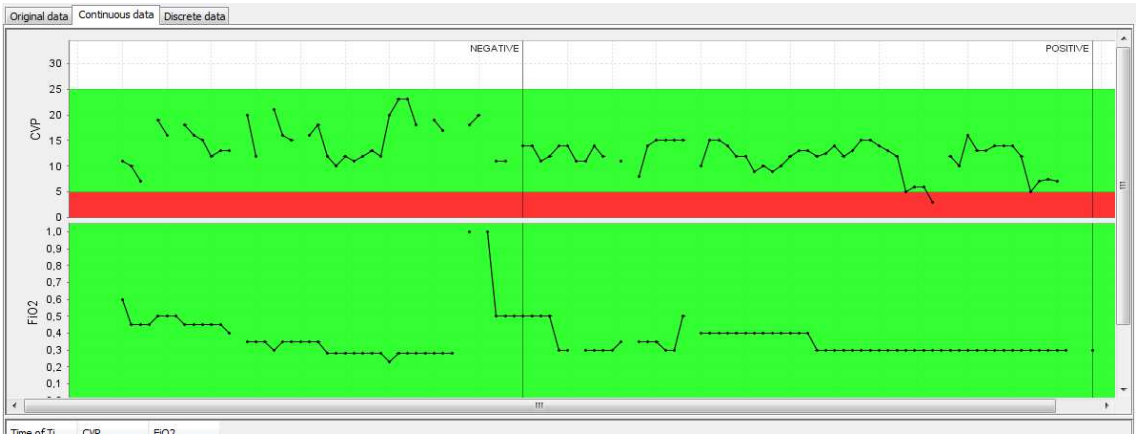

Figure 6-9 – Continuous data graph

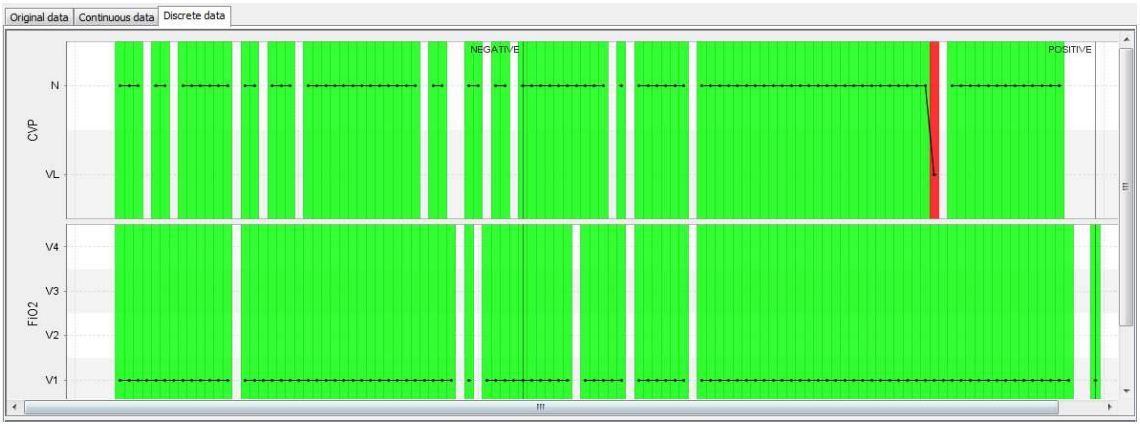

Figure 6-10 – Discrete data graph

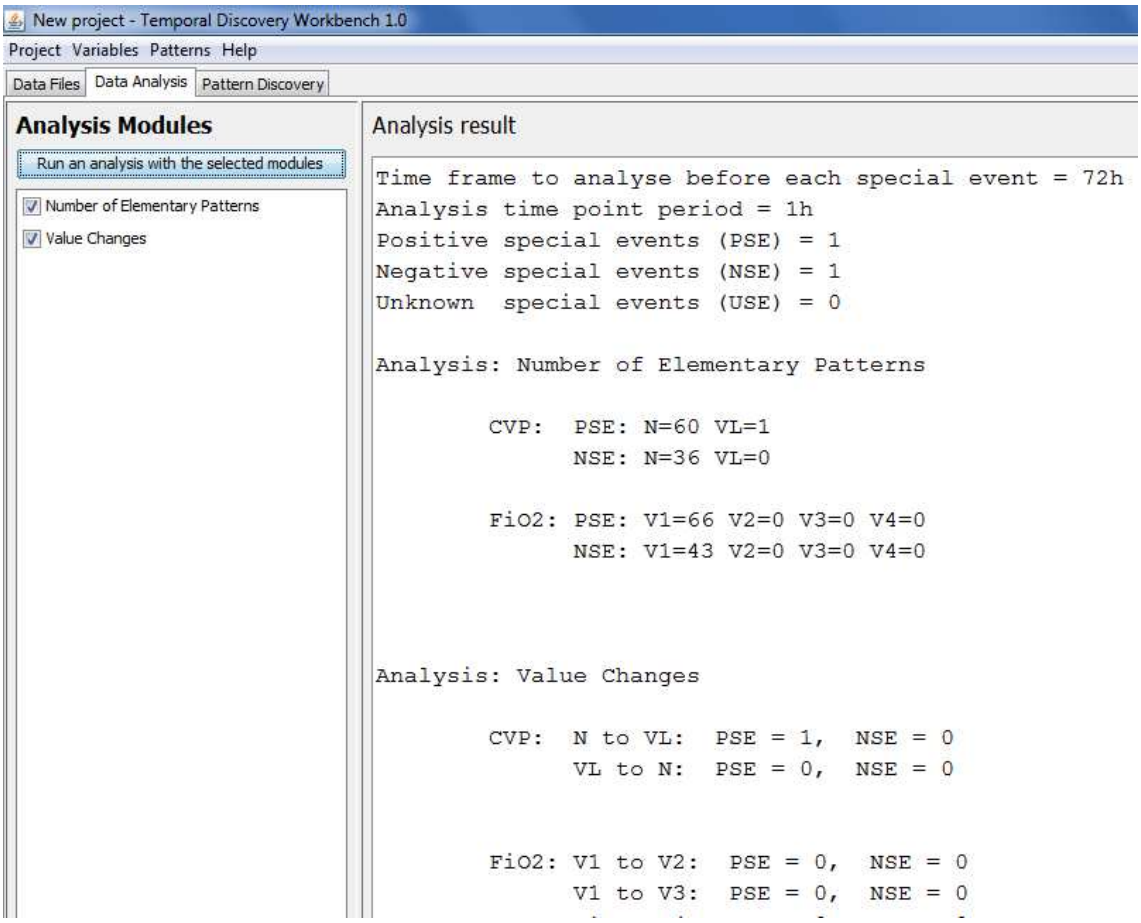

Figure 6-11 – Analysis panel

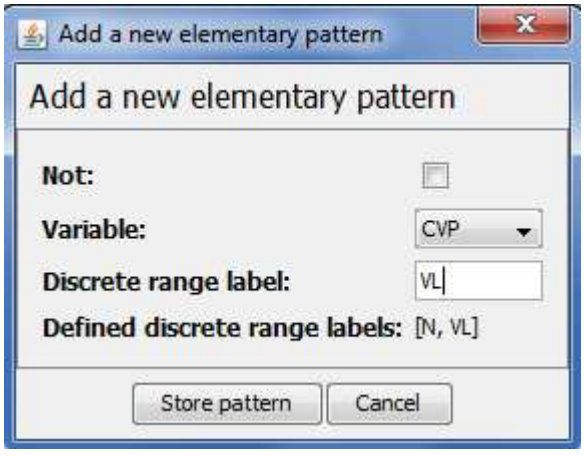

Figure 6-12 – New elementary pattern dialog

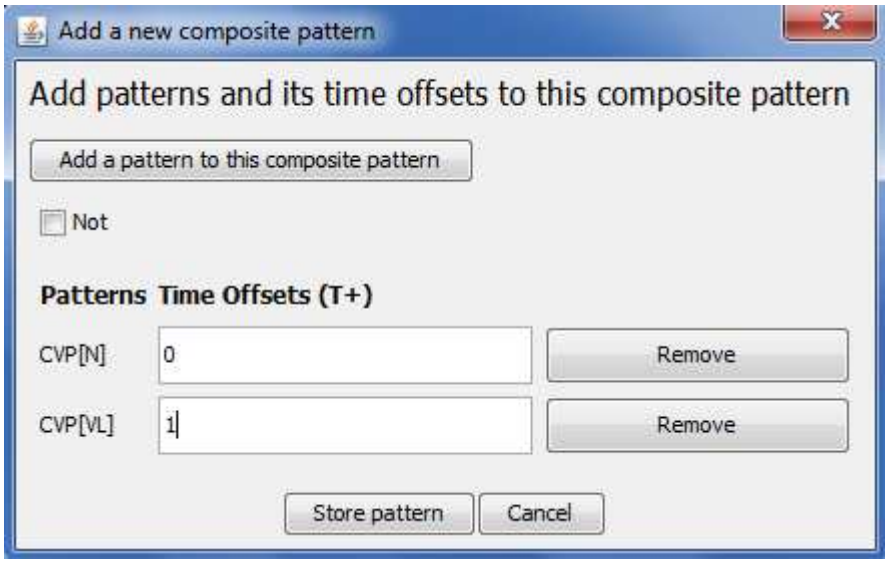

Figure 6-13 – Add a new composite pattern dialog

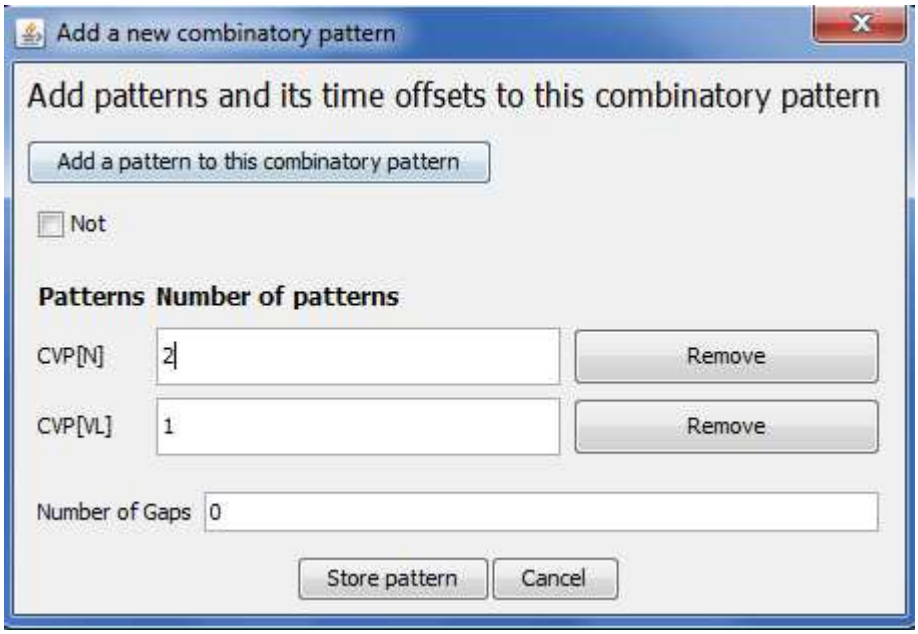

Figure 6-14 – Add a new combinatory pattern dialog

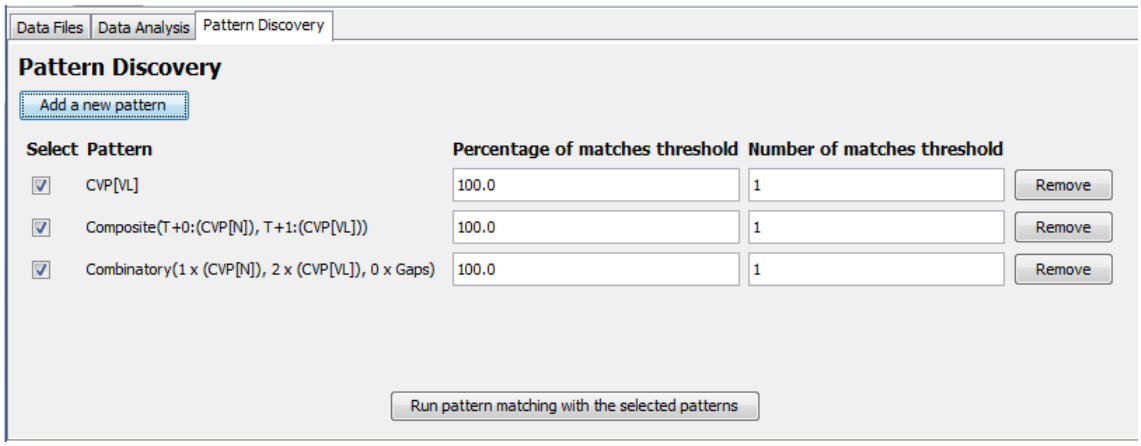

Figure 6-15 – Pattern discovery panel

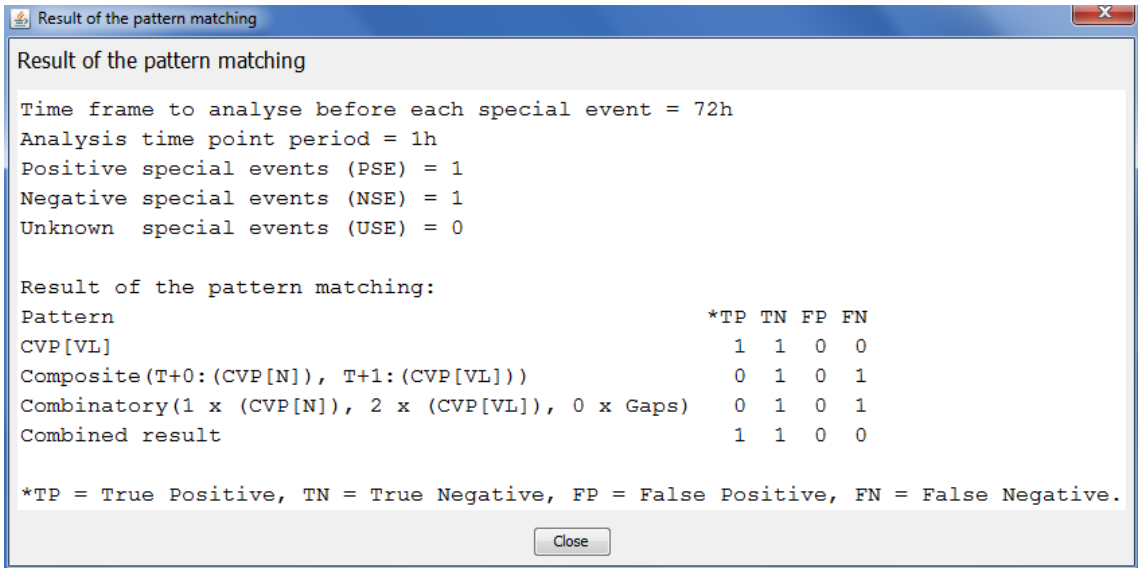

Figure 6-16 – Pattern matching report

6.1.4 Model-view-controller

The final version is structured with the classic architectural pattern model-view-controller, or MVC [EG 94].

The model layer contains all the classes that represent the information and the methods to transform it.

The view layer has the classes that draw the GUI in the system screen showing the state of the information and gives mechanisms to allow the user for interact with the system.

The controller is a layer between the view and the model. The controller handles the user events, modifies the model information accordingly to the user's request and then updates the view layer with results. In Figure 6-17 – MVC request process, this process can be seen graphically.

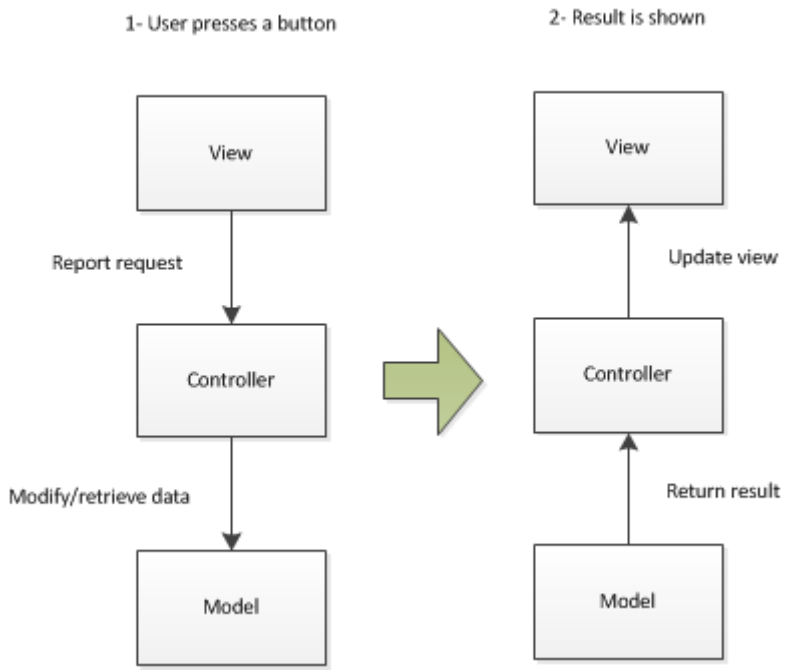

Figure 6-17 – MVC request process

The main purpose of the controller layer is to abstract the model from the view, then it is easier for a programmer to modify the model without modify the view, or to modify the view without modify the model [MVC 1] [MVC 2].

6.1.5 UML

6.1.5.1 Model layer

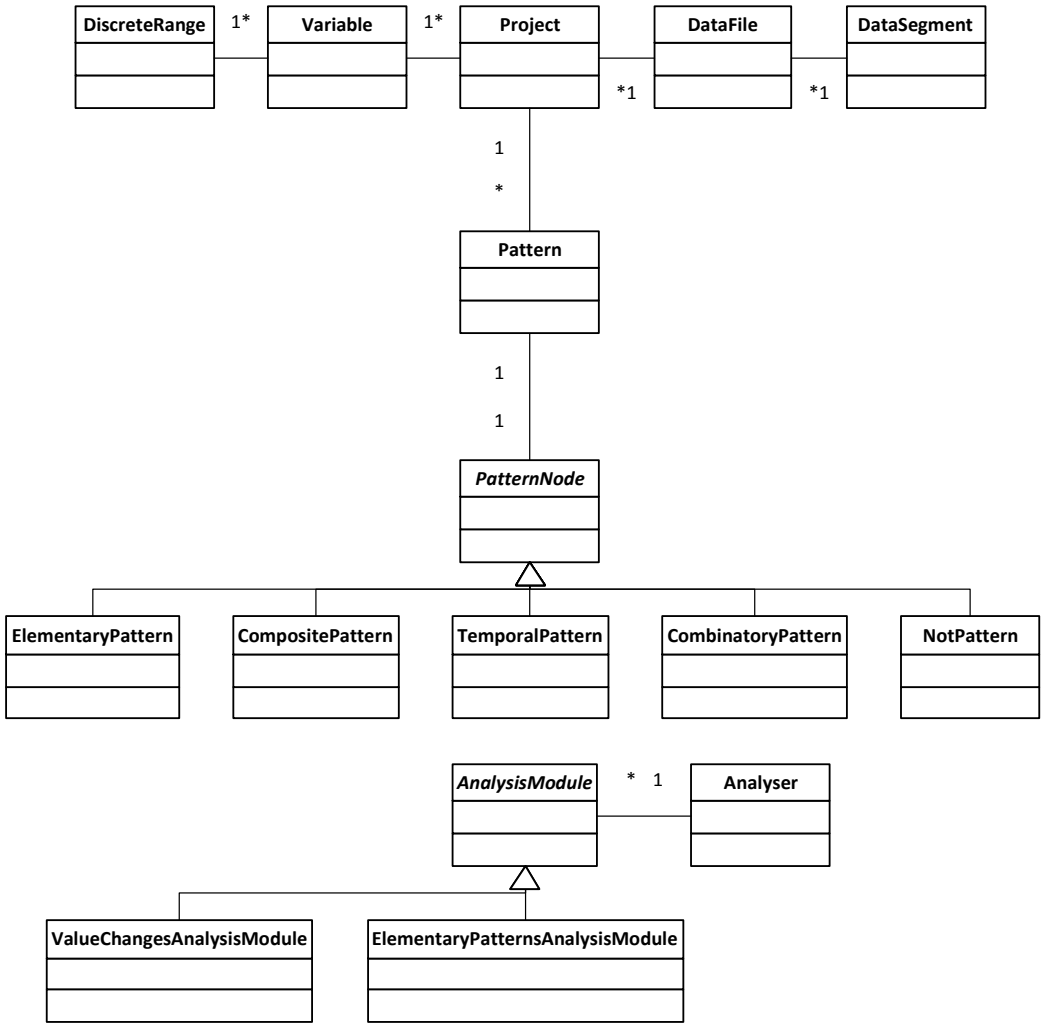

Figure 6-18 – Model UML

6.1.5.2 Controller layer

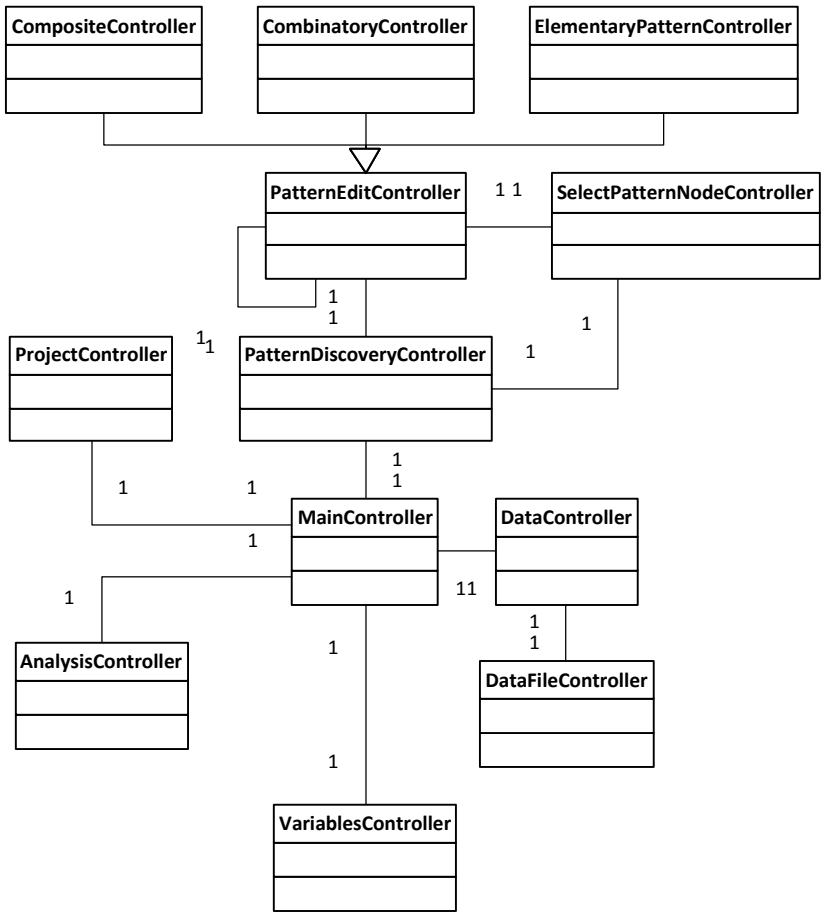

Figure 6-19 – Controller UML

6.2 Implementation

6.2.1 Generate the analysis data

In the class DataFile there is a method that transforms the raw variable data from the CSV file into analysable data, generating the smoothed continuous values and the discrete values.

6.2.2 Analysing the data

The Analyser.java file has a method that analyses the data and returns the result of the analysis report.

6.2.3 Generating composite patterns in a combinatory pattern

After a combinatory pattern is created, its composite patterns must be generated. In the file CombinatoryPattern.java are the methods that generate the composite patterns.

6.2.4 Pattern matching

The file Pattern.java contains the methods that determine if a pattern matches against a data segment.

The PatternNode.java flips the result of the match (of its subclasses) if the attribute isNot is set to true.

6.3 Optimizations

6.3.1 Pre-processing combinatory patterns

Originally, before match a combinatory pattern, all its composite patterns were generated. This adds more time to the pattern matching process which is also very time-consuming.

The solution is to generate the composite patterns when the combinatory pattern is first created. Then, the overall time is distributed becoming in shorter waiting times.

The trade-off is that more memory resources are needed but the actual computers have enough, also when the project is saved into the system disk, it needs more space.

6.4 Scalability

6.4.1 Changing the graph library

JFreeChart is very complete. But maybe, for a specific domain- or because a new version is available- it is necessary to change it.

I have implemented a class interface named ChartLibrary.java and used the JFreeChart library through this interface. Then, to change the graph library is easier. Figure 6-20 is a UML representing this scheme.

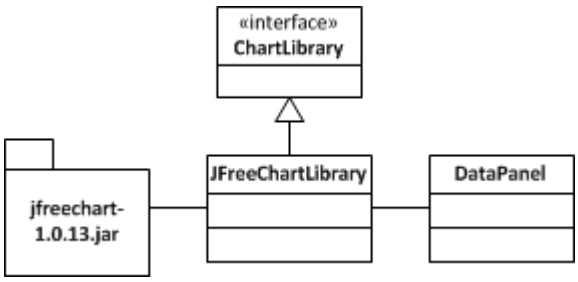

Figure 6-20 – Chart library abstraction

6.4.2 Adding new analysis modules

Currently, there are only two analysis modules, but future analysis undertaken by TDWB may require additional modules to be added. So I have implemented a system which allows one to easily add or remove analysis modules.

The analysis modules are implementing classes of a common interface AnalysisModule.java, and Analyser.java is a singleton class that manages the AnalysisModule's subclasses.

To add a new module one simply creates a new implementing class of AnalysisModule.java, implementing the methods getModuleName() and analyse(). Then, one modifies the constructor method of the Analyse.java class to add an instance of the new analysis module. An example of this architecture is shown in Figure 6-21.

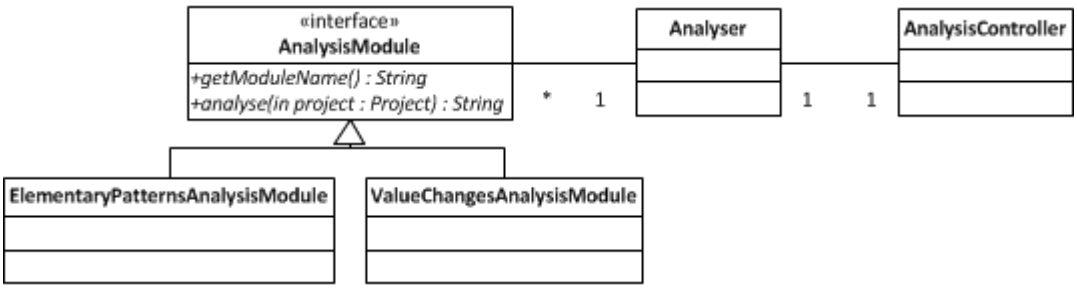

Figure 6-21 – Analysis modules abstraction

6.4.3 Adding new pattern types

For the myocardial damage study, combinatorial patterns are needed. For other studies, maybe it would be very useful to add other kinds of patterns that are not implemented currently. So a system to add new patterns easily is needed.

The solution is very similar to the solution for the analysis modules, namely we use an abstract class- PatternNode.java- and a manager class- Pattern.java- that will use the classes that implements PatternNode.java. However, here a different UI and a different controller are needed for the different patterns. This adds complexity to the solution. So I needed to implement the same solution for the controllers, with the abstract class PatternEditController.java.

A further component needed supports a dialog which selects the kind of pattern to be added, and this component needs to know about all the different kinds of patterns. For that purpose I have implemented the SelectPatternNodeTypeController.java and SelectPatternNodeTypeDialog.java classes.

An UML giving the architecture of this solution is shown in Figure 6-22.

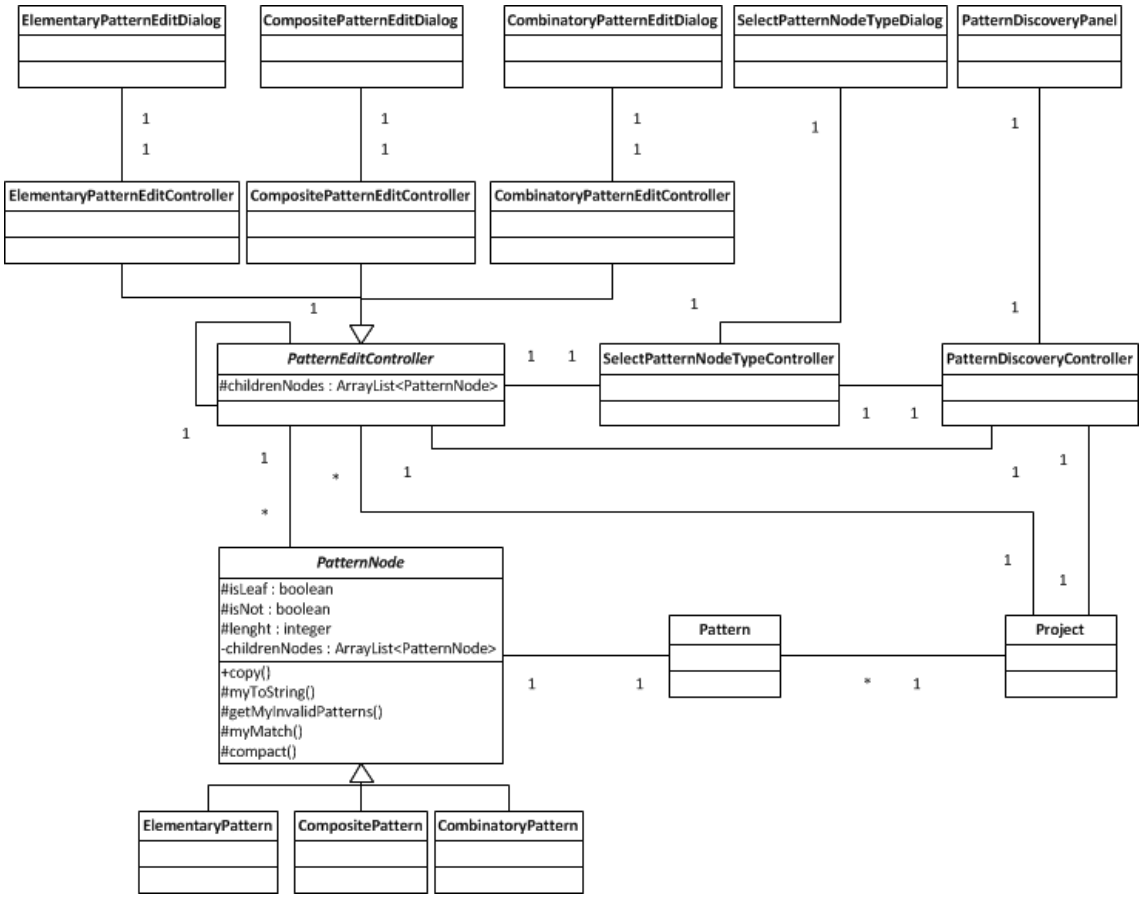

Figure 6-22 – Pattern types abstraction

To add a new pattern, simply write a new class that implements PatternNode.java, a class that implements PatternEditController.java, and its dialog.

To connect it all, edit the file PatternTypeEnum.java, add a new value to the enumeration and edit the function getController.

6.5 Users evaluation

Derek Sleeman and I (Daniel Blasco Calzada) interviewed Laura Moss on 16 January 2011. Laura Moss is co-author of [DS 11] and knows about the myocardial damage domain. The interview procedure was the following:

- Derek and I gave a copy of the user manual to Laura.
- Laura read the user manual.

- Derek and I provided CSV data files and project files to Laura.
- Laura used the program with the help of the manual.
- First, explored test projects.
- Secondly, tried to create a new project using the CSV files provided.
- When Laura didn’t understand some functionality Derek and I helped her.
- After using all the functionalities of the program, Laura gave us her feedback about the program.

In overall, Laura could use the program intuitively, except after determining the variables when she didn’t know what to do next, so she checked the user manual and realized what to do without our help.

The pattern creation is the functionality where Laura had more problems. In that point, Derek and I had to help her. For a lack of time, I couldn’t implement visual aids for the pattern creation process, neither edit functionality. That should be the first functionality to be improved in a further version

.

6.5.1 Users feedback

After the evaluation, this is the list of proposed changes by Laura:

.

6.5.1.1 Data management

- Allow data files with mixed time point periods for the same project.

6.5.1.2 Variables

- A button to select all the variables at the same time in the variables dialog

6.5.1.3 Data display

- Synchronize scroll bars for the graphs of the three tabs.
- Remove the lines between points in discrete graphs because with lines seems like continuous variables.

6.5.1.4 Data analysis

- Don’t sort the discrete ranges in the report

6.5.1.5 Pattern discovery

- Show more helpful information in the pattern creation dialog
- Allow patterns to be edited
- A functionality to export rules to standard file formats like CSV

7 Conclusion

The final version of the program provides functionalities to analyse and match patterns against multiple data sets. This program can be used not only in the domain of myocardial damage but in a wide range of domains. It is easy to add new analysis modules and new patterns for different domains.

In conclusion, the primary and secondary goals of this software engineering project have been achieved and the result is more than satisfactory.

7.1 Discussion

The project have been completed on time but more features and a better pattern editing UI could have been implemented, had the project been managed better. The following issues have added unnecessary work:

The specifications of the core algorithm (pattern discovery) have been implemented gradually along the project. The revised specifications forced an extensive rewrite of almost all the classes and structures of each prototype. If the core algorithm were fully implemented in the first prototype then I would have had more time to add and improve the functionalities of the UI for the last version.

This happened because all the important specifications were not firmly settled during the first weeks of the project. I should have been more proactive and should have asked for details about the core algorithm. Also, it would have been good to fully implement the core algorithm, without any GUI, and to agree with the customer representative that the core algorithm was complete and correct, before the first prototype production.

.

7.2 Further work

This is a list of possible new features and improvements for the next version. Some of these improvements have been proposed by the program testers.

7.2.1 Data files

- More file formats to load data segments, e.g. excel
- A panel to edit/extend the date formats accepted by the CSV parser
- Allow the user to edit the data
- Export the processed data as files
- Allow data with mixed time point periods in the same project

7.2.2 Data Panel

- Synchronize scroll bars for the graphs of the three tabs.
- Remove lines in discrete graphs

7.2.3 Variables

- Overlap discrete ranges; add an option to specify whether or not this can occur
- A button in the variable dialog which selects all the variables

7.2.4 Data analysis

- Analyse the continuous data
- Analysis time frame after and/or around the special event
- Allow the user to edit the Special Events
- A check button to allow the Data Segments to overlap or to not overlap
- Don’t sort the discrete ranges in the report

7.2.5 Pattern discovery

- Show more helpful information in the pattern creation dialog
- Ability to edit patterns
- Reference existing patterns inside the pattern-creating facility(s)
- Advanced UI pattern entry, like a text parser to input patterns with a text field

- Show the list of matched data segments in each pattern and the list of matching patterns in each data segment. Also, highlight in the data segment the matched time points of the pattern.
- Store/load the reports
- PDF reports
- Export the patterns in standard file formats like CSV.

8 References

[AG 1] http://en.wikipedia.org/wiki/Agile_software_development (19 January 2012 20:00h).

[AG 2] http://agilemanifesto.org/ (19 January 2012 20:00h).

[DS 11] Derek Sleeman, Laura Moss, Malcolm Sim, and John Kinsella. 2011. Predicting adverse events: detecting myocardial damage in intensive care unit (ICU) patients. In Proceedings of the sixth international conference on Knowledge capture (K-CAP '11).

[EG 94] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides. 1 edition (November 10, 1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

[LAX 06] Laxman S. and Sastry P,S. 2006. A survey of temporal data mining. SADHANA, Academy Proceedings in Engineering Sciences 31, 2.

[MVC 1] http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller (19 January 2012 20:00h).

[MVC 2] http://www.oracle.com/technetwork/articles/javase/index-142890.html (19 January 2012 20:00h).

[XP 1] http://en.wikipedia.org/wiki/Extreme_Programming (19 January 2012 20:00h).

[XP 2] http://www.cs.usfca.edu/~parrt/course/601/lectures/xp.html (19 January 2012 at 20:00h).

9 Appendices

9.1 User Manual

The User Manual details how to use the TDWB program, including instructions for all of the functionalities it provides.

9.1.1 Running the program

Just double click the file TDWB.jar located in the 'dist' folder of the NetBeans project.

If you want to execute the program from NetBeans, first open NetBeans and then open this project by click File -> Open project in the upper menu of NetBeans.

Then, select the folder of this project and click on the "Open project" button. To execute the program is just press the key "F6" in your keyboard.

9.1.2 Data files

Files with a specific format are required to be loaded into the program.

9.1.2.1 CSV (Comma Separated Values)

The kind of files accepted by the program are the comma separated values files, or CSV files. These files are plain text format files with a "csv" file extension, these can be edited with any standard text editor, but the easiest way to edit them is with Microsoft Excel or with OpenOffice.org Calc.

OpenOffice.org Calc is open source, free and can be downloaded from its website http://www.openoffice.org

The CSV format requires that each value or field is separated by a special character. The special characters accepted by TDWB are the comma (,) and the semicolon (;).

Each record is on a single line.

The first line of the file must contain the names of the variables. This is a example of a CSV file using the comma character as the field separator:

```
Year,Make,Model,Length
1997,Ford,E350,2.34
2000,Mercury,Cougar,2.38
```

In this example there are four variables: Year, Make, Model and Length; and two records, each one ending with a line-break.

More information about CSV files can be found in the Wikipedia: http://en.wikipedia.org/wiki/Comma-separated_values

9.1.2.2 How to use OpenOffice.org Calc

If you don't have installed OpenOffice.org Calc, first you need to download it from its website http://www.openoffice.org and install it on your system.

The OpenOffice.org Calc version used for this manual is the 3.3.0.

To create a new CSV file:

1. Open OpenOffice.org Calc
2. In the new file, write in the first row the variables name. Each variable name must be in a different cell.
3. For each data record write the values of each variable in the corresponding column. At the end you should have something like Figure

9-1.

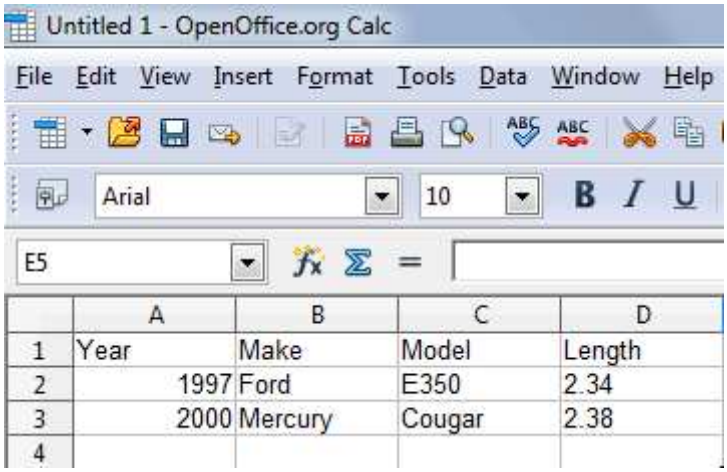

Figure 9-1 – Creating/editing a CSV file with OpenOffice.org Calc

4. To save the document as a CSV file: In the upper menu click: File -> Save As… Then, select the CSV format. When prompted, choose the comma (,) or the semicolon (;) as the field delimiter.

9.1.2.3 Required variables

TDWB requires two specific variables in each data file to load them. One of the variables is the **Time** variable. This variable is the time stamp for each data record. The formats accepted by this program for the time stamps are:

```
dd-MM-yyyy kk:mm:ss.SSS
dd-MM-yyyy kk:mm
dd-MM-yyyy kk:mm:ss
yyyy-MM-dd kk:mm
yyyy-MM-dd kk:mm:ss
yyyy-MM-dd kk:mm:ss.SSS
dd/MM/yyyy kk:mm
dd/MM/yyyy kk:mm:ss
dd/MM/yyyy kk:mm:ss.SSS
yyyy/MM/dd kk:mm
yyyy/MM/dd kk:mm:ss
yyyy/MM/dd kk:mm:ss.SSS
yyyy.MM.dd G 'at' HH:mm:ss z
h:mm a
yyyyy.MMMMM.dd GGG hh:mm aaa
yyMMddHHmmssZ
yyyy-MM-dd'T'HH:mm:ss.SSSZ
```

Where, the pattern letters are described in Figure 3-2:

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Figure 9-2 – Date time pattern letters

The other variable is the **Special Event** variable. Remember that the purpose of the TDWB system is to help the domain experts find patterns in the data before a specific event happens. For example, special events could be that starts to rain, or that a patient in the ICU suffers myocardial damage, so you need to “tell” the system when these special events happen.

The special events can be POSITIVE or NEGATIVE. If it is positive, means that a special event happened; if negative, means that there is not special event. This is useful for the training data segments. A perfect pattern will match only the data segments with a positive special event and none of the data segments with a negative special event (NSE).

For example, Figure 9-3 shows a data file with a positive special event.

Time	Humidity (%)	Special Event
16/10/08 01:18	80	
16/10/08 02:00	80	
16/10/08 03:00	85	
16/10/08 04:00	85	POSITIVE

Figure 9-3 – Positive special event

In this case, we can guess that when the humidity changes from 80 to 85 then it starts to rain in the next hour. Figure 9-4 shows a negative special event.

Time	Humidity (%)	Special Event
16/10/08 01:18	80	
16/10/08 02:00	80	
16/10/08 03:00	80	
16/10/08 04:00	80	NEGATIVE

Figure 9-4 – Negative special event

This means that there is no rain after 3 hours of 80% humidity. So when the domain expert writes a pattern, he/she should write something like the positive events happen when the humidity reaches 85%.

Duplicated variable names are not accepted in the data files.

Each special event (Positive or negative) determines the end of a **data segment**.

9.1.3 Creating a new Project

To start a new project in TDWB you have to click, on the upper menu, Project -> New project. And a dialog like Figure 9-5 appears.

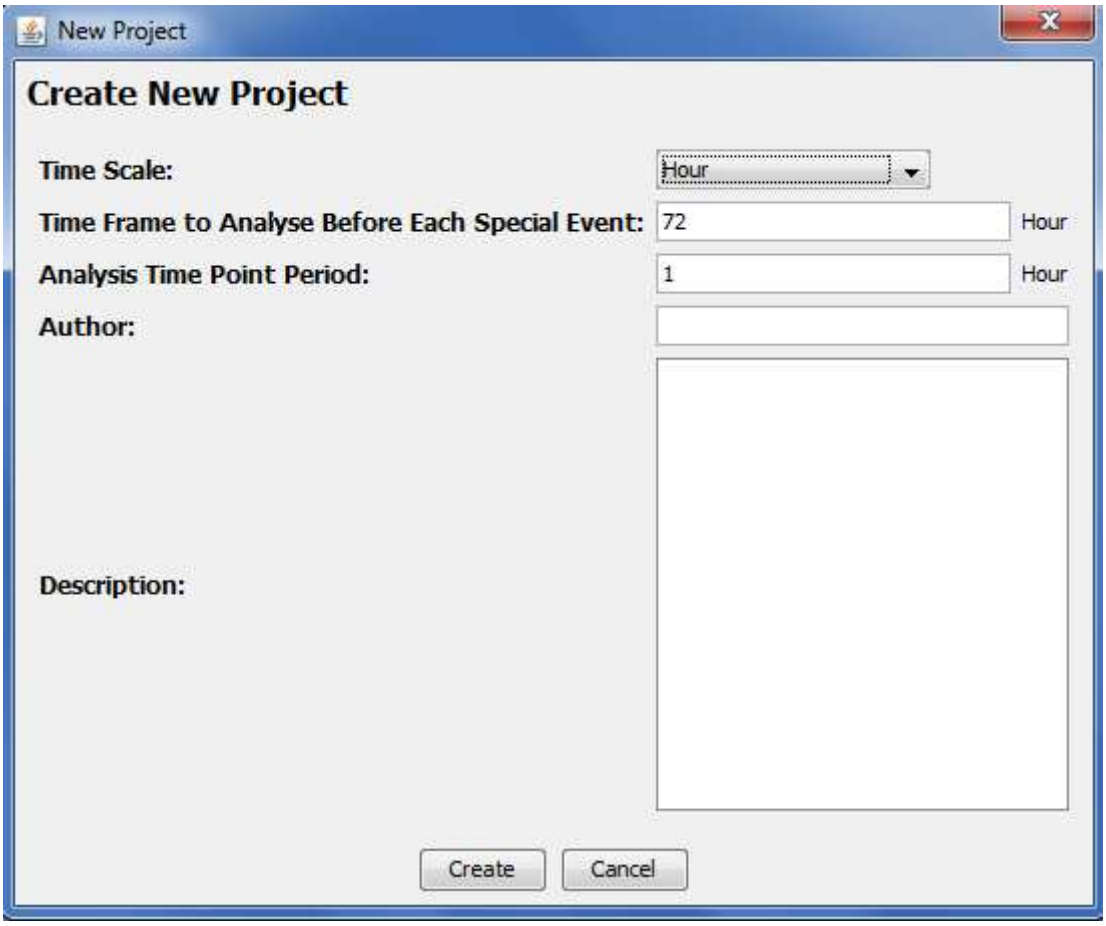

Figure 9-5 – New project/project properties dialog

The "**Time Scale**" field sets the units for the analysis period. For example, for a medical study about patients in the ICU, could be hours because the data from the patients are collected hourly.

The "**Time Frame to Analyse Before Each Special Event**" determines the length of time, before a special event, that would be analysed in the analysis and the pattern discovery processes. The **data segments** are as long as this parameter. So for example, if we have a data file with two special events, and a time frame to analyse before each special event of 72h, then we have two data segments of 72h long in this data file, like described in Figure 9-6.

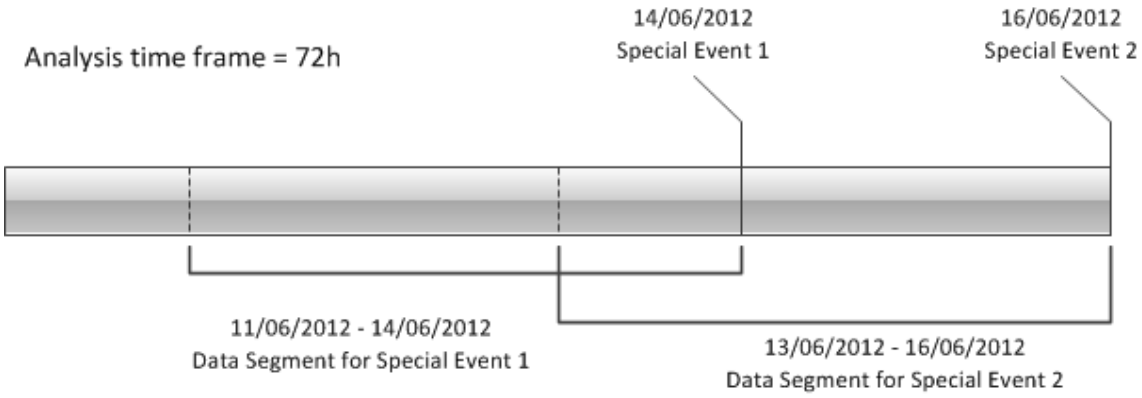

Figure 9-6 – Data segments overlapping

If the special events are closer than the analysis time frame, then their corresponding data segments will overlap. That could affect the analysis if both special events are of a different kind. To avoid this situation it is better to split the data file in two different data files with their corresponding non-overlapping data segments.

The "**Analysis Time Point Period**" is the time period of the data records. For example, if the data from a patient in the ICU is recorded once per hour and the time scale is set to hours, this field should be '1'.

If you need to change these values, you can do so by opening the dialog from the Figure 9-5 clicking in the upper menu, in Project -> Project properties.

9.1.4 Adding data files

After creating a new project, the system will ask to the user to load data for the analysis and the pattern discovery process. A dialog like Figure 9-7 appears to load a data file.

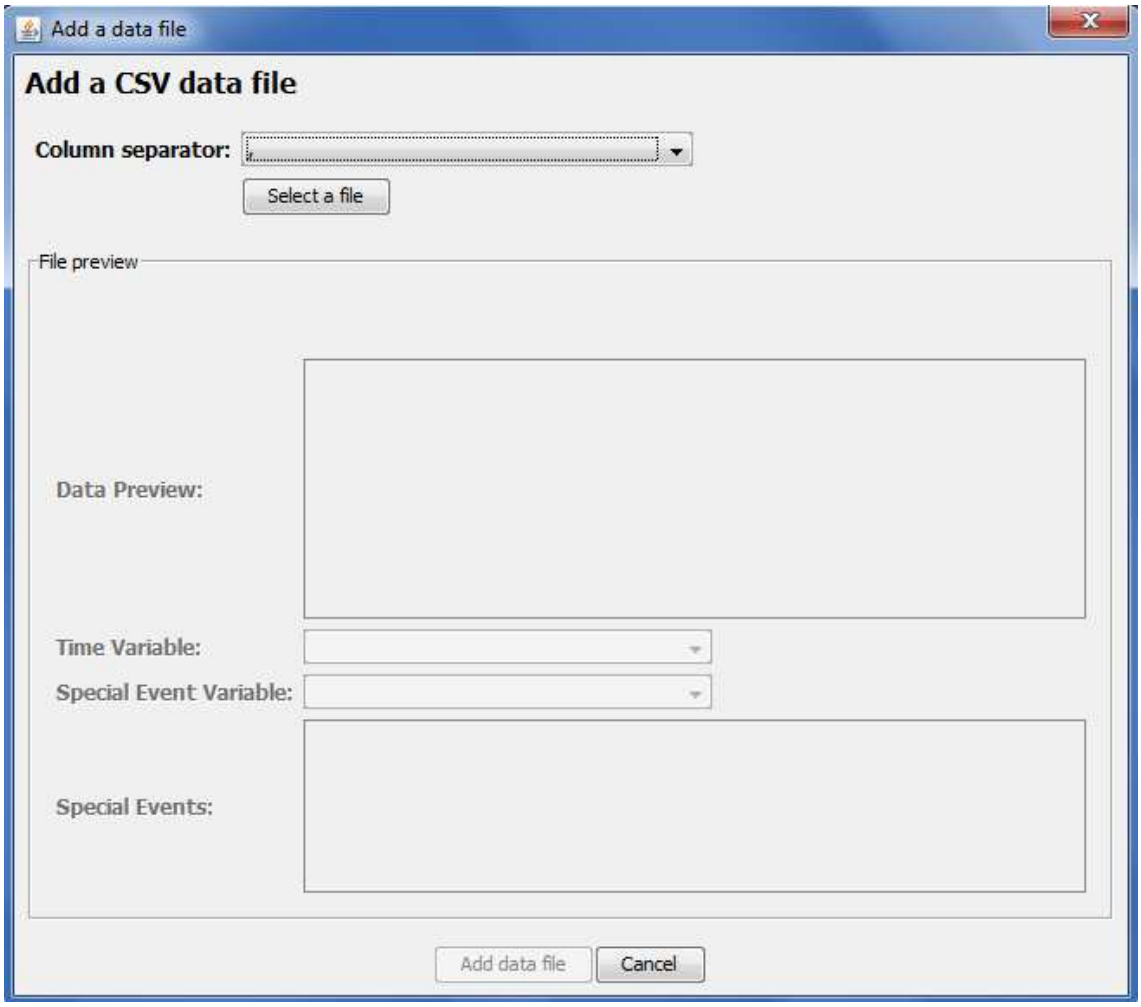

Figure 9-7 – Add data file dialog

Also, you can access this dialog by clicking in the Data Files tab, near the top of the system screen, and then click on the button “Add a data File” which is in the upper left corner of the panel, now it is time to load the CSV file that contains the analysis data.

1. Select the “**Column separator**”; namely, the character that delimits the values in the CSV file to be loaded.
2. Click in the button “**Select a file**” to load the CSV file.
3. If the format of the file is correct (with a time variable which has time stamps in the correct format, with a special event variable with one or more positive and/or negative special events and without duplicated

variable names) then the data is previewed in the "**File preview**" section as seen in Figure 9-8.

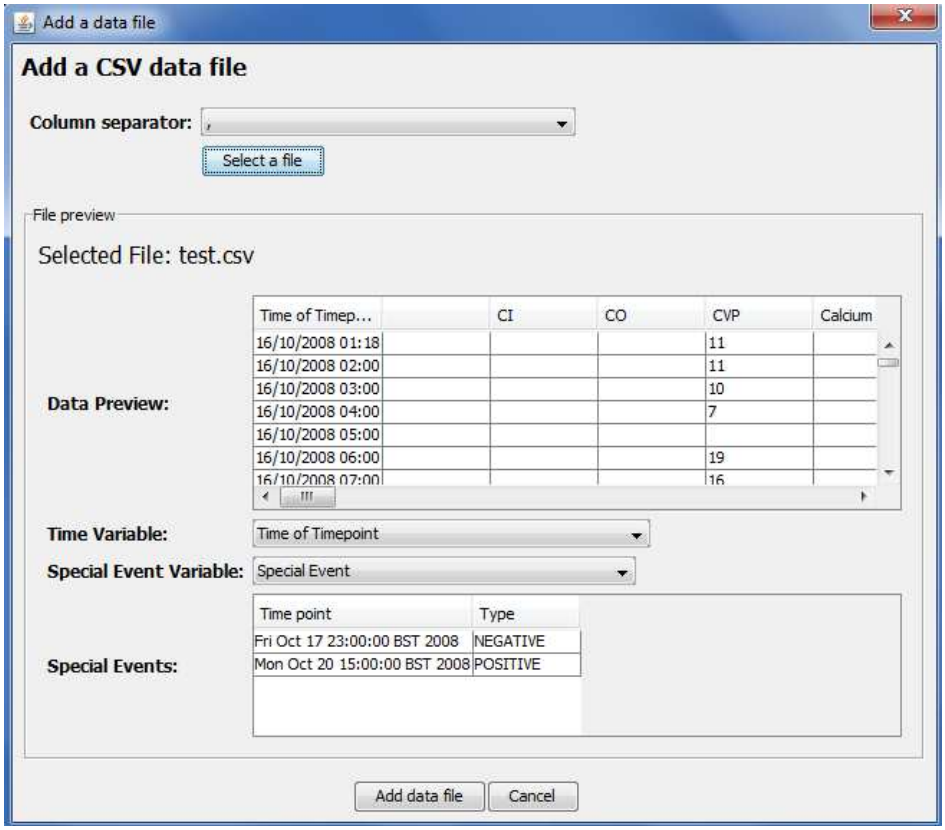

Figure 9-8 – Add data file dialog with data preview

4. If the system detects more than one possible time variable and/or more than one possible special event variable, you have to choose which variables are the correct ones.
5. When everything is as desired, click on the button "**Add data file**" to add the data file into the system.

After this, the system will prompt you about add more data files, setting variables for the analysis or closing the dialog.

9.1.5 Selecting and configuring the variables to analyse

If you want to select and configure the variables, a dialog like in Figure 9-9 appears.

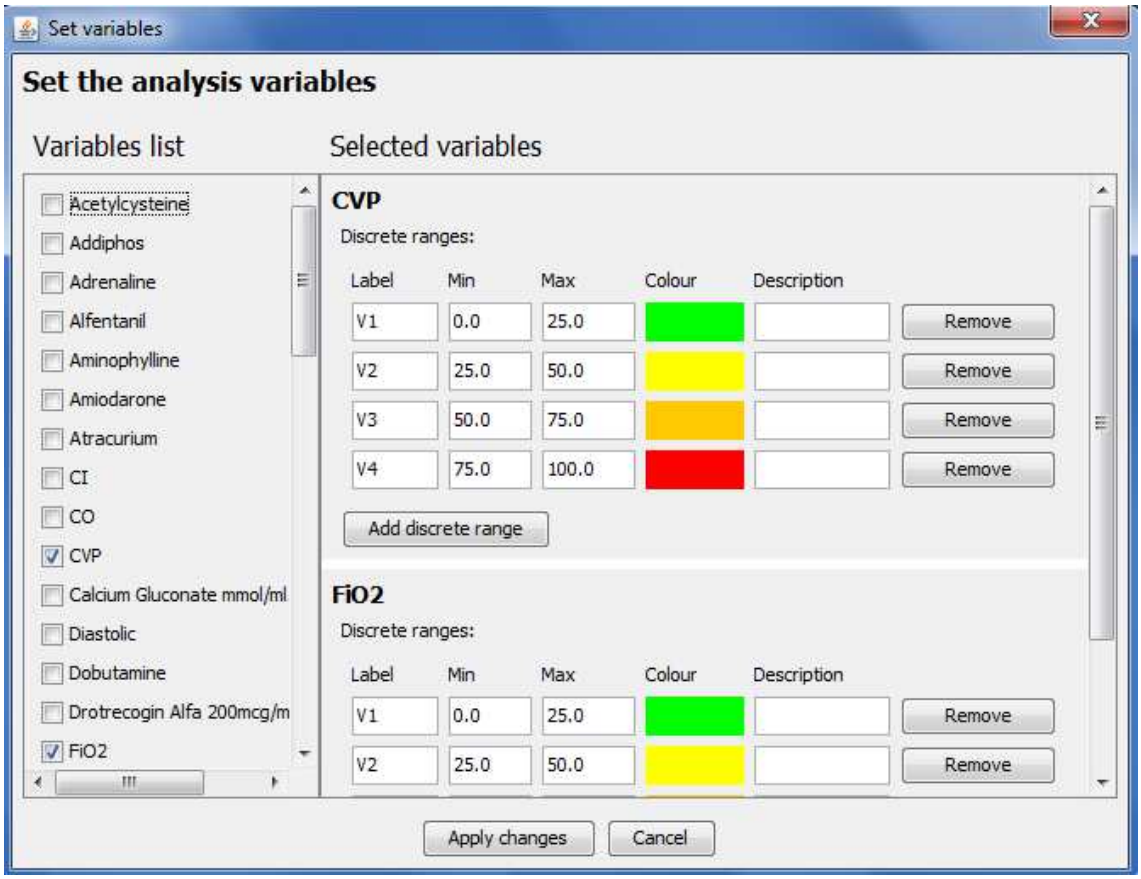

Figure 9-9 – Set variables dialog

Also, you can access this dialog whenever you wish by clicking the upper menu: Variables -> Set variables.

In the left panel you can select the variables to be analysed, in this example, the CVP and FiO2 variables are selected.

In the right panel you can define the range of values that classifies the values of each variable in different classes or discrete values. For example, the domain expert thinks or knows that the range of values for the CVP variable between 0 and 5 are too low. So then, the domain expert should set discrete ranges as in Figure 9-10.

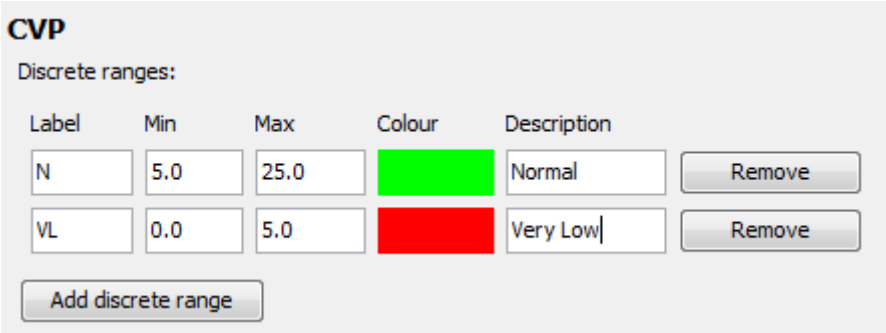

Figure 9-10 – Discrete ranges panel

The field "**Label**" is required and duplicated values are not accepted.

The value in the field "min" must be less or equal than the value in the field "max".

The discrete ranges of a variable cannot overlap. For example, if a variable has two discrete ranges "dr1" and "dr2" with the values min=0, max=50 for "dr1" and the values min=40 and max=70 for "dr2" then the system will give an alert when pressing the "Apply changes" button of the dialog.

When generating the discrete values for the data analysis, a continuous value belongs to a range if it is greater or equal than "Min" and less than "Max".

When everything is defined, to accept the changes you have to click on the "**Apply changes**" button.

9.1.6 Analysing the data

After loading all the data files and setting the variables. In the **Data Files** tab, we can visually check the data and its discrete ranges, as seen in Figure 9-11.

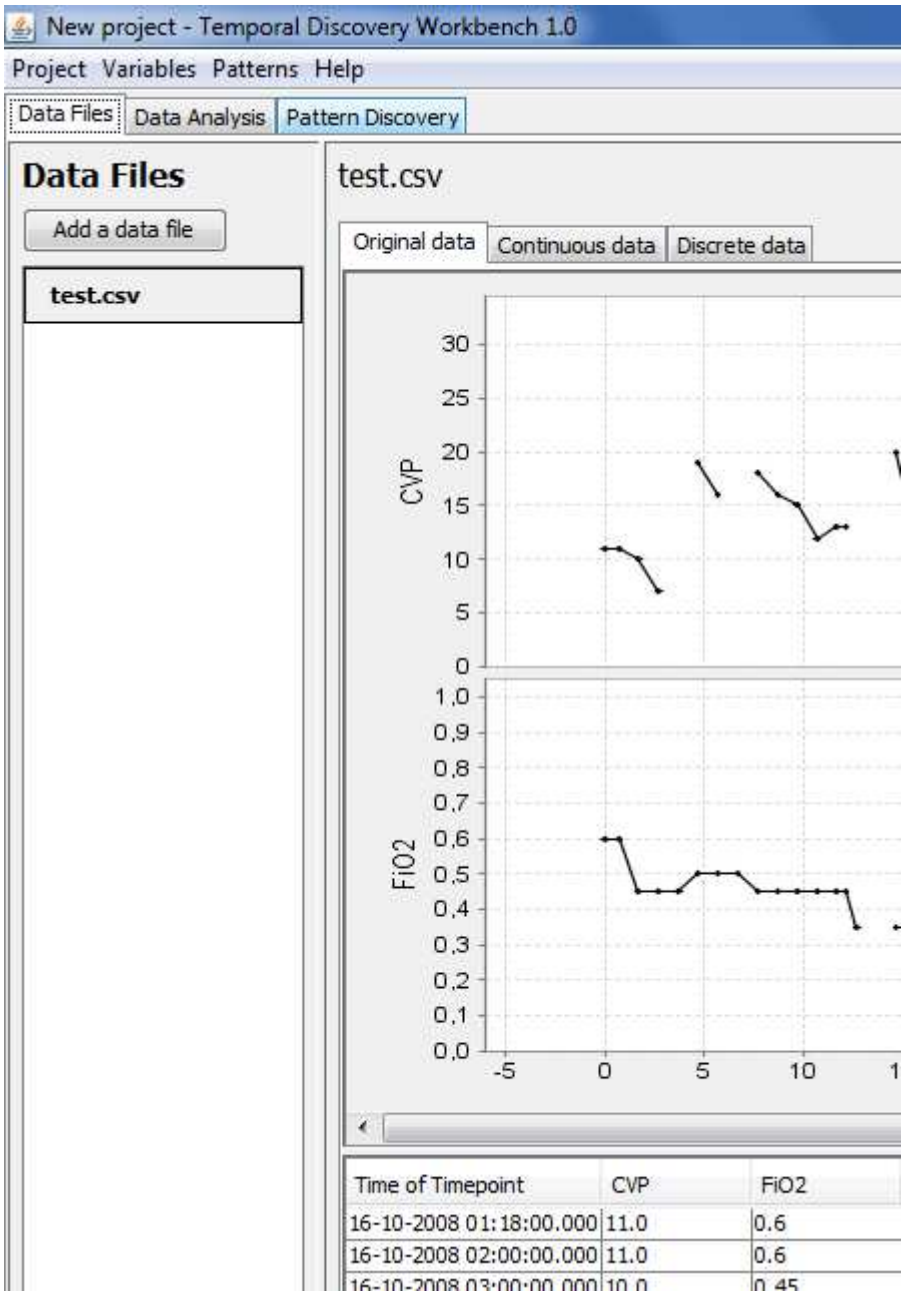

Figure 9-11 – Data files panel

In the right panel you can see the graphs and the values of the loaded data files. In the "**Original data**" tab the original data is displayed.

In the "**Continuous data**" tab a smoothed version of the loaded data is displayed. That is, time points generated in the period defined when creating the project, see Figure 9-5. If for example the defined time point period is 1h and in the loaded data are recorded every 30 minutes, which is 2 per hour, the

system will generate the mean of all the records within hour period. Also, as you can see in Figure 9-12, in the continuous tab the colours of the discrete ranges are shown in a way that you can see which values fall into each range.

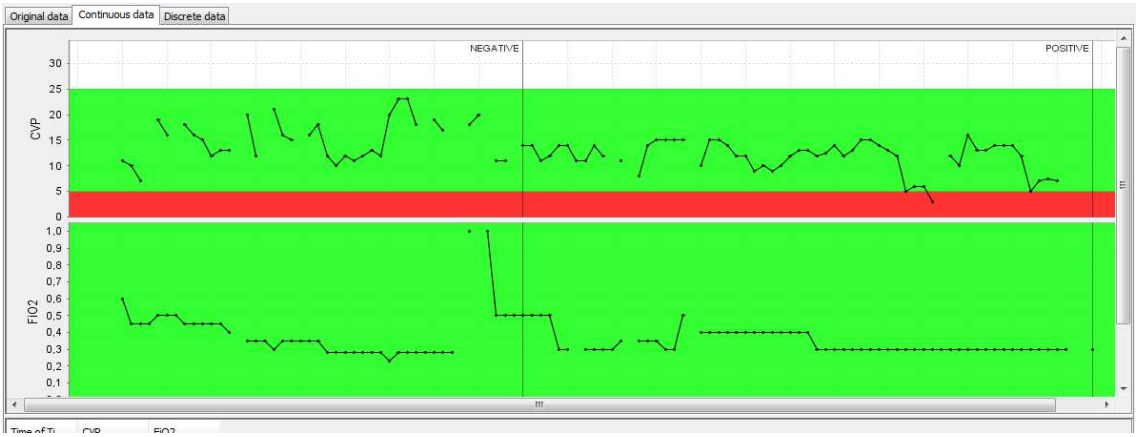

Figure 9-12 – Continuous data graph

In the CVP variable there is a value that falls in the very low (VL) discrete range in less that 72h before the positive special event occurs, but not in the negative special event. So we can guess that a positive special event happens after the variable CVP drops down into the very low (VL) range.

Also, there are two values very near of the VL range. To see more precisely if that values are or not in the VL range we can check the "**Discrete data**" tab shown in Figure 9-13.

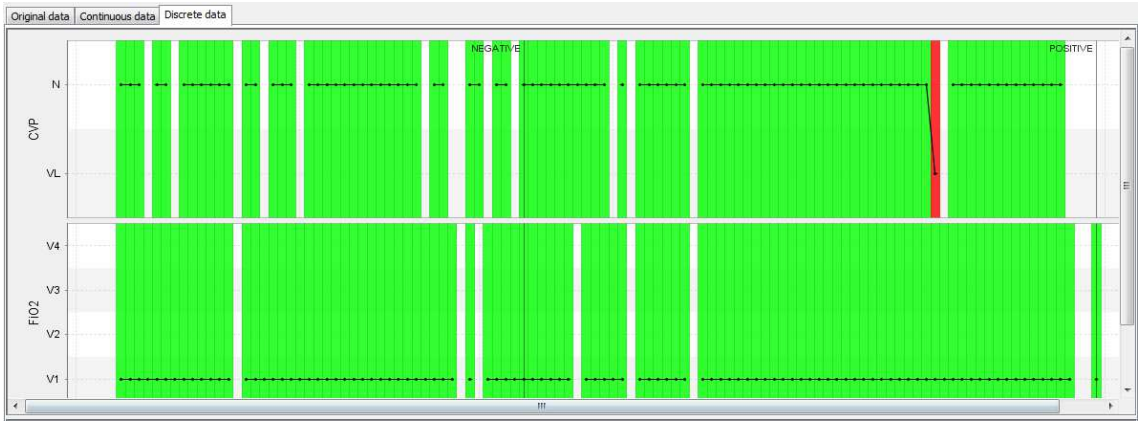

Figure 9-13 – Discrete data graph

Definitely, checking this graph, we can say that only one value of CVP is in the VL range.

Another useful tool to analyse the data are the analysis modules. These modules can be found in the **Data Analysis** tab, near the top of windows program.

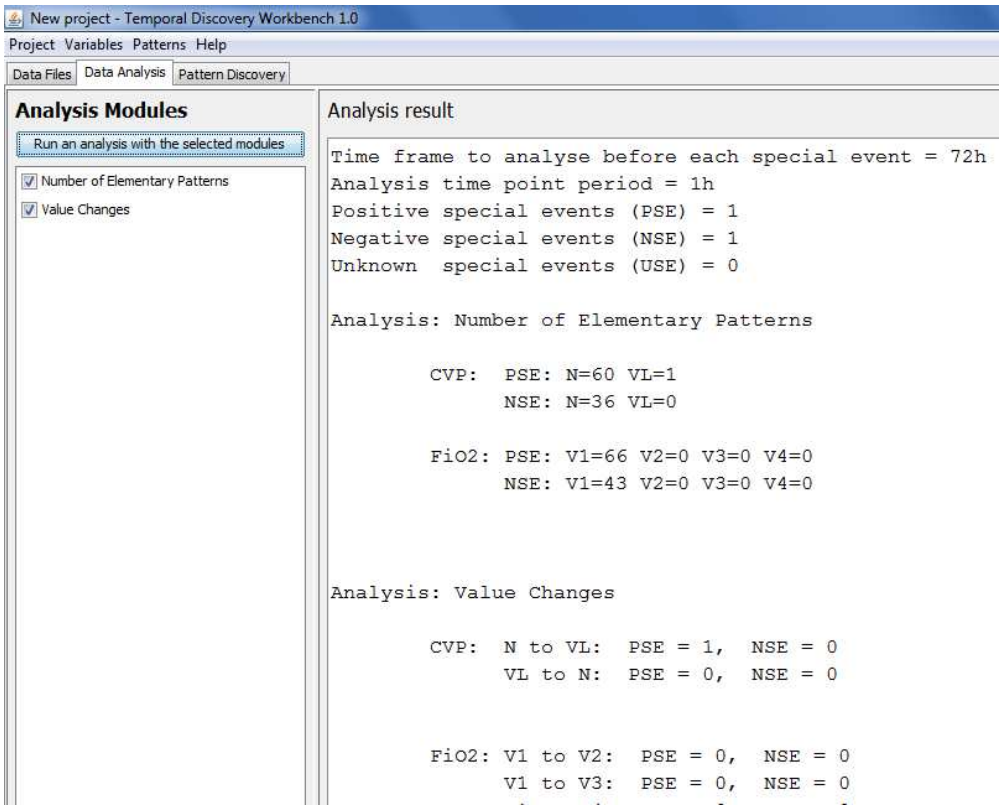

Figure 9-14 – Analysis report

In Figure 9-14, in the left panel the modules to analyse the data can be selected.

After selecting the desired modules, click on the "**Run an analysis with the selected modules**" to run the analysis. After that, in the right panel the result of the analysis is shown.

In this case we can see that, for all the positive data segments, the CVP variable contains one VL value, and also one change from N to VL.

9.1.7 Pattern discovery process

After analysing the data, it is time to formalize all the hypotheses. This is done by writing a pattern and matching that pattern against all the data segments. (Remember that a data segment is defined by a special event and by the analysis time frame before each special event).

The pattern discovery process is done in the **Pattern Discovery** tab, which can be found near the top of the windows program.

To add a new pattern to match, click in the "**Add a new pattern**" button, in the upper left corner. You can then choose between three kinds of patterns:

- Elementary pattern
- Composite pattern
- Combinatory pattern

In all the patterns there is a "**Not**" option. If this option is selected, the program will report as a match the not matching data and as not matches the matching data. This could be useful to build patterns that match negative special events.

Following are described each kind of pattern.

9.1.7.1 Elementary pattern

This is the simplest pattern, which looks for the discrete range of a variable.

The elementary patterns are defined by triplets of the form "Variable name[Discrete label]". For example, for the 'CVP' with discrete ranges [N, VL], a possible tuple would be "CVP[N]" or "CVP[VL]". The elementary patterns are checked at each time point of every data segment.

For example, the pattern "CVP[VL]"

For the sequence:

```
CVP: N N VL
```

Reports a match:

```
CVP: N N VL
```

Figure 9-15 is the dialog to add a new elementary pattern.

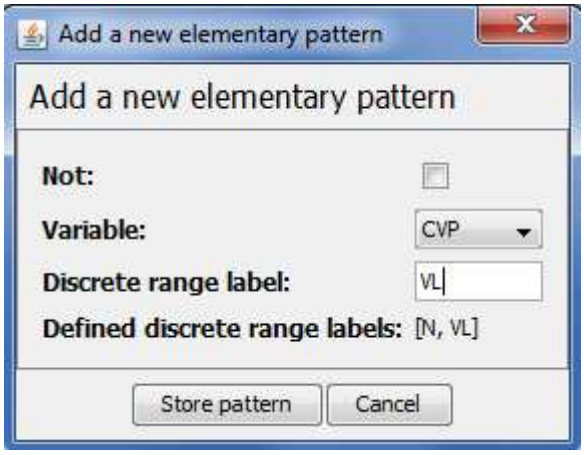

Figure 9-15 – New elementary pattern dialog

The “**Discrete label**” field is to input the discrete range label. The “**Possible label values**” shows the labels for the discrete ranges of the selected variable.

9.1.7.2 Composite pattern

A composite pattern is composed of one or more patterns. If when matched against data at a time point, all of its component patterns match, then the composite pattern reports a match. If some of its patterns don’t report a match then the composite pattern doesn’t report a match.

The component patterns can have different time offset. The time offset parameter is added to the pattern with the format “T+**Time point offset**:Pattern”.

For example, the pattern “Composite(T+0: (CVP[N]), T+1: (CVP[VL]))”:

For the sequence:

```
CVP: N N VL
```

Reports a match:

```
CVP: N N VL
```

Because a pattern CVP[VL] is found one time slot after a pattern CVP[N].

The above pattern is defined by adding two elementary patterns to a composite pattern and defining a time offset (T+) of 1 to the second elementary pattern. See Figure 9-16.

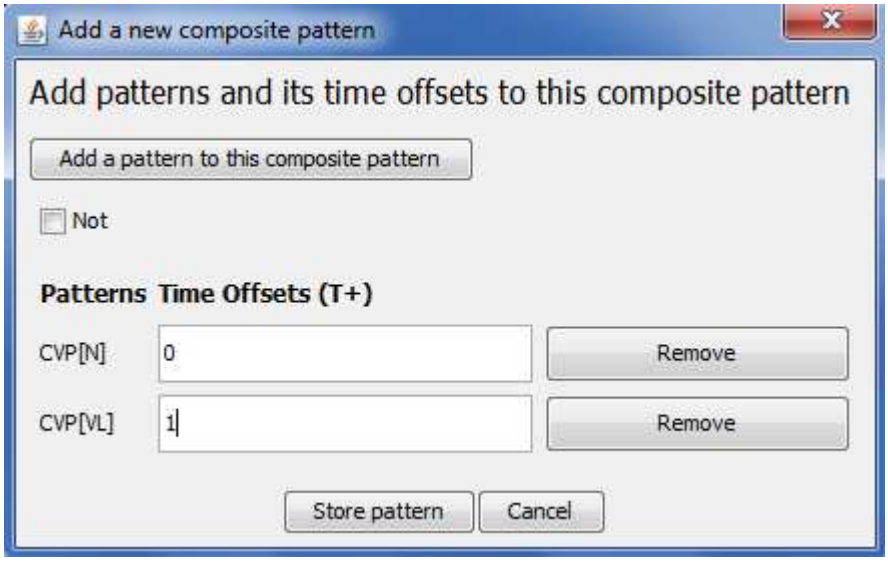

Figure 9-16 – New composite pattern dialog

9.1.7.3 Combinatory pattern

A combinatory pattern is, like the composite pattern, composed of a number of patterns. The difference with the composite pattern is that the component patterns are combined in all possible ways to generate all possible composite patterns. For example, if the user define a combinatory pattern of two CVP[N] and one CVP[VL], the generated composite patterns would be:

```
Composite(T+0: (CVP[N]), T+1: (CVP[N]), T+2: (CVP[VL]))
Composite(T+0: (CVP[N]), T+2: (CVP[N]), T+2: (CVP[VL]))
Composite(T+0: (CVP[N]), T+1: (CVP[VL]), T+1: (CVP[N]))
Composite(T+0: (CVP[N]), T+1: (CVP[VL]), T+2: (CVP[N]))
Composite(T+0: (CVP[VL]), T+0: (CVP[N]), T+1: (CVP[N]))
Composite(T+0: (CVP[VL]), T+1: (CVP[N]), T+2: (CVP[N]))
```

An example of this combinatory pattern can be seen in Figure 9-17.

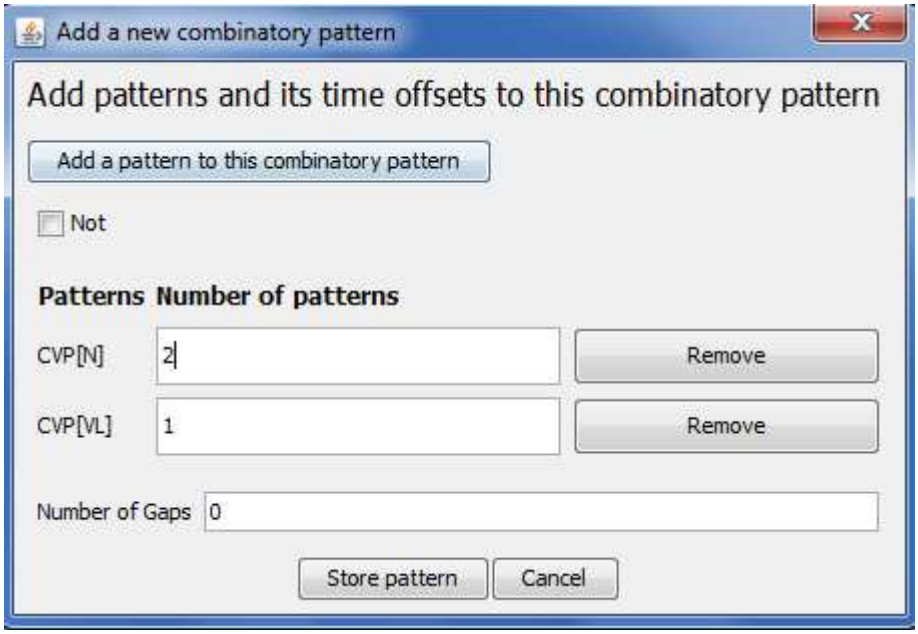

Figure 9-17 – New combinatory pattern dialog

Also, there is a “Gaps” parameter, which adds extra time gaps to the generated composite patterns, for example for the pattern Combinatory(2xCVP[N], 2xGaps) generates the composite patterns:

```
Composite(T+0: (CVP[N]),  T+1: (CVP[N]))
Composite(T+0: (CVP[N]),  T+2: (CVP[N]))
Composite(T+0: (CVP[N]),  T+3: (CVP[N]))
```

9.1.7.4 Creating complex patterns

The composite and combinatory patterns are composed by other patterns; the other patterns can be elementary, composite or combinatory patterns. This allows you to create for example a combinatory pattern composed of composite patterns. An example of this complex patters is shown in Figure 6-3.

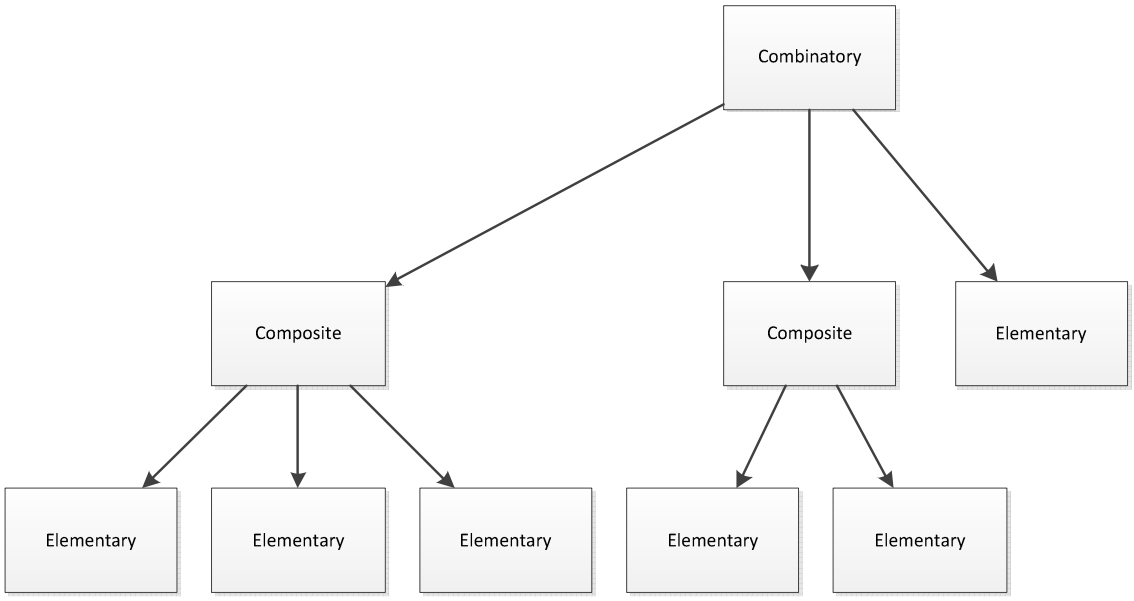

Figure 9-18 – Complex pattern

9.1.7.5 Matching thresholds

Once you have created your patterns it is time to define their matching thresholds. The patterns are tested against all the possible time points of the data segments. For example, for the data set:

```
A:  N, N, N, L, L, L, L, N, N, N
B:  N, N, H, H, H, H, N, N, N, N
```

If the pattern to be matched is **A[L]**, then this results in **4** matches out of **10** possible matches:

```
A:  N, N, N, L, L, L, L, N, N, N
B:  N, N, H, H, H, H, N, N, N, N
```

We have 4/10·100=**40%** matches against this data segment.

If the pattern to match is **Composite(T+0: (A[L]), T+1: (B[H]))**, the result is **2** matches in **9** possible matches:

A: N, N, N, **L, L**, L, L, N, N, N

B: N, N, H, H, **H, H**, N, N, N, N

We have 2/9·100=**22.22%** matches against this data segment. Is 9 possible matches because the length of the data is 10, and the length of the pattern is 2, so 10-2+1 = 9 possible matches.

To determine if the pattern matches against the data segment, the user determines a threshold for the number of matches or for the percentage of matches in a data segment.

For the patterns P1, P2 and P3 if the user determines the thresholds as in Table 6-1.

Thresholds	%	#
P1	40.00	-
P2	--.--	1
P3	0.13	23

Table 9-1 Pattern thresholds example

The pattern matching algorithm will report a positive data segment match it:

- Finds 40% or more matches for the pattern P1; or
- Finds 1 or more matches for the pattern P2; or
- Finds 0.13% or more, or a number of 23 or more matches for the pattern P3.

These thresholds can be defined in the “**Pattern Discovery**” panel. See Figure 6-15.

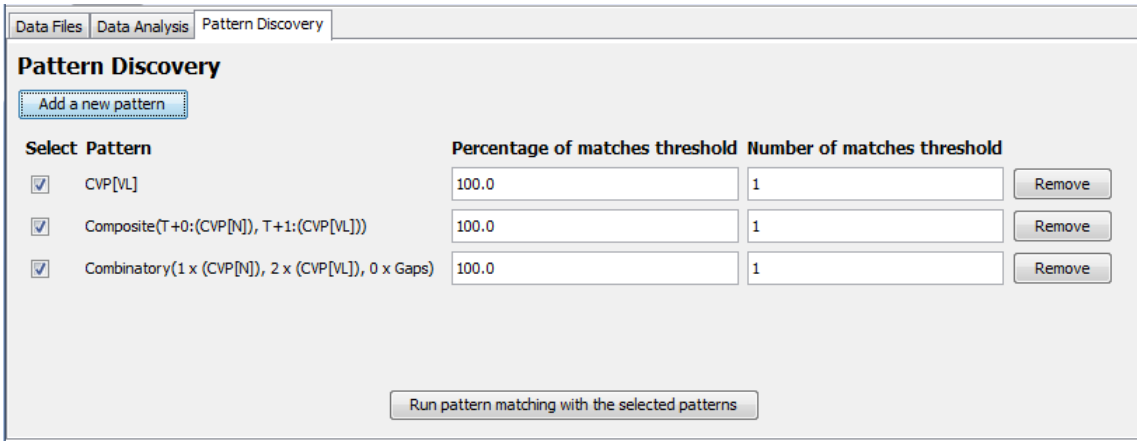

Figure 9-19 – Pattern discovery panel

9.1.7.6 The pattern matching report

When all the patterns with their thresholds are defined it is time to check the patterns against the data segments. To do that you have to click on the "**Run pattern matching with the selected patterns**" button, that is on the bottom of the panel.

Then, all the patterns are checked against all the positive and negative data segments. The program then displays a report similar to Figure 6-16.

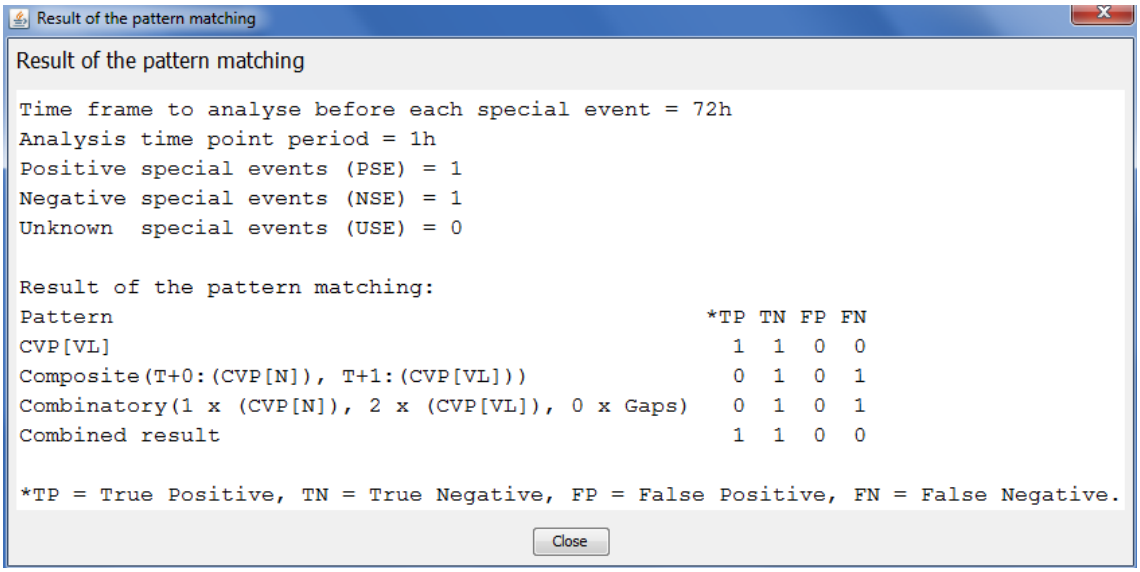

Figure 9-20 – Pattern matching report

When a pattern is matched against a data segment, four possible results can be reported:

- TP: True Positive, if the pattern matches a data segment with a positive special event.
- TN: True Negative, if the pattern doesn’t match a data segment with a negative special event.
- FP: False Positive, if the pattern doesn’t match a data segment with a positive special event.
- FN: False Negative, if the pattern matches a data segment with a negative special event.

At the end of the report there is an extra row which provides the combined result of all the patterns.

A perfect pattern only matches the data segments with a positive special event and none of the data segments with a negative special event.

9.1.8 Save and open projects

To save a project into a file, click on Project -> Save project or on Project -> Save project as… on the upper menu if you want to save the project in a different file.

To open an existing project from a file, click on Project -> Open project on the upper menu.

9.1.9 Save and load patterns

To save the patterns into a file, on the upper menu, click on Pattern -> Save patterns.

To load patterns from a file, on the upper menu, click on Pattern -> Load pattern.

9.2 Maintenance Manual

9.2.1 Installing the system

To explore the source code NetBeans is needed. For this project I have used the version 7.0.1 of NetBeans. Visit the website http://netbeans.org to download it and install it in your system.

Once installed, open NetBeans and load this project. To do this, on NetBeans, on the upper menu, click on File -> Open project

Then, select the folder of this project and click on the “Open project” button.

9.2.2 Compile/build the system

To compile the system, on the upper menu of NetBeans, click on Run -> Clean and build the project. This will generate the folder “dist” inside the root folder of the project. There you can find a jar file which is the executable.

9.2.3 Execute the program

To execute the program, double click on the jar file of the “dist” folder.

To execute the program from NetBeans, press the key “F6” on your keyboard.

9.2.4 Dependencies

This program depends of the Java version 6.0 that can be found at http://java.com/es/download/

If you have installed NetBeans, then you will have a newer version of Java 6.0

Other dependency is JFreeChart 1.0.13 and JCommon 1.0.17 but these are included in the source code.

9.2.5 Organisation of files

Files in the root folder:

- build.xml and manifest.mf: NetBeans project configuration files
- Code listing.pdf: the source code of the project
- example.csv: a CSV file example
- Maintenance manual.pdf: this manual
- readme.txt: text file describing the installation, compilation, execution and dependencies of the project
- User manual.pdf: user manual which is copied to the folder dist after compiling the NetBeans project

Folders in the root folder:

- build: compile temporary files
- nbproject: NetBeans project configuration files
- dist: the executable file (TDWB.jar) and the user manual
- dist\lib: libraries used by the executable file
- lib: JFreeChart and JCommon libraries imported in the NetBeans project
- patterns: example pattern files
- projects: example project files
- src: Source code

9.2.6 Model-view-controller

The final version is structured with the classic architectural pattern model-view-controller, or MVC [EG 94].

The model layer contains all the classes that represent the information and the methods to transform it.

The view layer has the classes that draw the GUI in the system screen showing the state of the information and gives mechanisms to allow the user for interact with the system.

The controller is a layer between the view and the model. The controller handles the user events, modifies the model information accordingly to the user's request and then updates the view layer with results. In Figure 6-17 – MVC request process, this process can be seen graphically.

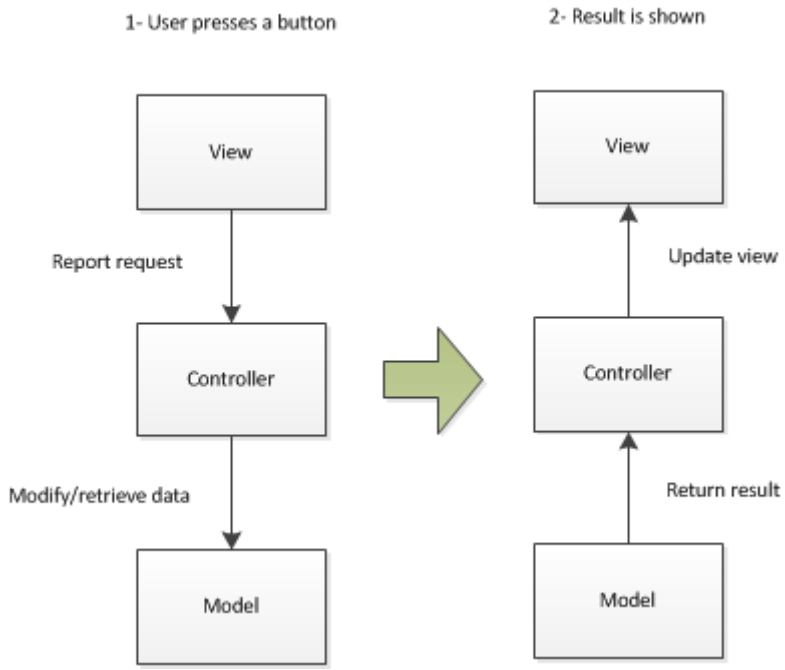

Figure 9-21 – MVC request process

The main purpose of the controller layer is to abstract the model from the view, then it is easier for a programmer to modify the model without modify the view, or to modify the view without modify the model.

More information about this pattern can be found here: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

And here: http://www.oracle.com/technetwork/articles/javase/index-142890.html

9.2.7 UML

9.2.7.1 Model

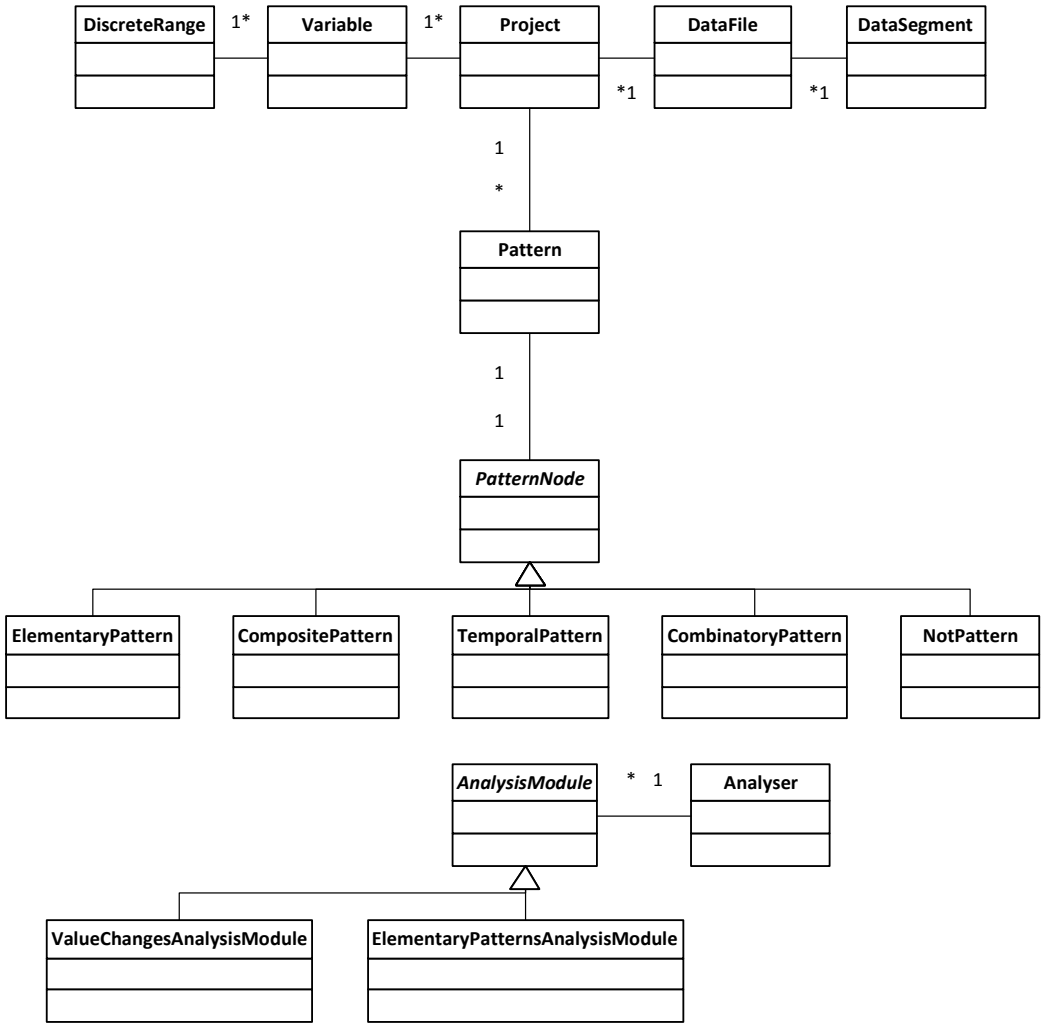

Figure 9-22 - Model UML

9.2.7.2 Controller

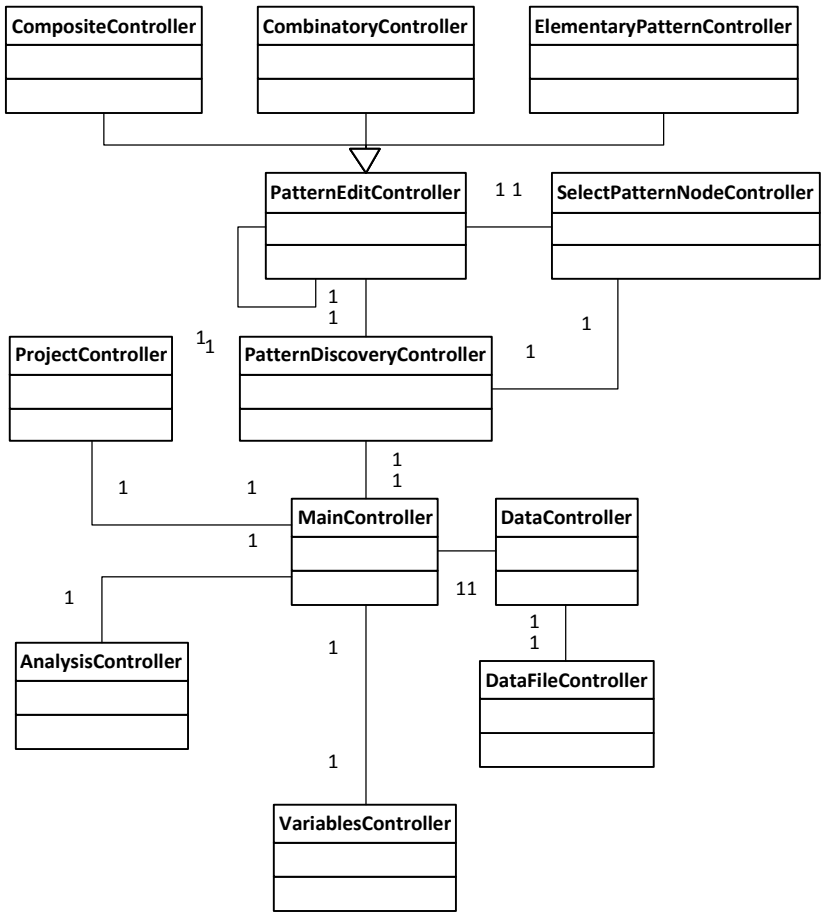

Figure 9-23 - Controller UML

9.2.8 List of source code files

The following are the Java packages and a short description about the files of each package.

9.2.8.1 Tdwb package

This is the root package.

File	Description
App.java	File that contains the main function. Instantiates and shows MainView.java.
Cons.java	Has some constants used in the other classes.

9.2.8.2 Tdwb.controller package

The controller package contains the classes of the controller layer. These classes receive the user's inputs and change the state of the system accordingly- modifying the model layer and reporting the changes to the view layer.

File	Description
AnalysisController.java	Handles the analysis tab panel user events, in the main view.
CombinatoryPatternEditController.java	Handles the CombinatoryPatternEditDialog.java user events.
CompositePatternEditController.java	Handles the CompositePatternEditDialog.java user events.
DataController.java	Handles the data tab panel user events, in the main view.
DataFileController.java	Handles the DataFileDialog.java user events.
ElementaryPatternEditController.java	Handles the ElementaryPatternEditDialog.java user events.
MainController.java	Handles the MainView.java user

	events.
PatternDiscoveryController.java	Handles the pattern tab panel user events, in the main view.
PatternEditController.java	Abstract class implemented by the patterns edit controllers.
PatternTypeEnum.java	Contains the representation of the different kinds of patterns. Also, provide for each pattern its corresponding controller.
ProjectController.java	Handles the dialogs related to create a new project, edit the project properties and the save and load project dialogs user events.
SelectPatternNodeTypeController.java	Handles the SelectPatternNodeTypeDialog.java user events.
VariablesController.java	Handles the VariablesDialog.java user events.

9.2.8.3 Tdwb.model package

The model package contains the classes of the model layer and the core algorithms to process the data and match the patterns.

File	Description
Analyser.java	Proxy class between the analysis modules and the system.
AnalysisModule.java	Interface implemented by the analysis modules. It is used by

	Analyser.java.
CombinatoryPattern.java	Extends PatternNode.java. Represents the combinatory pattern.
CompositePattern.java	Extends PatternNode.java. Represents the composite pattern.
DataFile.java	Contains the original data loaded from the CSV and the continuous and the discrete data. Also contains its data segments.
DataSegment.java	Represents a data segment and its special event.
DiscreteRange.java	Represents a discrete range.
ElementaryPattern.java	Extends PatternNode.java. Represents the elementary pattern.
ElementaryPatternsAnalysisModule.java	Implements AnalysisModule.java and contains the algorithm to calculate the elementary patterns in a data segment.
Pattern.java	Contains a PatternNode and its thresholds.
PatternNode.java	Abstract class used to represent a pattern. It is extended and used by the different patterns. Also, is used by Pattern.java.
Project.java	Represents a project with its parameters, data files, variables

	and patterns.
SpecialEventTypeEnum.java	Contain the representation of the special event types.
ValueChangesAnalysisModule.java	Implements AnalysisModule.java and contains the algorithm to calculate the discrete value changes in a data segment.
Variable.java	Represents a variable with the variable name and its discrete ranges.

9.2.8.4 Tdwb.utils package

The utils package contains useful classes for all the other classes.

File	Description
DataFileReader.java	Used to read the information contained in CSV files.
DateUtils.java	Functions to manage dates.
PatternReaderWriter.java	Used to save and load pattern files.
ProjectReaderWriter.java	Used to save and load project files.
StringUtils.java	Functions related to strings.
TimeScaleEnum.java	Contains the representation of time scales.

9.2.8.5 Tdwb.view package

The view package contains the classes of the view layer. These classes are responsible for drawing the GUI on the system screen and for providing

mechanisms to allow the user to interact with the system. Also shows the state of the model.

File	Description
AboutBox.java	It is a dialog with some information about the program.
AddDiscreteRangeDialog.java	A dialog to add a new discrete range to a variable.
AnalysisPanel.java	The analysis panel displayed in the tab panel of MainView.java.
ChartLibrary.java	A interface class used as a proxy between DataPanel.java and JFreeChartLibrary.java
CombinatoryPatternEditDialog.java	A dialog to add a new combinatory pattern.
CompositePatternEditDialog.java	A dialog to add a new composite pattern.
DataFileDialog.java	The dialog to load a CSV file into the program.
DataModeEnum.java	Contains the representation of the data mode to be displayed.
DataPanel.java	The data panel displayed in the tab panel of MainView.java
ElementaryPatternEditDialog.java	A dialog to add a new elementary pattern.
JFreeChartLibrary.java	The class that interacts directly with the JfreeChart library and implements ChartLibrary.java.

MainView.java	This is the main frame of the UI.
PatternDiscoveryPanel.java	The project discovery panel displayed in the tab panel of MainView.java
PatternMatchingResultDialog.java	Dialog that shows the result of the pattern matching.
ProjectDialog.java	The dialog used to create a new project and to edit the project's properties.
SelectPatternNodeTypeDialog.java	The dialog used to select the kind of pattern to add.
SpringTable.java	It's a swing custom component used for adapt JTables to its contents.
SpringUtilities.java	A swing utility used for set SpringLayouts. This file has been developed by Oracle[1].
TimeSerie.java	Class used by JFreeChartLibrary.java to create a time serie.
UIDiscreteRange.java	Helping class to represent a discrete range in the view layer.
UIPattern.java	Helping class to represent a pattern in the view layer.
UIVariable.java	Helping class to represent a variable in the view layer.
VariablesDialog.java	The dialog used to edit the variables.

[1] http://docs.oracle.com/javase/tutorial/uiswing/examples/layout/SpringGridProject/src/layout/SpringUtilities.java

ViewStyle.java	Constants used by all the view layer classes to format its components.
ViewUtils.java	Functions used by all the view layer classes to format its components and to show confirmation, warning and errors dialog.

9.2.9 Main procedures and methods

9.2.9.1 Generate the analysis data

In the class DataFile there is a method that transforms the raw variable data from the CSV file into analysable data, generating the smoothed continuous values and the discrete values.

9.2.9.2 Analysing the data

The Analyser.java file has a method that analyses the data and returns the result of the analysis report.

9.2.9.3 Generating composite patterns in a combinatory pattern

After a combinatory pattern is created, its composite patterns must be generated. In the file CombinatoryPattern.java are the methods that generate the composite patterns.

9.2.9.4 Pattern matching

The file Pattern.java contains the methods that determine if a pattern matches against a data segment.

The PatternNode.java flips the result of the match (of its subclasses) if the attribute isNot is set to true.

9.2.10 Configuration files

9.2.10.1 Cons.java

Contains some constants and formats for data processing.

- The CSVseparators array contains the characters used as element separator to load the CSV files.
- The timeFormats array contains all the accepted formats to convert dates (from strings) that are loaded as part of the CSV files.

9.2.10.2 TimeScaleEnum.java

This file contains all the time scales accepted by the program.

9.2.10.3 SpecialEventTypeEnum.java

This file contains the different kinds of special events.

9.2.10.4 ViewStyle.java

This file contains constants used to configure the style of the GUI.

9.2.11 Directions for future improvements

9.2.11.1 Changing the graph library

JFreeChart is very complete. But maybe, for a specific domain- or because a new version is available- it is necessary to change it.

I have implemented a class interface named ChartLibrary.java and used the JFreeChart library through this interface. Then, to change the graph library is easier. Figure 6-20 is a UML representing this scheme.

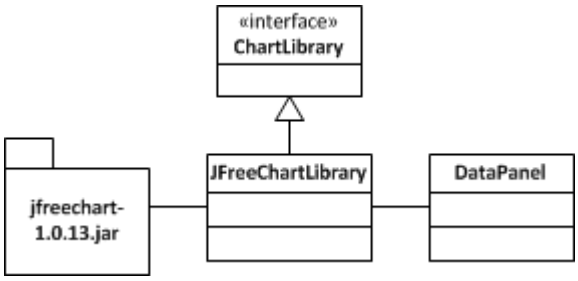

Figure 0-1 – Chart library abstraction

9.2.11.2 Adding new analysis modules

Currently, there are only two analysis modules, but future analysis undertaken by TDWB may require additional modules to be added. So I have implemented a system which allows one to easily add or remove analysis modules.

The analysis modules are implementing classes of a common interface AnalysisModule.java, and Analyser.java is a singleton class that manages the AnalysisModule's subclasses.

To add a new module one simply creates a new implementing class of AnalysisModule.java, implementing the methods getModuleName() and analyse(). Then, one modifies the constructor method of the Analyse.java class to add an instance of the new analysis module. An example of this architecture is shown in Figure 6-21.

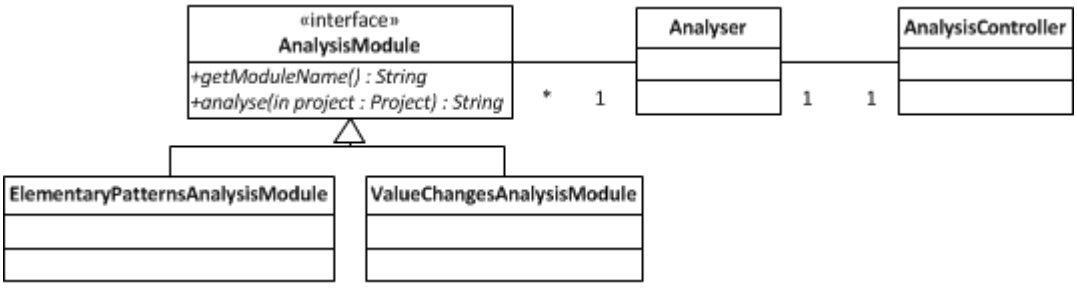

Figure 0-2 – Analysis modules abstraction

9.2.11.3 Adding new pattern types

For the myocardial damage study, combinatorial patterns are needed. For other studies, maybe it would be very useful to add other kinds of patterns that are not implemented currently. So a system to add new patterns easily is needed.

The solution is very similar to the solution for the analysis modules, namely we use an abstract class- PatternNode.java- and a manager class- Pattern.java- that will use the classes that implements PatternNode.java. However, here a different UI and a different controller are needed for the different patterns. This adds complexity to the solution. So I needed to implement the same solution for the controllers, with the abstract class PatternEditController.java.

A further component needed supports a dialog which selects the kind of pattern to be added, and this component needs to know about all the different kinds of patterns. For that purpose I have implemented the SelectPatternNodeTypeController.java and SelectPatternNodeTypeDialog.java classes.

An UML giving the architecture of this solution is shown in Figure 6-22.

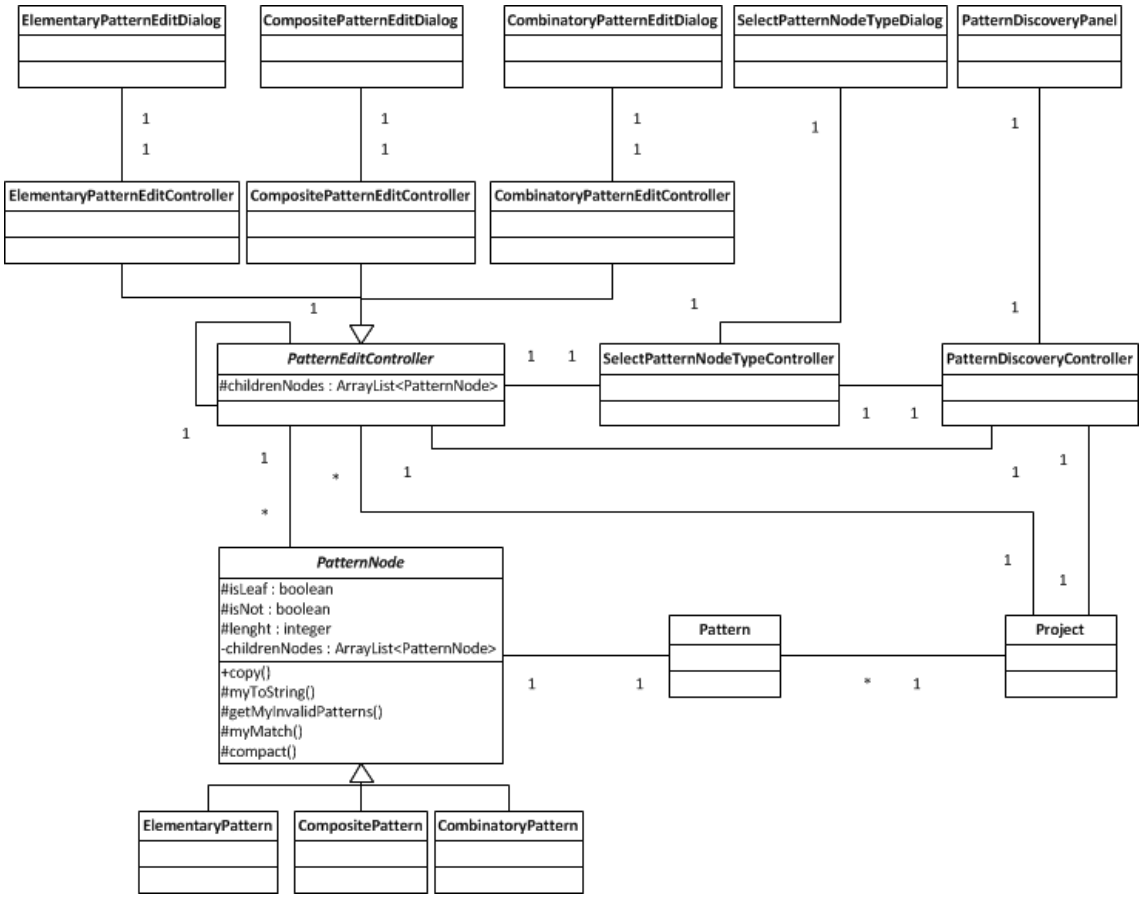

Figure 0-3 – Pattern types abstraction

To add a new pattern, simply write a new class that implements PatternNode.java, a class that implements PatternEditController.java, and its dialog.

To connect it all, edit the file PatternTypeEnum.java, add a new value to the enumeration and edit the function getController.