Linnæus University
School of Computer Science, Physics and Mathematics

Degree project

Visualization in Genealogical Data

Genealogical tree application for Facebook

Author: Jesus Miguel de la Fuente
Date: 2011-09-02
Subject: Information Visualization
Level: Bachelor
Course code: 2DV00E

Acknowledgements

I want to thank and acknowledge all the people who have help me with this thesis in the different ways they could help me.

- Andreas Kerren, because he accepted me to develop this project.
- Ilir Jusufi, to be my guide and spend his time with me helping and guiding with this project
- My family, specially my parents to support me whenever I needed and helping me from Spain and doing my life in Sweden easier.
- My friend family in Sweden. They always have been so nice with me, and thanks to them for the amazing dinners and wonderful company.
- My friends of Erasmus. Without them was impossible to have a wonderful year, great experience and nice university work.
- My friends of Spain, because they always are next to me, in the best times and the worst times.

Abstract

Information visualization is becoming more and more popular due to the development of the Internet and the rapid increase of data. Since human eyes receive visual information very quick and easy, the visualization can make complex and large data more understandable.

In this thesis is developed an application (ex. Facebook Application) that will allow users to build and share visualization of genealogical data. Additionally it should visualize time-dependant data, such as various events like marriage, etc. Since genealogical data are basically trees, it would be easy to adapt our application to fit the needs for other cases, such as company organizational structures charts that need to visualize different time dependant data.

This report describes the requirements for the project and the resulting implementation.

Keywords: Visualization, Visual representations, Genealogical Tree, Hierarchical structures.

CONTENT

Table list

Figure list

1 INTRODUCTION

Facebook is a social networking tool which it is becoming very important. Nowadays, there are more than 400 million of members all over the world. Consequently, that is a good reason to make an application in order to visualize genealogical data. Currently, the number of applications now in Facebook is more than 500.000. [1]

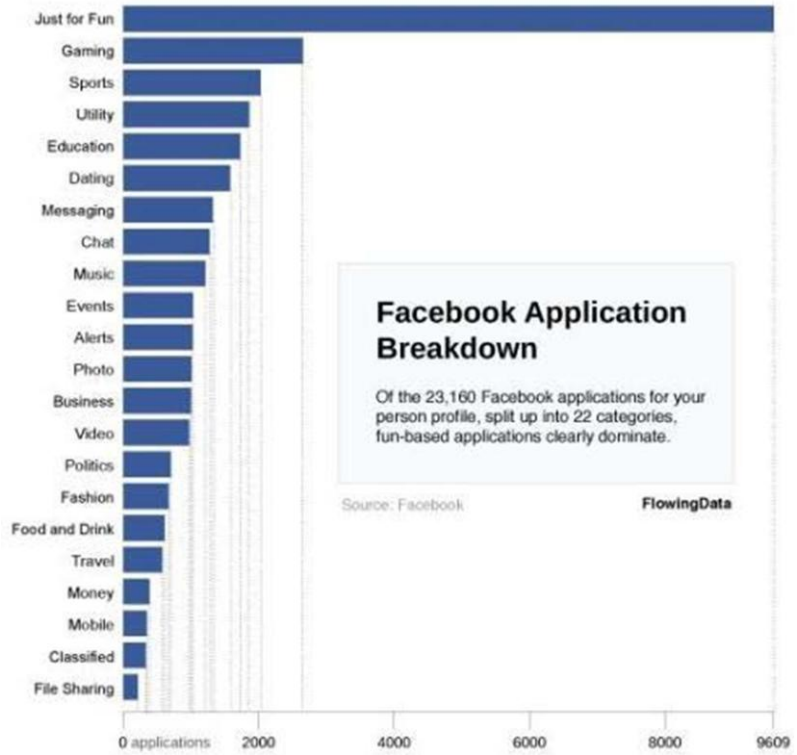

Fig 1.1 Number of applications on Facebook taken from [2]

1.1 Problem and Motivation

Genealogical data can be shown in different ways, such as ancestor charts, family trees or family relation with your friends in a social networking website, like Facebook. In family trees, the oldest generations usually appear at the top and the youngest ones at the bottom. On the contrary, the youngest generations are placed at the top in ancestor charts, which usually resemble more a real tree.

With the expansion of different social networking websites such as Facebook, a lot of family members are getting "reconnected" on a virtual level. We could use the data provided by these websites and their users to build up and visualize genealogical data. The interest in data visualization is always important for companies, which have strong hierarchies of their employees. Furthermore, this topic has given me the opportunity to investigate and develop an application like that, has been really challenging and motivating.

However, one of the main problems is that all the families are not perfect. For instance, there are some divorces, new weddings. Then, how can we represent this? We must review the structure of the tree to represent these events and new relationships among their members.

1.2 Goals and Criteria

This section describes the goals pursued by this thesis in order to solve the problem and the criteria used for reach the goal:

- The first and most important aim is to solve the problem to represent the genealogical tree. In order to get that, the interface must have look and feel. In addition, it must be easy to be used.
- The second goal is to show the events of people in the tree. For example, weddings, divorces, deaths...
- The third goal is to be able to add, edit and show people's information in the tree. In addition, this application must try to connect with Facebook in order to take the information from this social network.
- The last goal is to have the possibility to open the stored information of the genealogical tree and to save the data from the current data tree.

The development of this application could be divided into four parts. The first one is based on searching and choosing the platform to build up the application. There are different and interesting languages and platforms to start to work. Nevertheless, it is important to choose a good platform. Among all of them, we should work with the one that it is easy to implement, easy to be visualized by the user and compatible with different operating systems or browsers.

The second and main part of the project consists of the implementation of genealogical data in the family tree with a nice visualization. Then, the user must understand it easily. If it is possible, it would be great to connect with a social network like Facebook to take the information.

In the following part, a panel showing the events of all the people must be implemented. Of course, the information of this panel must be taken from the tree.

The last part is important for the application, which must be able to save and open the trees with their people and all the people's data.

In conclusion, an application that will build up a genealogical tree must show the data in a nice visualization. It should also give the option to save the current tree or open stored one. Finally, a left panel should show the event data related to the family members.

1.3 Report Structure

The present project comprises mainly six points. In the current one, the introduction is being carried out.

In the second Chapter, I will try to explain the studies and steps which had been done before I started with the implementation of the system.

Thirdly, different points of view to implement the system and to represent the data in visualization will be shown.

In Chapter four the explanation of the requirements of the application will be mentioned. After that, there will be an explanation of the way how the different parts of the application have been built. These parts refer to: the main panel where the tree is represented, the left panel with the representation of the events dynamically, the way to store and use the data of the tree, the way to show the information of each person in the application and how the system connects with a social network in order to get people's information.

The implementation details are introduced in Chapter five. Finally, the conclusion of the thesis and some future works are listed in Chapter six.

2 RELATED WORK

In this chapter, I am going to explain the different related work of data and the genealogy visualization which I based to start the project on. Moreover, some examples about the ways to visualize the data and other works of genealogy data will be given.

2.1 History

American genealogy traditionally dates from 1771, when Luke Stebbins published an account of his New England family.3 America was on the verge of a Revolution that would upend politics and undercut the respect for ancestors that had strengthened every society since Biblical days. Postwar, ancestral matters became not just politically incorrect but suspect. To many, genealogy smacked of elitism. In 1783 former Continental Army officers sparked a national controversy by organizing a society with hereditary rights, the Society of the Cincinnati— prompting fears that it would breed a new ruling dynasty. Amid social and political paranoia of the Early Republic, even Americans like George Washington understood the wisdom of camouflaging their own curiosity about their ancestral past.

Post–Civil War America was consumed by the ideology of race in its broadest sense. Hereditary organizations sprang up everywhere for those who could prove descent from this group or that. Once peace and prosperity returned, the nation attracted unprecedented waves of immigrants (particularly Catholics from Eastern and Southern Europe) and nativism spread like a pox. Many "old American families" of Protestant, Northern European stock reacted with hostility toward peoples who had not created America but who were arriving on its shores expecting to share in its greatness.

Genealogy became a tool of ideologies and prejudices rooted in concepts of blood, heredity, race, and stock. Genealogical organizations, including NGS, echoed those ideas. The first issue of the NGS Quarterly praised the (Northern European) "Blood that Made the Sturdy Races of New Netherland." [30] That same year, the society's head, a physician, focused his presidential address on "The Problems That Now Confront Us"—specifically, the "degeneracy and decay of modern society" and the "negative" influence of immigrants. He argued that solutions to these "problems" lay in wise reproductive choices made possible by the new "sciences" of genealogy and eugenics. Similar comments appeared in other genealogical journals.

The eugenics to which he referred was a new pseudo-science embraced by most western nations. Founded by Charles Darwin's cousin Sir Francis Galton, eugenics defined itself as the "science" of improving the human race by controlling reproduction. Naturally, the movement fed on genealogy; Galton even offered prizes for the biggest compilations of family data. Some within the new academic history held comparable views. However, that profession's insistence upon objectivity constrained their influence, while genealogy remained a foil for pride based upon genetic heritage—a pride valued more than objectivity or truth.

Adolph Hitler's atrocities committed in the name of race and blood discredited eugenics, but ancestral study continued to be equated with personal edification and amusement rather than serious study. The American Antiquarian Society's annual reports show how far historians and research facilities went to distance themselves from genealogy during the mid-twentieth century. Founded in 1812, the society had been the first American library to place priority on family history.

For more than a century its mission was unchanged. By 1953, it boasted one of the

nation's top three collections of genealogies—but added that the society did not encourage "genealogical investigation" when a researcher was interested only in his own family ancestry, although it realizes that such research is of much entertainment.

The 1960 annual report showed even more disdain: "For many years we took all genealogical serials but we dropped many of them as potboilers of no utility to the historians."[31, 32]

2.2 Related Visualization Work

Trees are used to visualize and model different hierarchical data sets. There are various cases where tree visualization are used to represent these hierarchies. For instance, visualizing a company organizational tree to show the positions of the employees in a company or the hierarchies of car companies. Another example is visualization of a tree for different kind of beers. We can see these examples in the figures 2.1 and 2.2

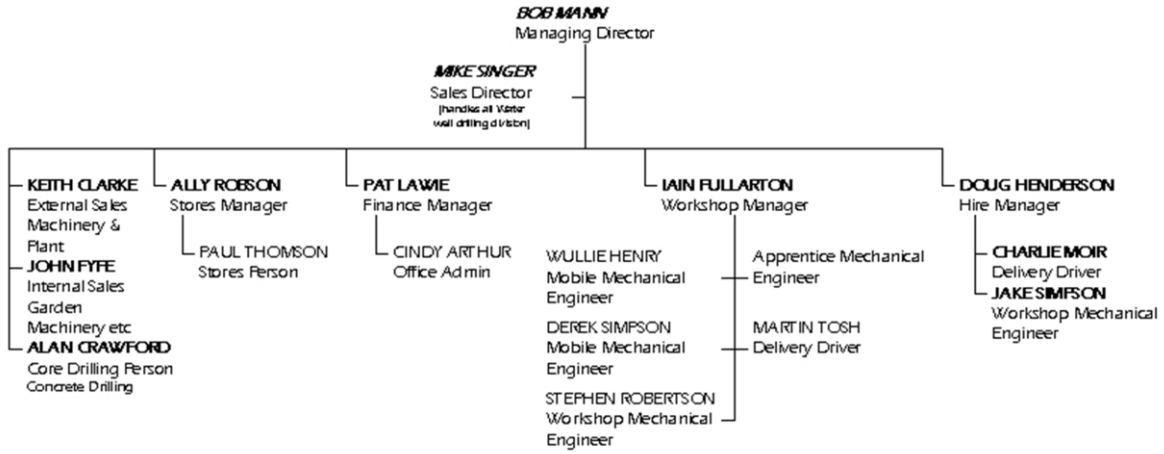

Fig 2.1 Tree of a company taken from [14]

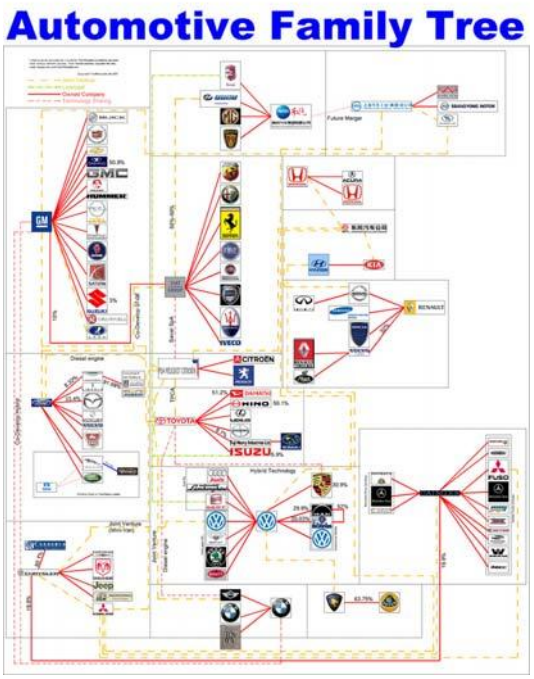

Fig 2.2 Tree of car companies taken from [15]

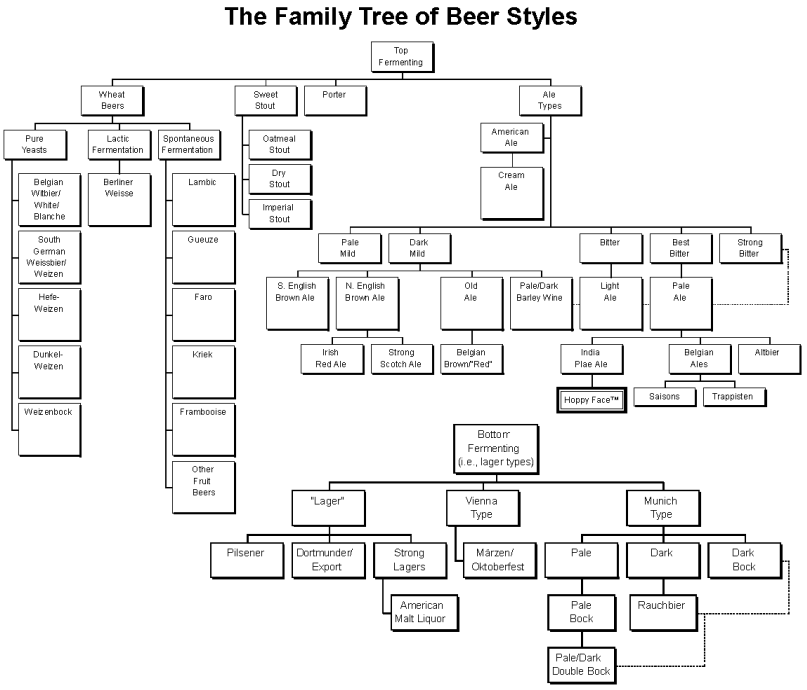

Fig 2.3 Tree of kind of beers taken from [16]

There are some kinds of trees to see the data. Depending on what we want to represent or on the kind of information that we must visualize, we should choose one of the following ones:

The Classical Hierarchical Tree View (Figure 2.4): Children are placed "below" their common ancestor. One of the main disadvantages of this kind of tree is the unused space. If the size of the tree is small, there is no problem, but when its size is big, it is difficult to visualize the full tree because of the unused space.

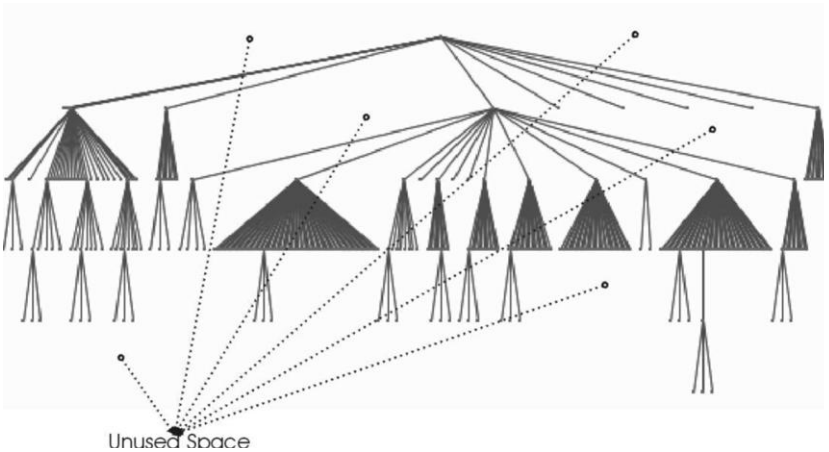

Fig 2.4 Classical Hierarchical Tree View taken from [28]

The H-Tree Layout (Figure 2.5): It is the classical drawing technique for representing binary trees. In our project would be impossible to choose this type of tree because the first person to appear would be the youngest one and the last one the eldest. In addition, in this project we need to cover the possibility to start from the eldest ones to the youngest ones.

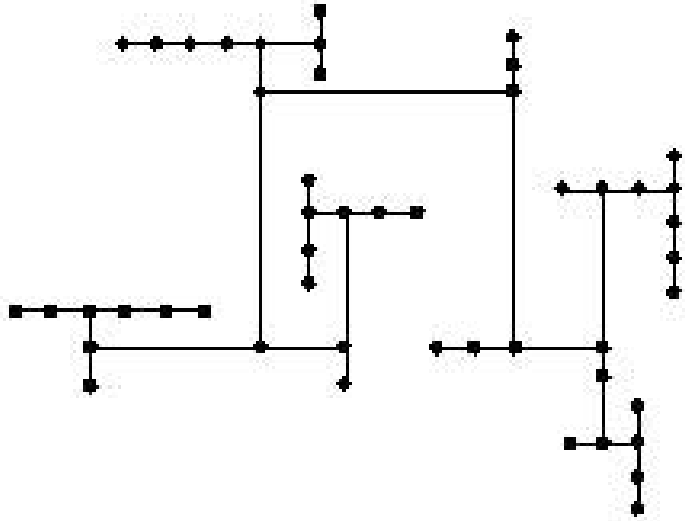

Fig 2.5 The H-Tree Layout taken from [28]

The Radial View (Figure 2.6): The radial view is based on an algorithm described in Eades [5]. Good be perfect for trees with a lot of children. Nevertheless, in case it has too many levels, the problem of unused space could appear again.

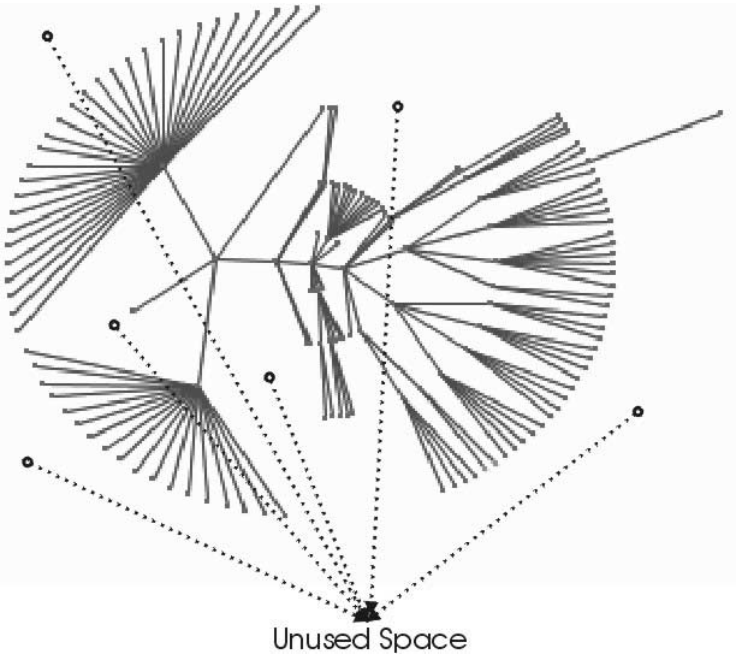

Fig 2.6 The Radial View taken from [28]

The Balloon View (Figure 2.7): The balloon view can also be obtained by projecting a cone tree onto a plane. Once more, the trees with a big size will have the problem with unused space.

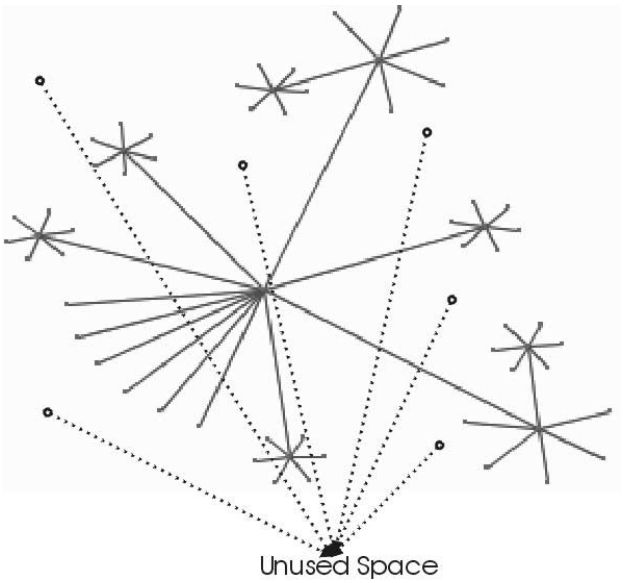

Fig 2.7 the Balloon View taken from [28]

The Hyperbolic Browser (Figure 2.8): This layout technique builds trees in a hyperbolic plane and then maps which are structured in ordinary Euclidean plane. As I have mentioned above, there might be a problem with the unused space if the size of the tree is too big.

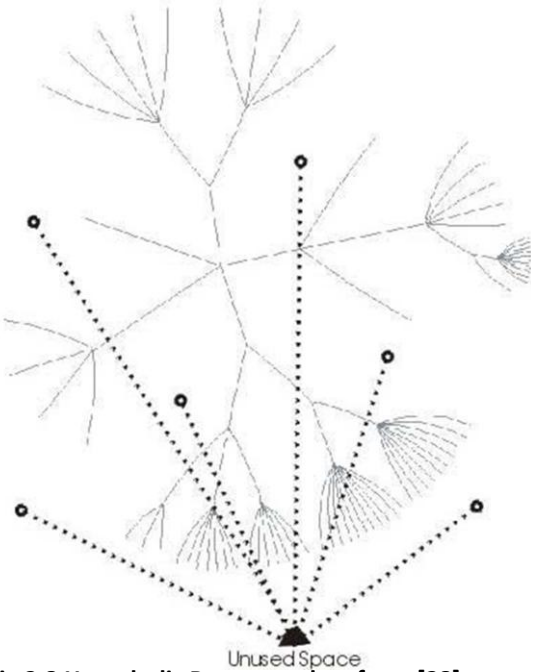

Fig 2.8 Hyperbolic Browser taken from [28]

The Tree Map (Figure 2.9): Tree maps display hierarchical (tree-structured) data as a set of nested rectangles. Each branch of the tree is given a rectangle, which is then tiled with smaller rectangles representing sub-branches. A leaf node's rectangle has an area proportional to a specified dimension on the data. (In the illustration, this is proportional to a waiting time). Often the leaf nodes are colored to show a separate dimension of the data.

When the color and size dimensions are correlated in some way with the tree structure, one can often easily see patterns that would be difficult to spot in other ways. A second advantage of tree maps is that, by construction, they make efficient use of space. As a result, they can legibly display thousands of items on the screen simultaneously [29].

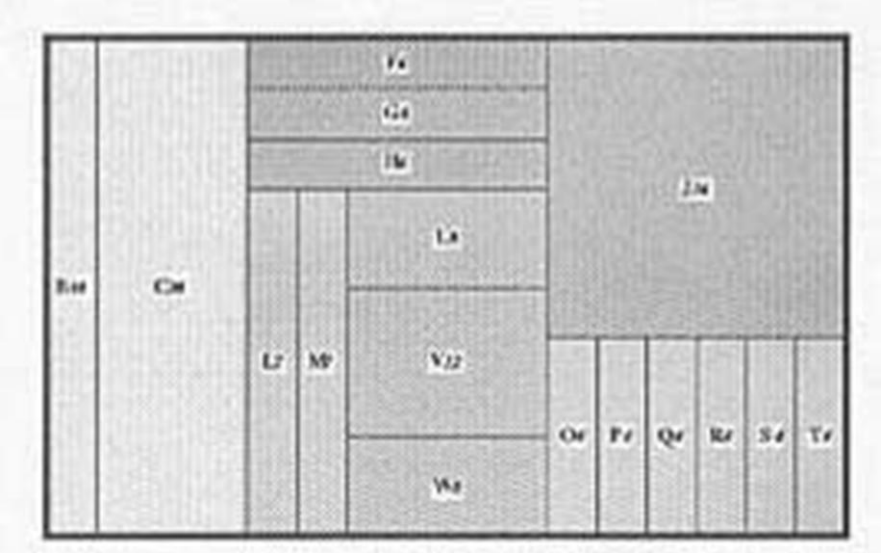

Fig 2.9 Tree Maps taken from [28]

2.3 Representation of Genealogies.

People collect genealogical data for several different reasons or purposes:

- To research on different cultures in history, sociology and anthropology (White et al., 1999), where kinship is taken as a fundamental social relation.
- To make genealogies of families and/or territorial units,
  - Mormons genealogy (MyFamily.com, 2004)
  - Genealogy of American presidents (Tompsett, 1993)
- For special genealogies
  - Students and their PHD thesis advisors:
- Theoretical Computer Science Genealogy (Johnson and Parberry, 1993)
  - Gods (antique).

There are also many programs for genealogical data entry and maintenance

(GIM, Brother's Keeper, Family Tree Maker...), but only few analyses can be done using these programs.

GEDCOM is a standard one for storing genealogical data, which is used to exchange and combine data from different programs, which have been used for entering the data. The following lines are extracted from the GEDCOM file of European Royal families.

```
0 HEAD
1 FILE ROYALS.GED
...
0 @I58@ INDI
1 NAME Charles Philip
Arthur/Windsor/
1 TITL Prince
1 SEX M
1 BIRT
2 DATE 14 NOV 1948
2 PLAC Buckingham Palace, London
1 CHR
2 DATE 15 DEC 1948
2 PLAC Buckingham Palace, Music
Room
1 FAMS @F16@
1 FAMC @F14@
...
...
0 @I65@ INDI
1 NAME Diana Frances /Spencer/
1 TITL Lady
1 SEX F
1 BIRT
2 DATE 1 JUL 1961
2 PLAC Park House, Sandringham
1 CHR
2 PLAC Sandringham, Church
1 FAMS @F16@
1 FAMC @F78@
...
...
0 @I115@ INDI
1 NAME William Arthur
Philip/Windsor/
1 TITL Prince
1 SEX M
1 BIRT
2 DATE 21 JUN 1982
2 PLAC St.Mary's Hospital,
Paddington
1 CHR
2 DATE 4 AUG 1982
2 PLAC Music Room, Buckingham
Palace
1 FAMC @F16@
...
0 @I116@ INDI
1 NAME Henry Charles
Albert/Windsor/
1 TITL Prince
1 SEX M
1 BIRT
2 DATE 15 SEP 1984
2 PLAC St.Mary's Hosp.,
Paddington
1 FAMC @F16@
...
0 @F16@ FAM
1 HUSB @I58@
1 WIFE @I65@
1 CHIL @I115@
1 CHIL @I116@
1 DIV N
1 MARR
2 DATE 29 JUL 1981
2 PLAC St.Paul's Cathedral,
London
```

Fig 2.10 GedCom File taken from [33]

From data represented in the described way (Fig 2.10), we can generate several graphs which are explained in the following paragraphs.

Genealogies can be represented as networks in different ways: as Ore-graph, as p-graph, and as bipartite p-graph.

In an Ore graph of genealogy every person is represented by a vertex. One the one hand marriages are represented with edges. On the other hand the relation between parent and children are shown with arcs pointing from the parents to their children.

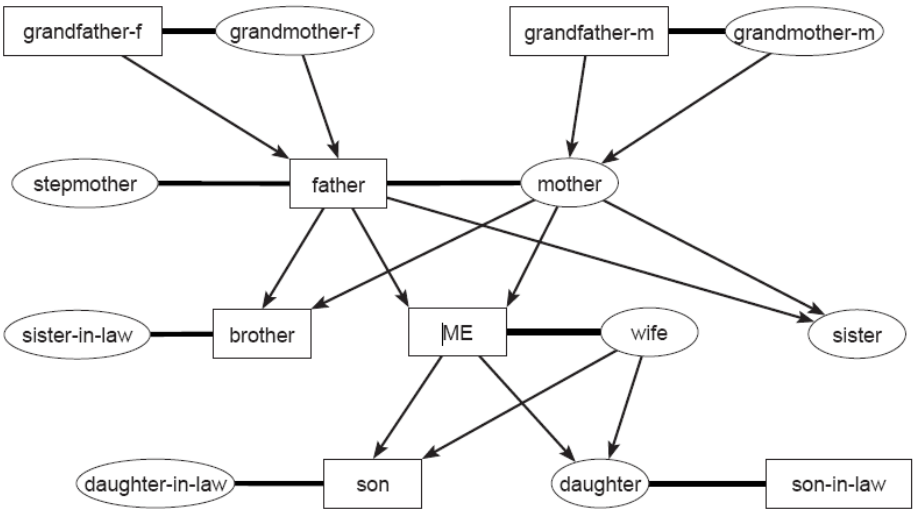

Fig 2.11 Ore Graph taken from [33]

In a P-graph vertices represent individuals or couples. In case that person is not married, they are represented by a vertex; otherwise the person is represented with the partner in a common vertex. There are only arcs in P-graphs – they point from children to their parents (Figure 2.12). The solid arcs represent the relation with a son of and the dotted arcs represent relation with a daughter of.

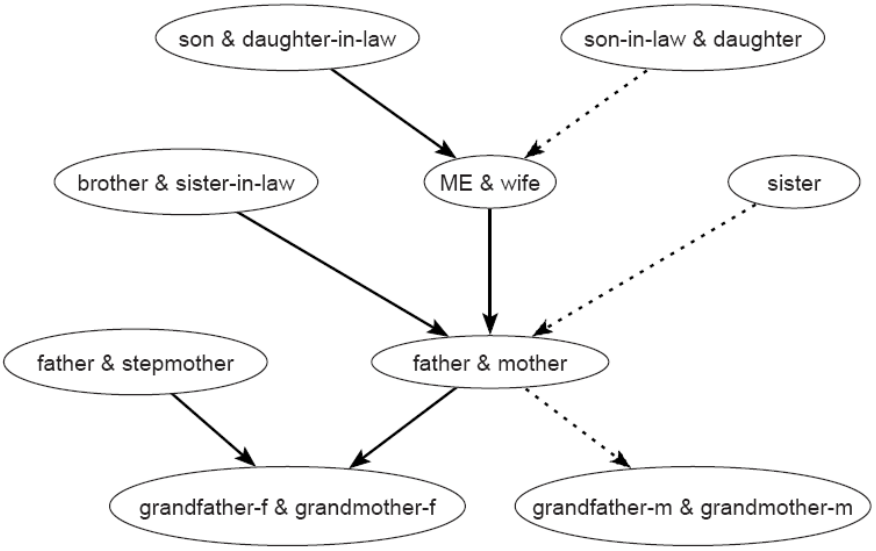

Fig 2.12 P-Graph taken from [33]

A Bipartite p-graph has two kinds of vertices – vertices representing couples (rectangles) and vertices representing individuals (circles for women and triangles for men) therefore each married person is involved in two kinds of vertices (or even more if he/she is involved in multiple marriages). Arcs again point from children to their parents.

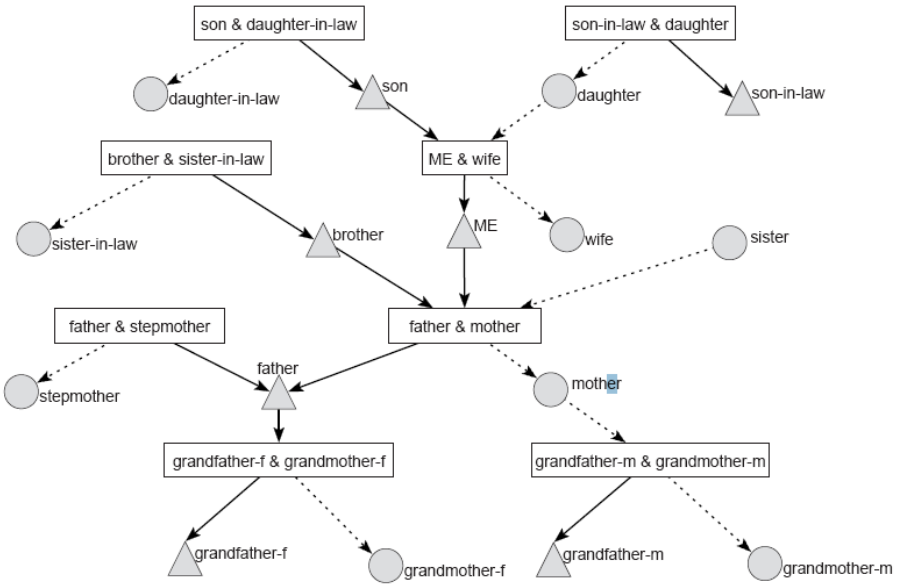

Fig 2.13 Bipartite p-Graph taken from [33]

P-graphs and bipartite p-graphs have many advantages:

• There are less vertices and lines in p-graphs than in corresponding Ore graphs;

• P-graphs are directed, acyclic networks;

• Every semi-cycle of the p-graph corresponds to a relinking marriage. There are two types of relinking marriages:

– Blood marriage: e.g., marriage among brother and sister.

– Non-blood marriage: e.g., two brothers marry two sisters from another family.

• P-graphs are more suitable for analyses.

Furthermore, bipartite p-graphs have an additional advantage. It is the distinction between a married uncle and a remarriage of a father. This property enables us, for example, to find marriages between half-brothers and half-sisters. [33]

2.4 New Visualization Techniques

Genealogy, i.e., the study of family relationships, is an increasingly popular activity pursued by millions of people, ranging from hobbyists to professional researchers. This is reflected in the large number of commercial and free genealogical software packages available nowadays. While most of these packages offer excellent support for building and maintaining genealogical databases, their support for visualizing these databases is quite weak.

The most widespread visualizations are based on node-link diagrams, which have been shown too quickly. As a result, they become unreadable as the graph size grows.

In fact, the users want to see interesting information without much effort. Because of that, it must be necessary to have an understandable and easy skin of visualization.

To start with a visualization tool about trees, I found the first one called Prefuse, a library of Java (explained in chapter 5.2). This visualization uses a kind of classical hierarchical view, which has been explained before. As example of it, we can see the next picture:

t r e e v i e w

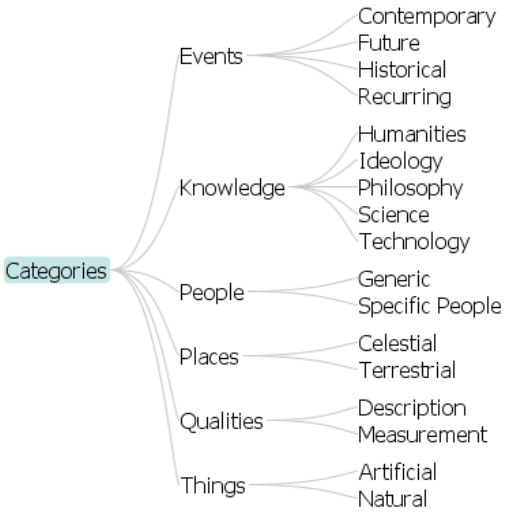

Fig 2.14 Tree graphically of a demo taken from [7]

The representation of family trees is becoming increasingly important. There are some current works and projects working on that. The need to show the information contained in a genealogic structure is doing that, more related works are starting and improving the oldest. Some examples of these works can be found out on internet.

The second important tool I found to work with trees is Flare. Flare is an ActionScript library for creating visualizations that run in the Adobe Flash Player. From basic charts and graphs to complex interactive graphics, the toolkit supports data management, visual encoding, animation, and interaction techniques. Even better, Flare features a modular design that lets developers create customized visualization techniques without having to reinvent the wheel. Flare is open-source software released under a BSD license, meaning it can be freely deployed and modified. Flare's design was adapted from its predecessor Prefuse, a visualization toolkit for Java. [25]

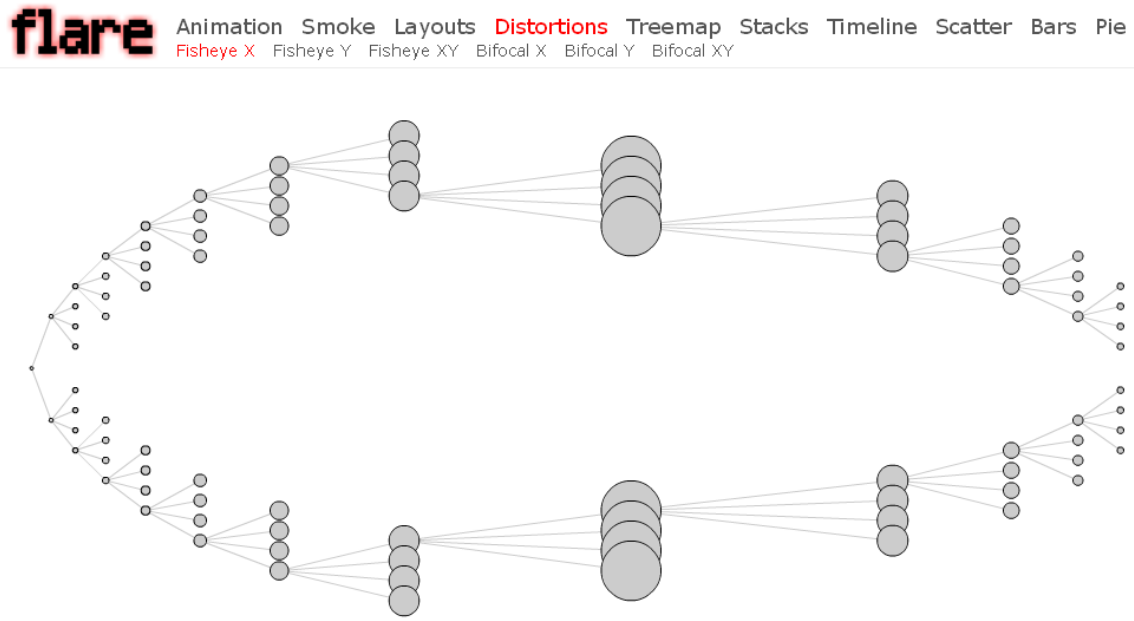

Fig 2.15 Example of Flare taken from [25]

2.5 Genealogy Visualization

As I have already said in the first chapter there are different forms to represent a family tree. On the one hand, the oldest generations could be placed at the top and the newest ones at the bottom. On the other hand, newest generations could be placed at the top and obviously, the oldest one at the bottom.

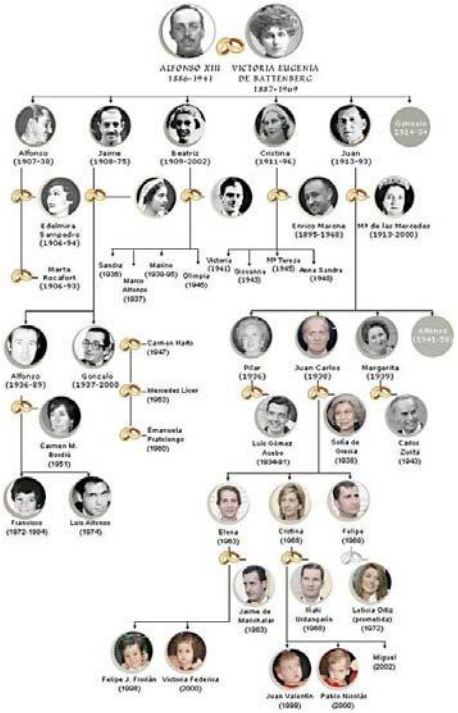

Fig 2.16 Tree with oldest generation at the top taken from [3]

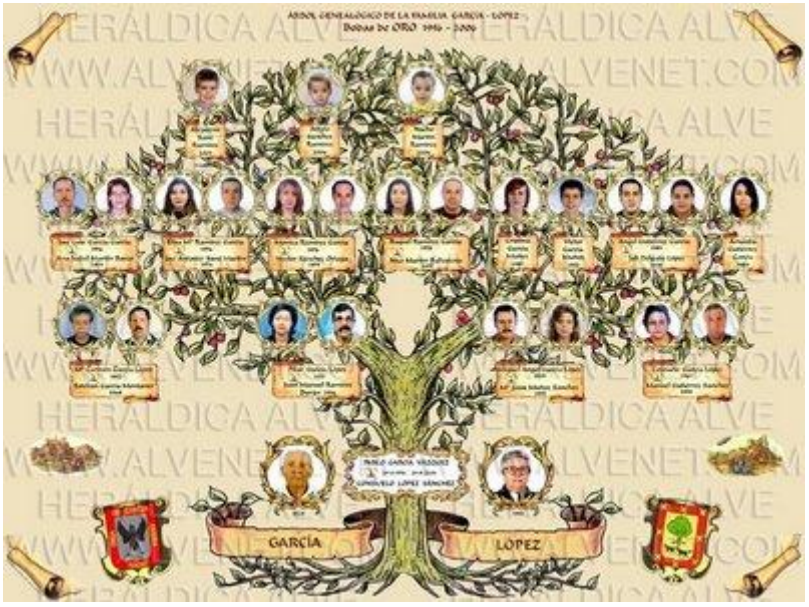

Fig 2.17 Tree with newer generations at the top taken from [6]

Considering the genealogical databases explained in section 2.3, those which are built by users individually can easily reach thousands of nodes. However, those which are built by organizations, such as companies, can reach tens of thousands of nodes. Thus, the need for a more scalable visualization solution is clear.

There is a solution based on a matrix representation, inspired by the Quilts [35] visualization for layered graphs. Quilts eliminate the confusing link crossings of node-link diagrams, and display layered graphs in a more compact manner than traditional matrix representations. The GeneaQuilts [35] technique maps rows to individuals and columns to nuclear families, effectively mapping groups of individuals from the same generation to alternating graph layers. I show how this approach allows us to benefit from all the advantages of the original Quilt technique while avoiding its drawbacks.

Different tasks are performed by genealogy researchers and enthusiasts. These tasks involve data collecting and recording, source documentation and analysis, and presentation. Although all these tasks are important, we focus on the initial analysis phase, where genealogists attempt to explore their datasets and quickly create or verify hypotheses.

We compiled a list of basic analysis tasks, collected from three extensive interviews with 8 users involved in genealogy research: 3 historians investigating transmission of land ownership and office titles across multiple families in France, 4 anthropologists interested in inter-marriage strategies within small populations/tribes worldwide, and a semi-professional genealogist who investigates family ancestry of individuals or families. We also included analysis tasks supported by commercial genealogy tools and research prototypes. Although our system does not currently support all the described tasks, their enumeration serves as a guide for genealogy visualization systems and for identifying future extensions to our work.

Since a genealogical graph is a special kind of graph, we build our taxonomy on the "Task Taxonomy for Graph Visualization". The graph objects become

genealogical entities: individuals, nuclear families, and paths (e.g., ancestors or descendants) in the genealogy graph or sub-graphs. The tasks are categorized as:

Topology-Based tasks where users need to identify global structures or patterns of interest in their data or among specific entities:

- Identify one's ancestors (pedigree) and/or descendants.
- Examine a nuclear family (identify parents, children).
- Identify one's extended family (aunts, uncles, nephews, nieces, cousins).
- Examine the nature of family relationships between two or more people in a genealogy. e.g., find if they are connected, if they have common ancestors, find all paths linking two individuals, examine if they are consanguine (by blood) or conjugal (by marriage) relatives, determine the nature of their connection.
- Find cases of inter-marriages between family members (both consanguine and conjugal relatives). These connections are often referred to as "rings" within families and may result in pedigree collapse (cases where married couples have common blood ancestors). The types of such inter-marriages are also important (e.g., between parental uncles and nieces, between maternal cousins, etc.) as well as their degree (how many generations are included in the ring).
- Identify complex family events, such as divorce, serial monogamy and polygamy, or marriages across generations (generation skipping or merging).
- Identify the main individuals in the genealogy (e.g., founder of a dynasty or of the largest lineage, or individual with largest number of children or marriages).

Attribute-Based tasks that involve the exploration of relations and attributes outside of blood and marriage connections:

- View detailed information on an entity's attribute, e.g., an individual's birth date, their location or the wedding's date.
- Organize important events for a family (e.g., births, deaths, marriages, etc.) in chronological order. This requires dealing with ambiguous or missing dates that is very common in genealogy data.
- Compare attributes of different individuals such as gender, status, etc. Of special interest is how attributes propagate within a family (e.g., inheritance, physical characteristics and genetic diseases) or across families. Commonly found examples include the succession of the title of patriarch within a family or the succession of a political office across families.
- Examine the evolution of numerical attributes across time and families. For example investigate how the dowry amount has evolved within a family or compare the division/distribution of inherited land between families across generations.
- Explore relationships outside blood and marriage, such as trading partners between families, foster children, family neighbors and friends, etc. For professional genealogists these relationships can also be crucial links for further research to expand their datasets.

For centuries, genealogical relationships have been illustrated in books with hand-crafted charts of a few dozen individuals. Genealogy software can now technically accommodate datasets of hundreds of thousands of individuals. Nevertheless, no software can visualize a large dataset in a legible way. So far, three types of approaches have been used for visualizing genealogies: node-based representations, line based representations and tabular representations.

For each person in a genealogy there is one tree of descendants and one of ancestors (pedigree). Most commercial software visualize these tree structures using traditional tree diagrams, or offer alternatives such as Fan charts (radial space filling diagrams) or hourglass charts (also called "centrifugal views"), drawing both descendant and ancestor trees. Hourglass charts have many similarities to the Zoomtree [35] interface. All of these visualizations break down quickly as the number of individuals grows, and fail even sooner when they depict not just consanguine trees (descendants and ancestors) but also the lattice formed by conjugal relationships (marriages). Visualizing a genealogical graph using a node-link diagram – either from an Ore-graph or from a bipartite graph – usually involves assigning a layer (i.e., a generation) to each individual and trying to minimize the crossings between layers.

Large genealogies exhibit very long edges and too many crossings to be suitable for exploration or presentation. Genealogy systems seldom implement these algorithms and usually resort to unpublished heuristics to layout the graphs, all of which break on special cases (e.g. cycles or multiple marriages on several generations). To solve the problem, they rely on hand-editing the layout, which is impractical for large genealogies.

Dual Trees [35], which are similar to Multi-Trees, extend the hourglass chart by offsetting and connecting roots of ancestor and descendant trees, with each root having an hourglass chart. This technique minimizes edge crossings but does not eliminate them, and it still only shows a limited number of nodes on screen. To address this, the authors have proposed interaction techniques for expanding or collapsing nodes and transitioning between subsets of the dual trees. In short, all node-based approaches have serious drawbacks: they do not scale well to large numbers of individuals, they cannot represent large family lattices, they fail to highlight complex relationships (such as polygamy), they do not show temporal attributes (like birth dates) and finally they fail to convey larger context and distant relationships.

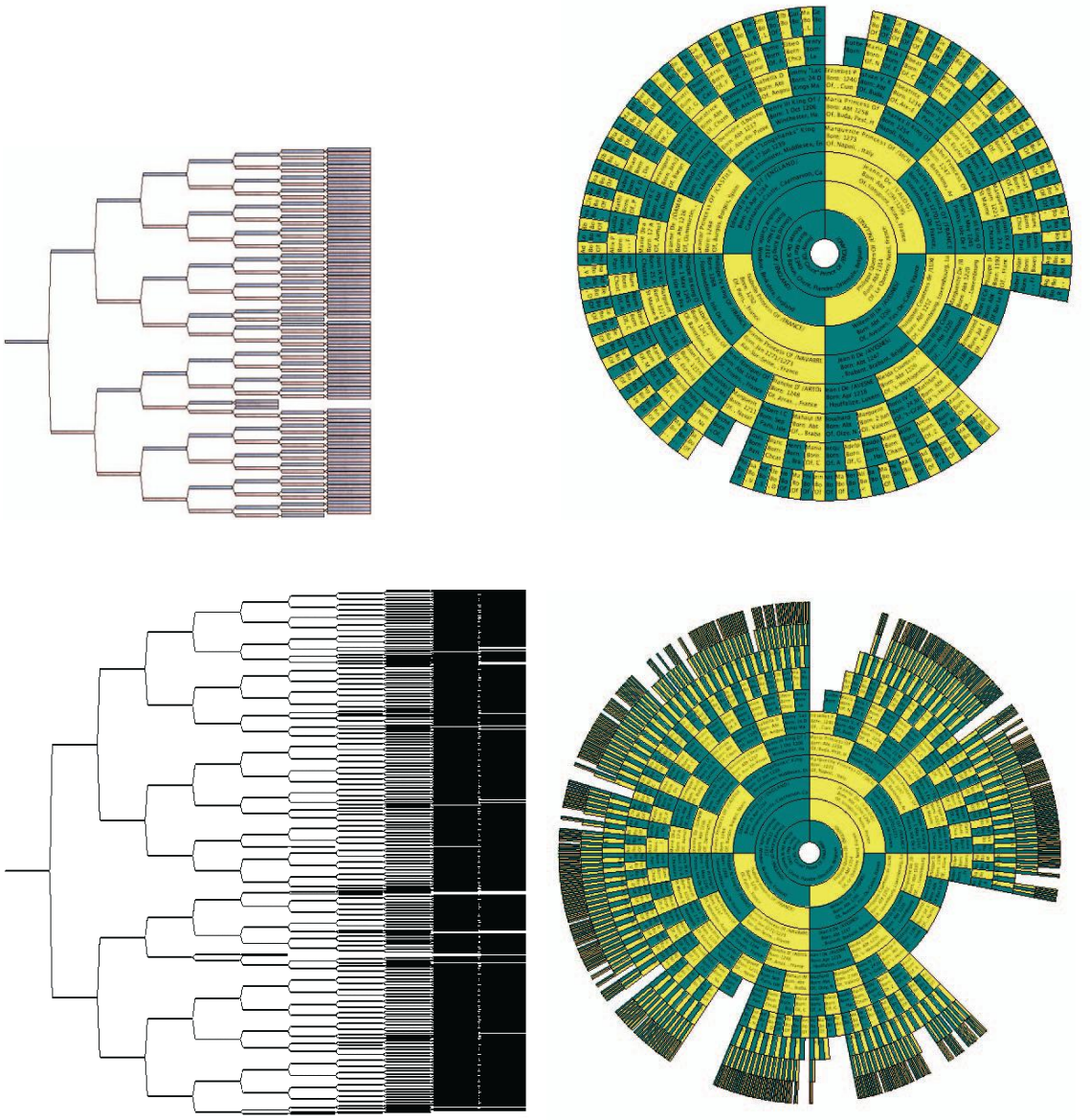

Fig 2.18 Comparing node-link diagram (left) and fan chart (right) when displaying eight (top) and eleven generations (bottom) taken from [35]

Another approach to genealogy visualization represents individuals as lines rather than nodes. For example we can present individuals as line segments and families as points. Each segment has two points, one connecting the individual to their parents, and the other to their children. But multiple marriages are difficult to depict: they require duplicating the lines representing the person for each marriage. P-graphs use a similar representation, with the person's gender indicated by the line orientation (vertical or tilted) and additional notes on the line segments indicate gender and patriarchal succession. P-graphs are often used for genealogy charts in anthropological literature, as the directions of the lines form interesting patterns when examining intermarriages within a family or clan.

Finally, most Genealogy systems provide extensive ways to navigate in large datasets by the means of tables: tables of individuals, tables of marriages, tables of places, etc. However, tables alone are poor at showing an overview of the relations between people and at supporting navigation and exploration. They need to be coupled with clear and scalable visualizations.

Some genealogy systems provide analysis tools, especially for the purpose of kinship analysis. Ethnographers study the strategies adopted by groups and build models of stable societies based on different kinship systems. Therefore, they develop tools to check their models in specific populations. The main characteristic of these models is based on marriage patterns and ring structures. A ring structure is a cycle in the non-oriented genealogy graph, closed by a marriage. For example, the Bible genealogy shows that Mary and Joseph have a common ancestor: King David. Therefore, there is a closed cycle starting at King David, splitting in two descendant lines, one reaching Joseph, the other one reaching Mary and closing at their marriage.

Researchers have advocated matrix-based representations as a scalable alternative to node-link representations. Recently, a variant called Quilts has been introduced. That one can represent layered and "quasi" layered graphs in a more compact way.

The Figure 2.19 illustrates the original **Quilts visualization**. The left image shows a node-link diagram of a directed graph where nodes have been assigned a layer (a row). Most edges run between successive layers.

The right image shows the corresponding quilt: nodes are laid out in a zigzagging pattern across the matrix diagonal, as opposed to being on the matrix's borders like in classical matrix representations. The nodes from the top layer (in blue) are laid out horizontally and the nodes from the second layer (in red) are laid out vertically. Links between the two layers are depicted as black dots, forming a sub-matrix. To the right of the second (red) layer are the sub-matrix depicting relationships between the second and the third layer (in green).

Problems arise when there are links between two non-successive layers, i.e., skip links. For example, it can be seen from the left image that two links go from the 1st to the 4th layer, and one from the 2nd to the 4th layer. Since not all skipped links can be displayed positionally (e.g. 2nd to 4th layer), Quilts appends skip-links to submatrices and uses a color-coding scheme to refer to distant nodes. In the picture for example, two colored dots have been added to the first (blue/red) submatrix to show links from the 1st (blue) to the 4th (purple) layer.

Another colored dot has been added to the right to depict the link from the 2nd (red) to the 4th (purple) layer. However, this solution is seriously limited, as it can be difficult or impossible to find the matching color of the destination node, especially in large graphs. [34]

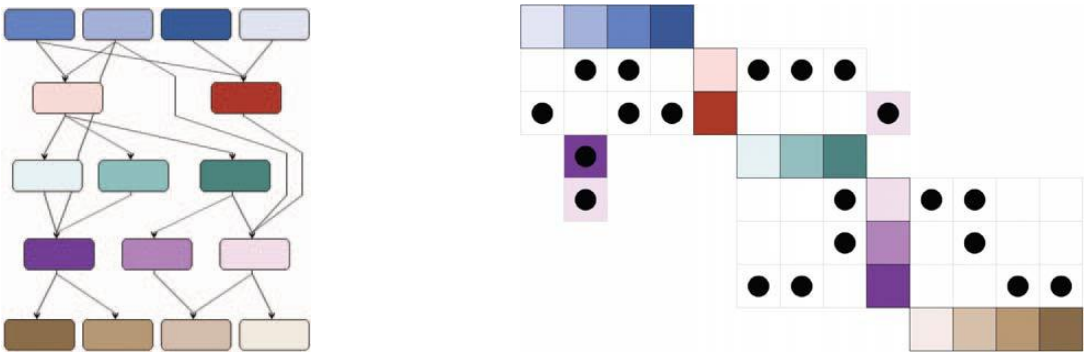

Fig 2.19 A node-link depiction of a small layered graph (left) and its quilt depiction (right) taken from [35]

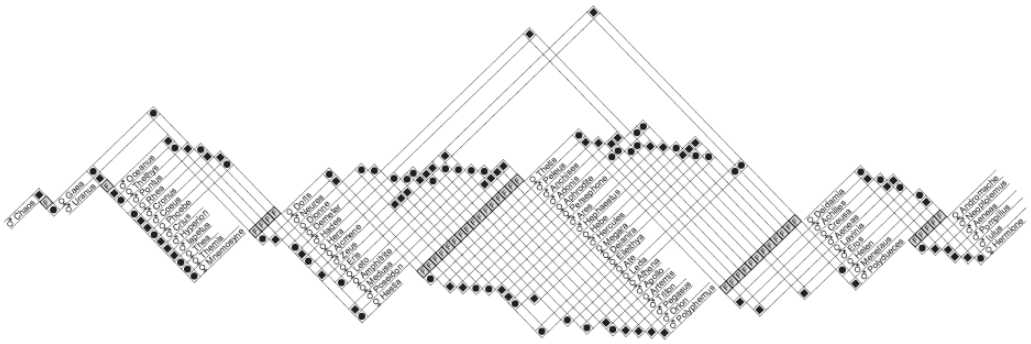

Fig 2.20 Quilts Visualization taken from [35]

TimeNets[36] is another new visualization technique for genealogical data. Most genealogical diagrams prioritize the display of generational relations. To enable analysis of families over time, TimeNets prioritize temporal relationships in addition to family structure. Individuals are represented using timelines that converge and diverge to indicate marriage and divorce; directional edges connect parents and children.

This representation both enables perception of temporal trends and provides a substrate for communicating nonhierarchical patterns such as divorce, remarriage, and plural marriage. We also apply degree-of-interest techniques to enable scalable, interactive exploration.

By depicting individuals as timelines which converge and diverge to depict marriage, TimeNets represent a number of real-world phenomena— including divorce, remarriage, plural marriage, and out-ofwedlock births—that are either difficult or impossible to represent using standard genealogical diagrams. By using degree-of-interest techniques, TimeNets also support scalable, interactive exploration. In a controlled experiment we found that TimeNets exhibited significant advantages over family tree diagrams for tasks involving temporal data: TimeNets accelerated task times 25% without diminishing accuracy. These results suggest that TimeNets could serve as a useful tool for genealogical researchers and hobbyists. [36]

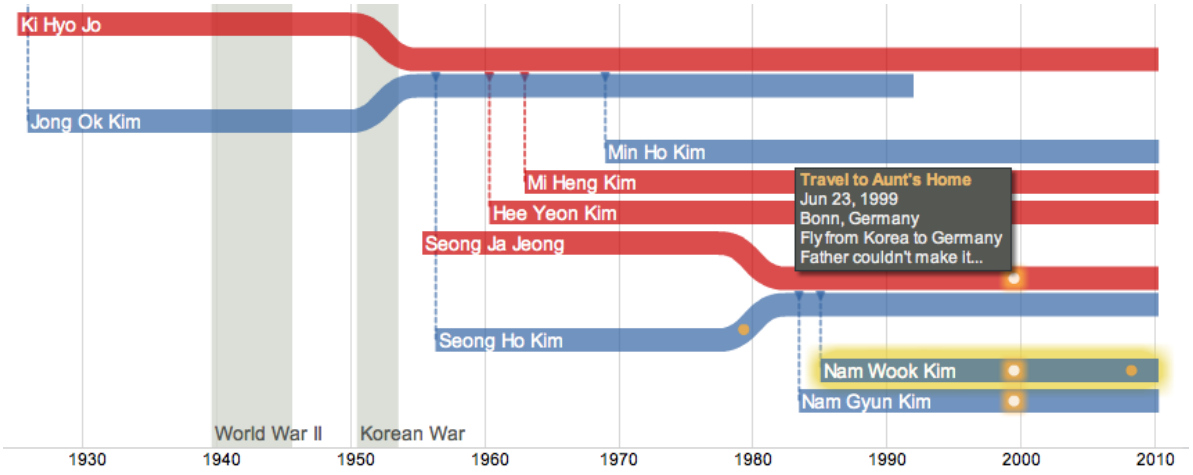

Fig 2.21 TimeNets Visualization taken from [36]

Some built applications I found on the browser to create specifically genealogical trees is **PhpGedView [23]** built with the PHP language. PhpGedView is a revolutionary genealogy program which allows you to view and edit your genealogy on your website. PhpGedView has full editing abilities, full privacy functions, and supports multimedia like photos and document images. PhpGedView also simplifies the process of collaborating with others working on your family lines. Your latest information is always on your website and available for others to see [23]. Of course, there are some webpages following the same steps (built with Php) to create genealogical trees.

Fig 2.22 Example of PhpGedView [23]

Very similar tool to create genealogical trees is **Myheritage.com.** MyHeritage.com was founded by a group of people with a passion for genealogy and a deep knowledge of Internet technology. It is not an open code, just create the tree. [24]

Fig 2.23 Example of myHeritage.com [24]

Finally, we can find **Gramps Genealogy System:** is a free software project for genealogy, offering a professional genealogy program, and a wiki open to all of them. It is a community project, created, developed and governed by genealogists. [22]

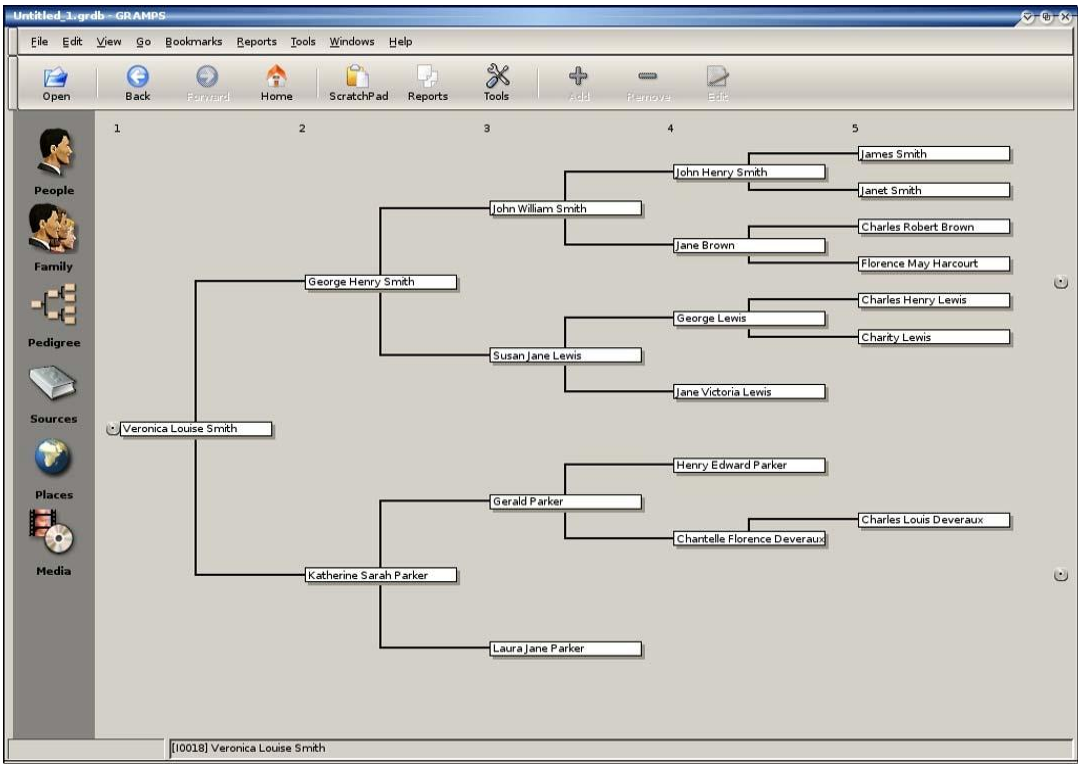

Fig 2.24 Example of Gramps Genealogy Tree taken from [22]

3 VISUALIZATION AND INTERACTION

This chapter gives an overview about the user interface of our prototype. I will explain the visualization and interaction about the main panel, the event panel and the windows to add, edit or show people. In addition, I will explain how the system is working among users and internally how the system saves the information which the user introduces.

3.1 User Interface

This section contains the explanation of the two panels that we are using in the system and the different views we have in the application, specially how they look like and how they work.

3.1.1 Overview

We will see an overview about how the application comprises different parts, which the user could find working with it. This will be explained more in detail in the following subsections.

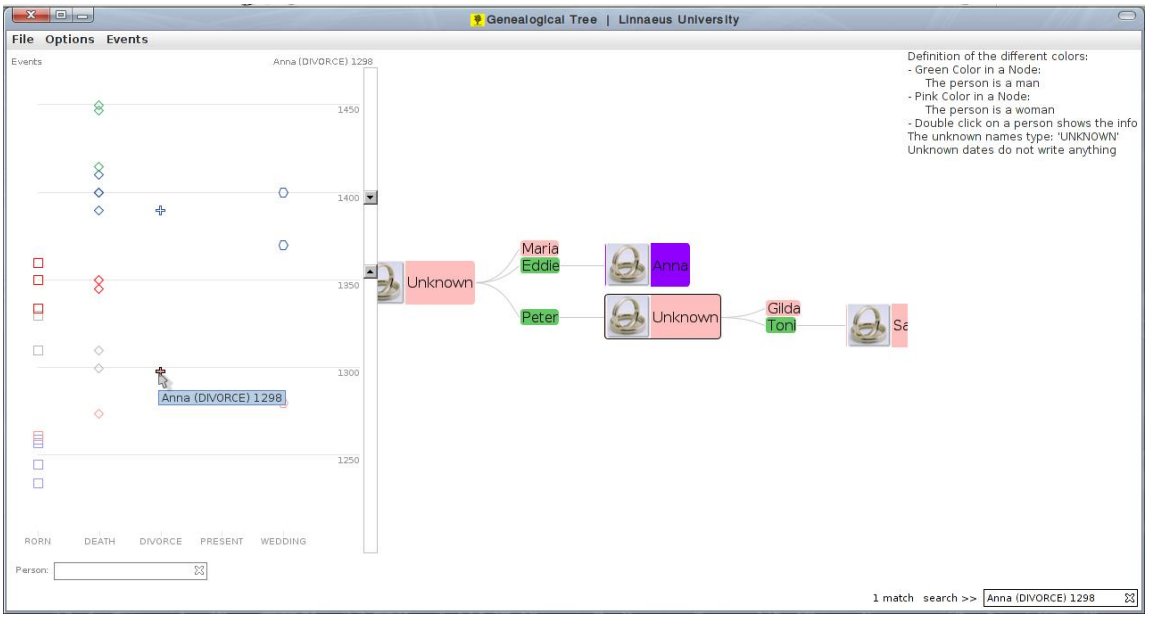

Fig 3.1 Application running with separated parts of the program

In the main window, which is the biggest one, we can find two different panels, one on the left with some symbols and one on the right with the tree. These are the event panel and the tree panel.

The other windows are for different activities. For example, to add a new person to the tree, to edit the information of the selected person or to show the information of this person.

Summarizing the process to build what the genealogical tree is, firstly the person must be added to the tree then, the wife and the children can be added with the

different options from the menu. The event panel is updated automatically when a person is added or modified. If the user wants to visualize the information of a person, must select the person and then using again the menu or double clicking, a new window will be opened with the information of that person.

Furthermore, the user can search some events by typing on the white box on the left corner. Then, the program will filter the events.

3.1.2 Event Panel

One important part of the system is the event panel placed on the left of the application. This panel depends directly on the other part of the system, the main panel which will be explained in the subsection 3.1.3.

This panel will show the information of the events. These events are taken from the data of the people. When a person is added or edited, the event panel is updated with the new information about the data of the person.

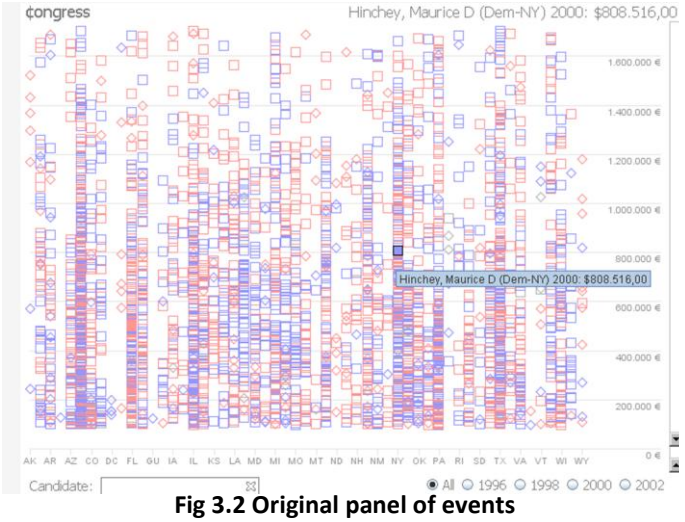

Fig 3.2 Original panel of events

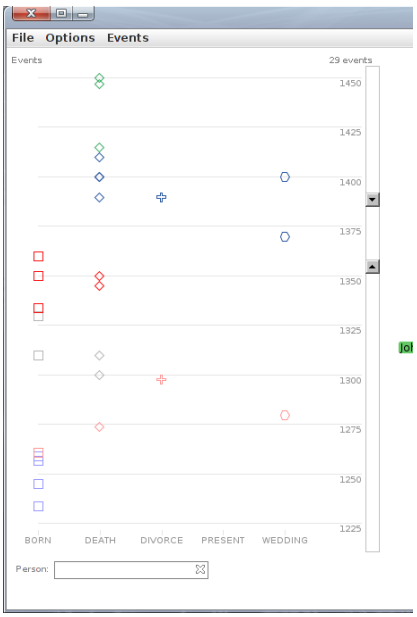

Fig 3.3 Current panel of events (running the application)

3.1.3 Tree Panel: One panel, two visualizations

This is the most important part of the system. The first thing I had to decide was the way to show the tree. To develop the tree I choose the way of representation. That is, the oldest generations are situated at the top and the newest generations at the bottom. In that way, the structure of a tree is easier to represent the oldest at the top and the newest at the bottom.

Nevertheless, the problem to represent the tree vertically is a problem of space. The screens usually have a bigger width than height (example, a laptop screen). Because of this, the tree will be shown from left to right (the oldest on the left and the youngest on the right), but following the same steps to build a tree from top to bottom.

In this panel the genealogical tree will be represented. This tree only will show the name of the person because the system works faster doing this (and later showing the information with an option). So, how can I distinguish the people? An implementation is done at least to have a differentiation between men and women. Why this? It is because the tree has an order to build the family. In addition, we can differentiate between husband and wife. That is the reason of the rings next to the women. Then rings only appear when the woman is the wife, to separate a female who is wife and female who is only child.

How to build the family? We can suppose there will be some divorces or some weddings "repeated" (after the divorce, or because the wife is death). So, thinking how to solve the problem to visualize this in the tree, the best idea I could find was creating the father. In fact, the wife will be considered a "child" of the father. After this, the son or daughter will be created from the wife. In this way, we can visualize easily who every child's mother is in case the father has had more than one wife.

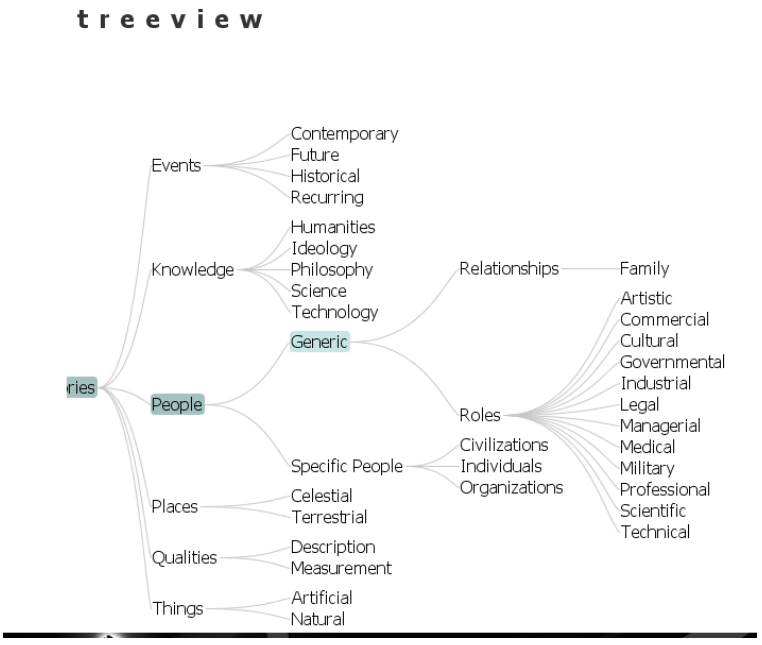

Fig 3.4 Original Tree

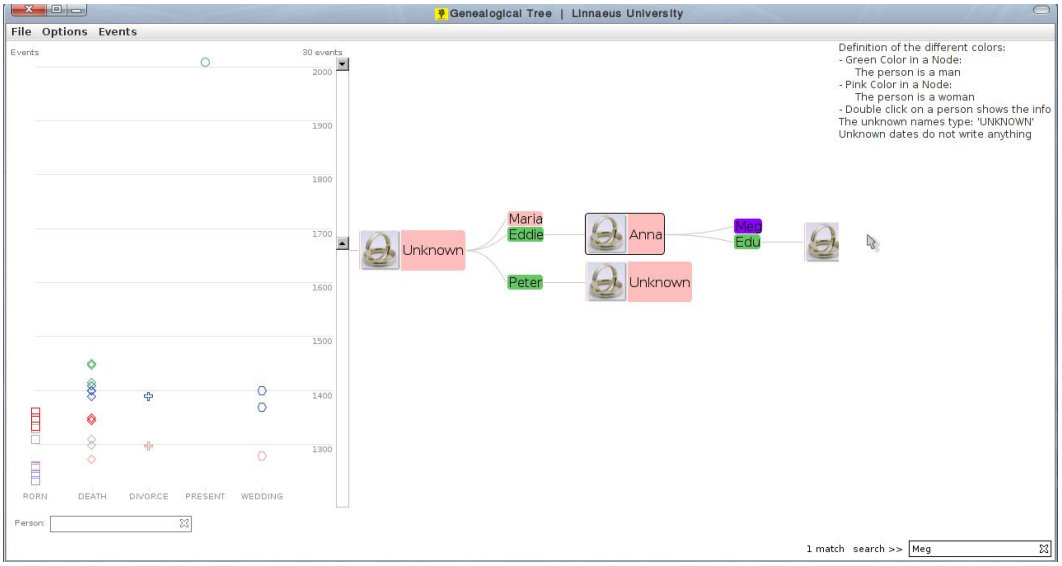

Fig 3.5 Genealogical Tree

The second visualization is based on the example of Prefuse DocuBurst. This visualization is better to represent an Ancestor Tree (from youngest to oldest) as we minimize the problem of unused space. This can be reduced because there is no space (no edges between people).

Why is the tree not created with the other visualization? It is because the user will find easily how ancestor is divided by father family on the one side and mother family on the other side.

Moreover, the problem to represent the whole family is solved taking into account that the other visualization is only possible following the male line of the family. In this case, it is possible to follow both lines, male and female.

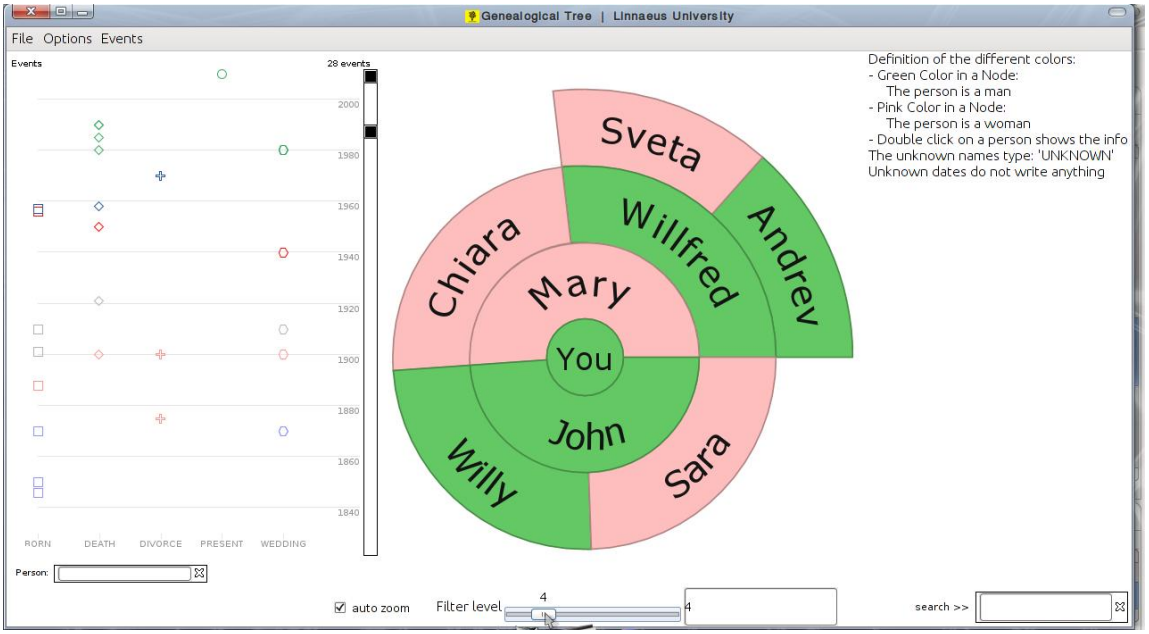

Fig 3.6 Ancestor Tree Visualization

3.1.4 Window Person: Add, edit and show.

The need to work with people in the tree forces us to create a part that is able to manage all these people.

The tree has nodes. Each node would have some children and each child more children. The question now is how I could add a person to a specific node or edit the information of the node that the user wanted. The user must click on the user who wants to change or add children and then choose the option in the options menu. Then, depending on the option, one window will appear to manage the person. And how is this working? We will see it in the following paragraphs.

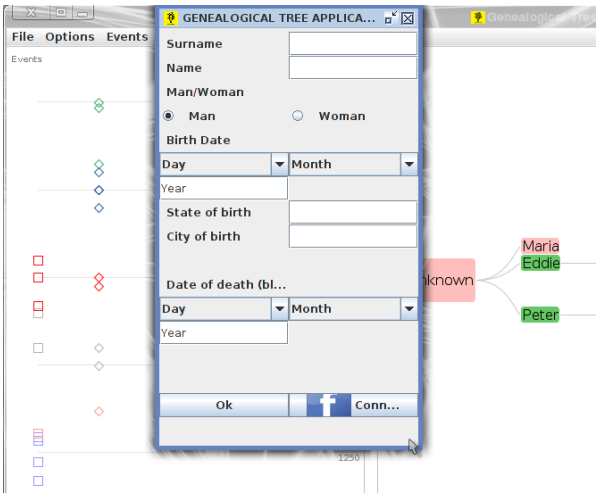

Fig 3.7 Add a child or the first node

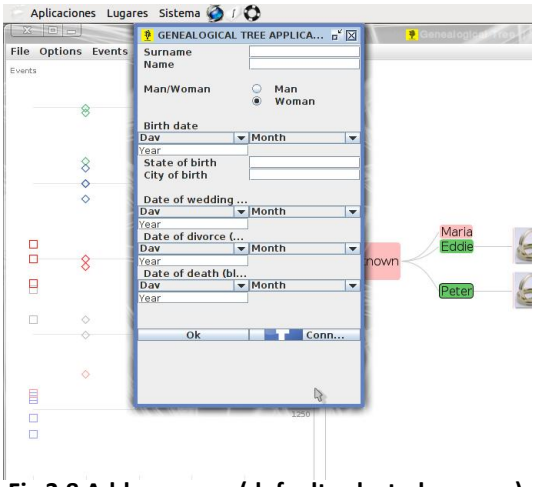

Fig 3.8 Add a spouse (default selected woman)

There are three options to add a person: add the first person of the tree, add a wife or add children. The windows show the same but internally are not the same. For example, when user adds a spouse, is allowed to add the wedding's date.

The next window shows the information of the person with the possibility to change the picture profile. I need this form because I do not draw in the tree this information as the system could be unstable and slowly. The information is

taken from the table which performs the tree we see in the main panel. When the user clicks on the person, the pointer goes directly to the row of the table where the person is.

To change the picture, the user just only needs to click on the default picture and choose the picture for the profile.

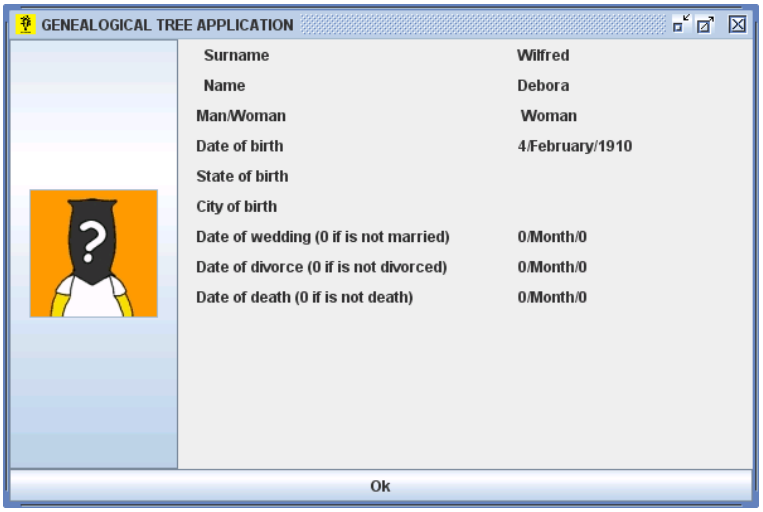

Fig 3.9 Show Information

3.1.5 Window Login Facebook

We do not have a servlet to do the connection between Facebook and our application but I implemented the window to connect with Facebook. This window just shows two fields: The username and the password. When the user clicks on connect, the system connects with Facebook through the public and the secret key, which will be explained below. Nonetheless, without a connection with the servlet, the information of the contacts from Facebook is impossible to get.

3.2 Interaction

One of the most important parts of the application is the interaction between the user and the tree. Remembering the topic of this thesis, the visualization becomes one of the most interesting and important issues to explain in this report.

The first interaction is how people are represented with different colors. The user can appreciate the different colors (green and pink) to have a first impression about the genre of the members of the family. Also, the rings next to a woman represent that is the spouse of the person next to her.

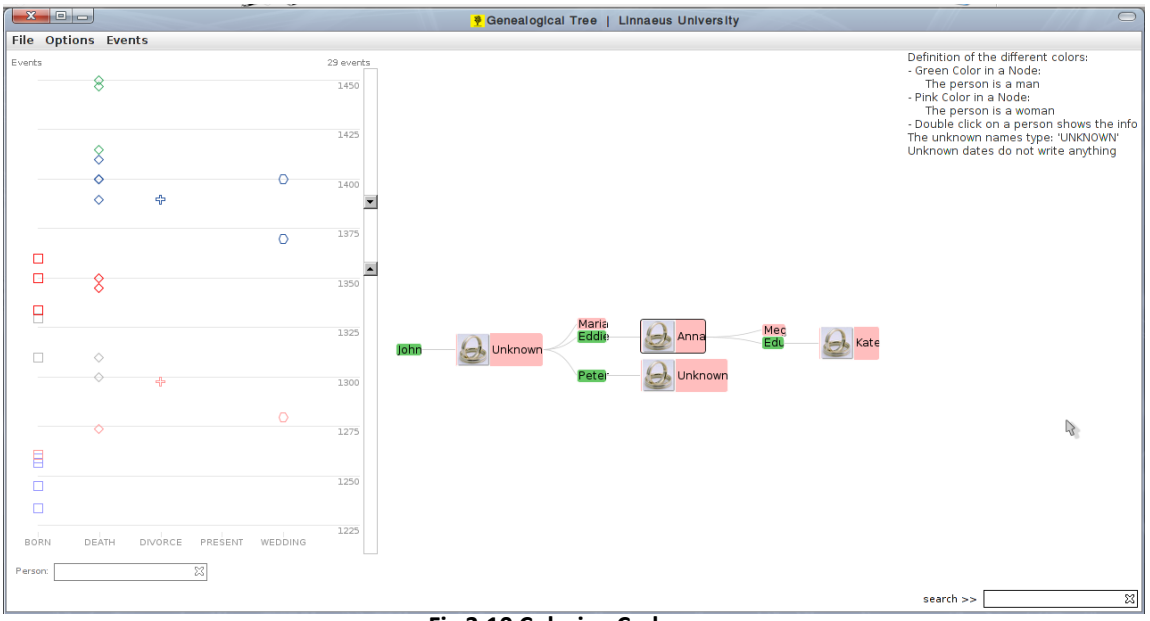

Fig 3.10 Coloring Code

After this, it is important to explain how the search is working. We can find two search fields. The user can find specific events of the people or specific people in the tree. As we can see in the Figure 3.11, in one case, a filter is applied to show the searched items and in the second case a new node coloring is added to the view:

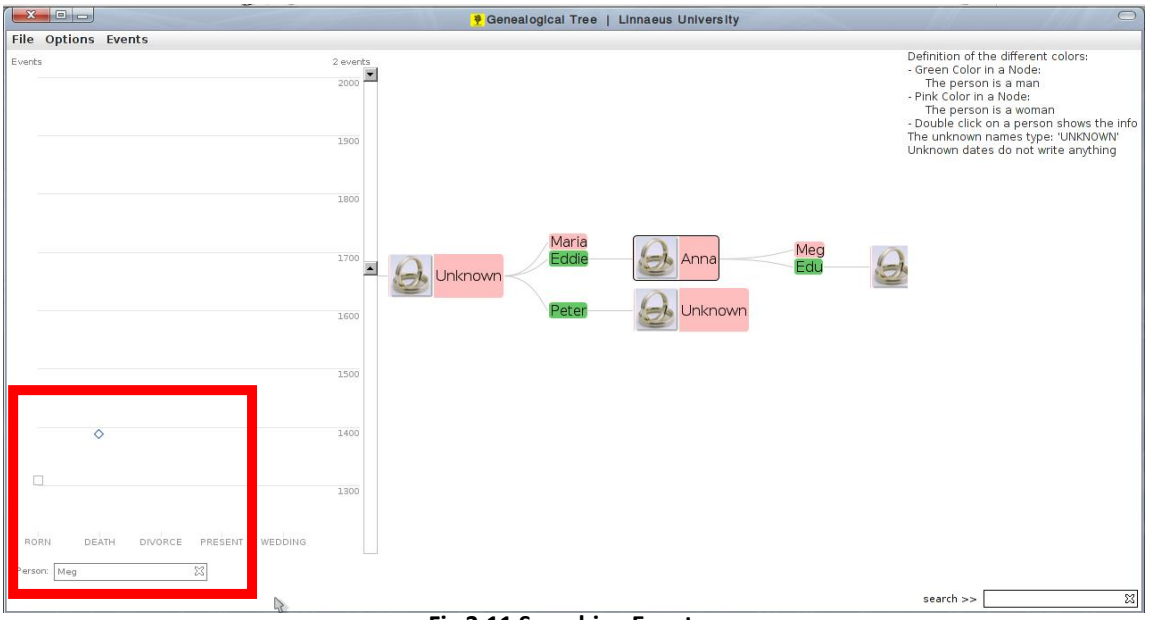

Fig 3.11 Searching Event

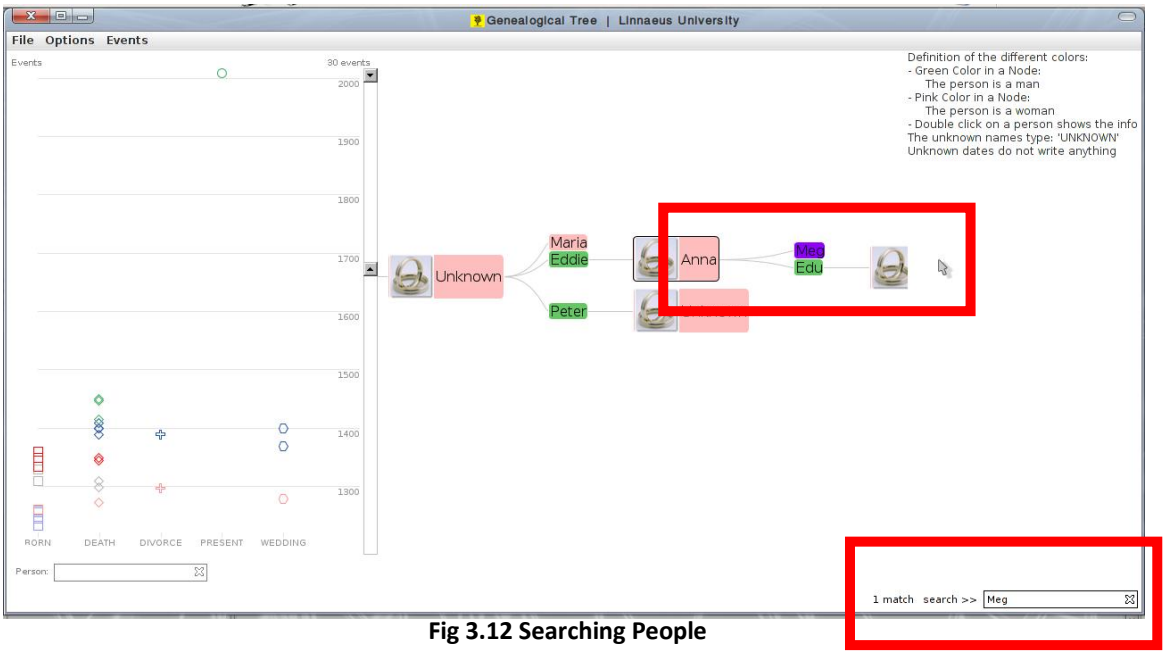

Fig 3.12 Searching People

The second interaction is between the event panel and tree panel. When the user does a mouseover on an event, the person involved in that event is highlighted in the tree like a search. The next picture shows an example of this:

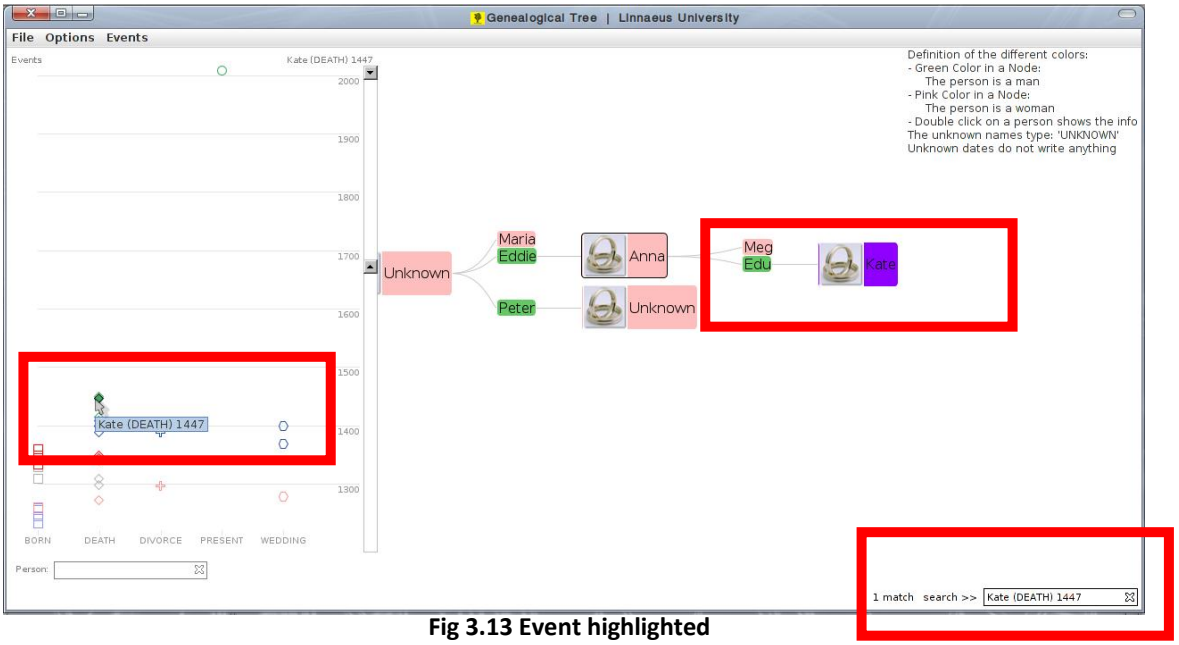

Fig 3.13 Event highlighted

The option of list of people is the same case. A person is selected on the list and this person is highlighted in the tree:

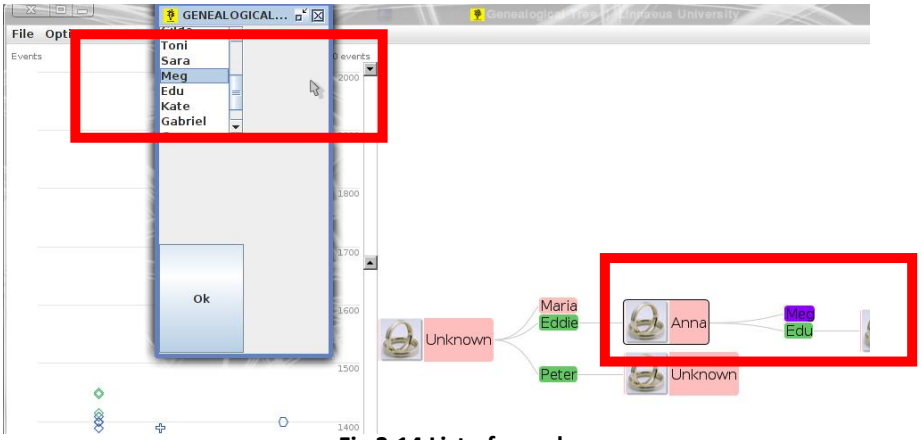

Fig 3.14 List of people

In the event panel we can find the Slider. The user can limit the range of the years to be showed. Moving the slider the panel filters the results depending of the years between the limits. In the last screenshots we can see “30 events” on the top. Now, after the filtering the number is “24 events”:

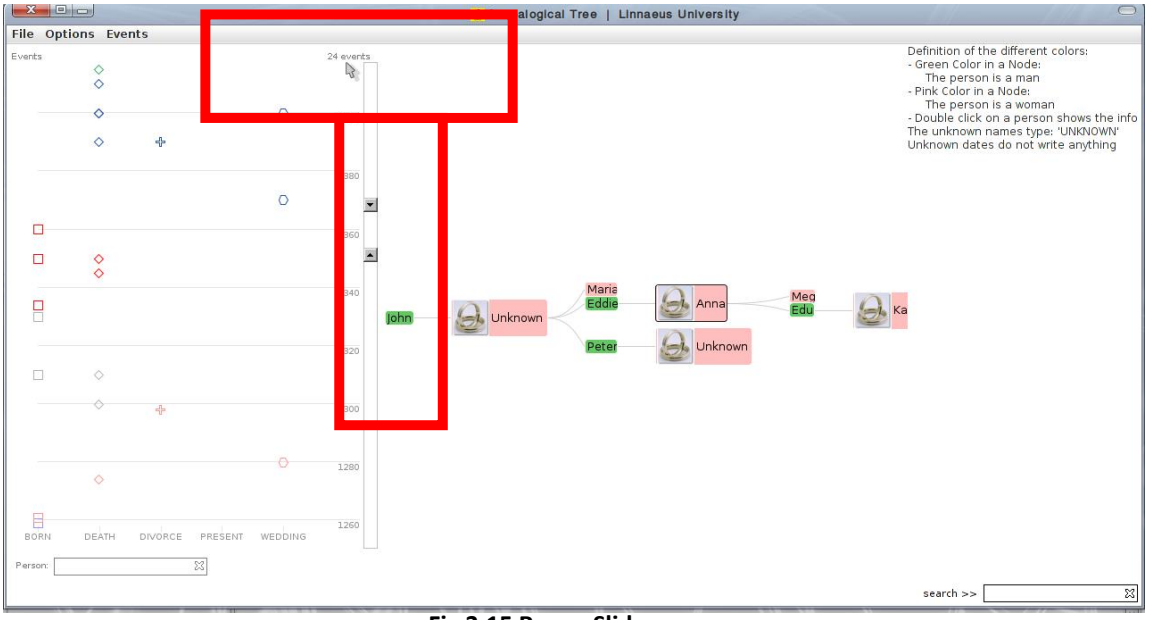

Fig 3.15 Range Slider

An interaction in the tree panel, depending on which kind of visualization that the user is using, we could find different interactions. The common actions are for example, to move the tree clicking outside the tree and moving the mouse, to zoom with the wheel of the mouse or to resize the tree with the right button of the mouse. As for the Ancestor visualization refers, we can find one important added feature. The filter level allows users to show more or less levels of the tree and to do "auto-zoom" or to disable it:

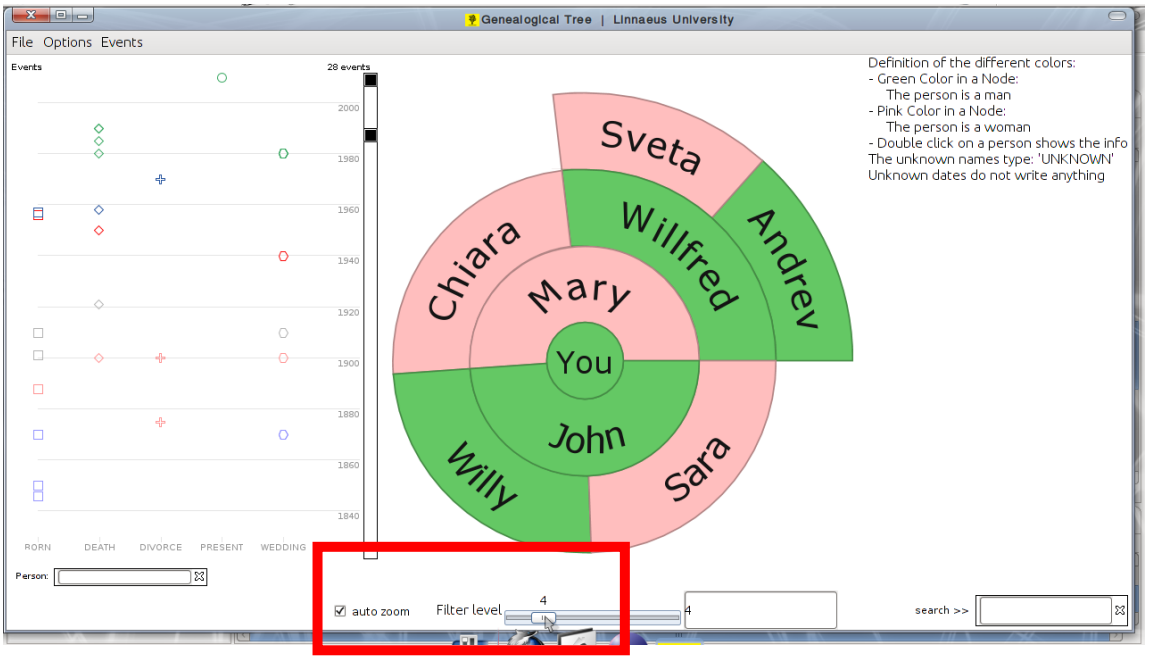

Fig 3.16 Filter Level

In addition, when a spouse is added with the date of the wedding, this date is automatically updated in the husband, too.

Other interaction which has been added to the trees is the possibility to delete one created person. Moreover, if a child or grandparent is created without identifying the person between the selected and the new person, an "unknown" person is automatically created between them.

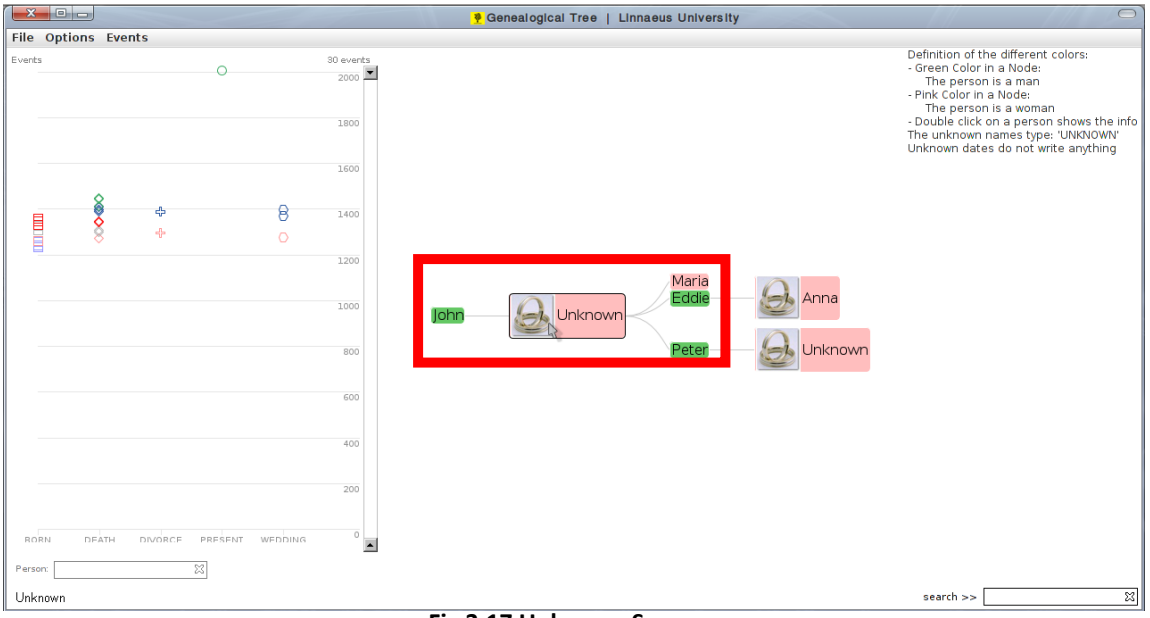

Fig 3.17 Unknown Spouses

Finally, the interaction when the tree is saved, it must work correctly using Xml files. However, this point has not been analyzed in detail as it is not relevant for the topic of the present report.

4 REQUIREMENTS, ANALYSIS AND OVERALL DESIGN

In this chapter, I will present the requirements of the final program, which include the requirements of the main panel, the "event panel" and the way to store the information of the tree. Then, I will list what the user can do with our system.

4.1 Requirements

Defining the requirements of the application in advance is very important. Thus, one can get a general overview about the application to distribute the work properly according to these requirements. In fact, there are two types of requirements: functional requirement and non-functional requirement. I will make a list of requirements that our system should fulfill.

4.2 Non-Functional Requirements

Non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. It is often called quality of a system, such as security, usability, testability, maintainability, extensibility, scalability, etc.

N-FR1	**The system should work fast**
Importance	Essential
Description	The system should work fast. When the user wants to add/edit/show information of a person, the window must appear immediately
N-FR2	**The opening action should be quick**
Importance	Essential
Description	The system must open a stored tree as fast as possible and visualize the content in the correct panel.
N-FR3	**The animation of the people should work fast**
Importance	Essential
Description	The system must work properly when the user changes the current person. Depending on the children of the current person it will have a different animation and different colored action.

Table 4.1 Non Functional Requirements

4.3 Functional Requirements

In software engineering, a functional requirement defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs.

FR1	**Add first person to the tree**
Importance	Essential
Description	The system should be able to create the first person of the tree. This person will be the oldest man of the family to have an order by building the tree.
FR2	**Add wife/children to the tree**
Importance	Essential
Description	The system must be able to create and add a new person (a wife or a child) to the tree
FR3	**Edit a person**
Importance	Essential
Description	The system must be able to show a window where the user can change all the information of a specific person.
FR4	**Show the information of a person**
Importance	Essential
Description	The system must be able to show a window where the user can change all the information of a specific person.
FR5	**Get the information from Facebook**
Importance	Desirable
Description	The user could choose to get the information of a new person from Facebook
FR6	**Save the tree**
Importance	Essential
Description	The system must be able to save the tree in a file
FR7	**Open a stored tree**
Importance	Essential
Description	The system must be able to open and show a tree from a stored file in the computer
FR8	**Show the information of events**
Importance	Essential
Description	The system must show a panel with the events of the people of the tree
FR9	**Represent with different icons/images the different events**
Importance	Desirable
Description	The event panel could show the different events with different pictures because the user can distinguish easier different events.
FR10	**Represent with different color the male and female**
Importance	Desirable
Description	The system must show the people who are male and female in a different color.

Table 4.2 Functional Requirements

4.4 Use Cases

Use case is a description of a system's behavior as it responds to a request that originates from outside of that system.

UC1	Change the color of the node of the tree
Description	When the user selects one node of the tree (a person) to apply an action with this node (add a new person, edit the information, show information…) the node must be colored with green/pink color depending on the sex of the person in that node.
Precondition	The sex of the person must be defined
Postcondition	The node is colored with the correct color.
Sequence Diagram	Figure 4.1

Table 4.3 Use Case 1

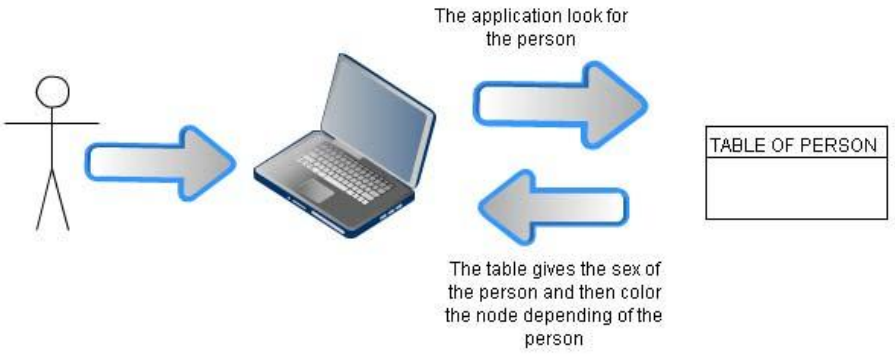

Fig 4.1 Schema about changing color of node

UC2	Update the panel event
Description	When a new person is added or a person in the tree is modified, the file where the events are stored must be modified to show the new information on the system. When the user wants to update the panel event to see the new events must be updated this panel.
Precondition	The file with the events must be created
Postcondition	The file is updated with the new information
Sequence Diagram	Figure 4.2

Table 4.4 Use Case 2

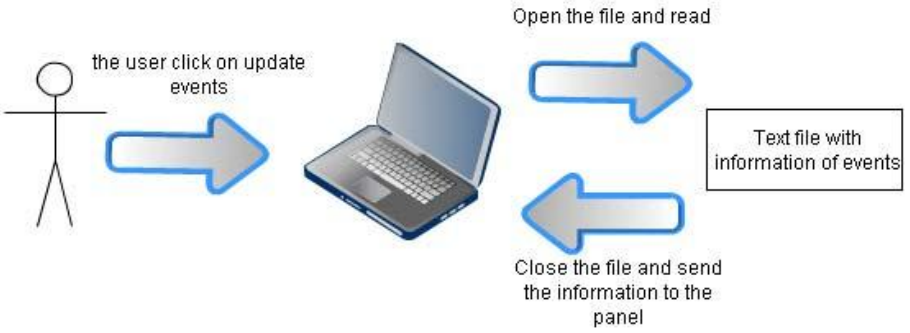

Fig 4.2 The schema how works event panel.

UC3	Change the picture of the person
Description	The user can change the picture of the person. This picture must be available (if the picture is not deleted from the computer), although it has been changed, for the next time the tree is opened.
Precondition	The person has a default picture
Postcondition	The person has a new picture
Sequence Diagram	Figure 4.3

Table 4.5 Use Case 3

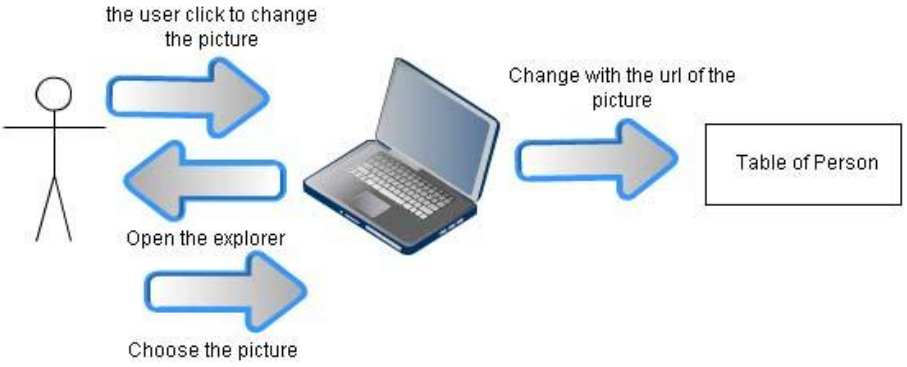

Fig 4.3 Schema how is working to change the picture.

5 IMPLEMENTATION

In this chapter, I would like to show the programming language, external libraries, store system, integrated development environment (IDE), and operating system. Moreover, I would like to continue with the description of the classes in my application.

5.1 Issues

This section is important as I am going to deal with the different languages and libraries which have been used to implement the application. In addition, I will explain the way to create an application on the main site of Facebook.

5.1.1 XML

Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C and several other related specifications, all gratis open standards.

XML's design goals emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for the languages of the world. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

Many application programming interfaces (APIs) have been developed that software developers use to process XML data, and several schema systems exist to aid in the definition of XML-based languages. As of 2009[update], hundreds of XML-based languages have been developed, including RSS, Atom, SOAP, and XHTML. XML-based formats have become the default for most office-productivity tools, including Microsoft Office (Office Open XML), OpenOffice (Open Document) and Apple's iWork.

5.1.2 Servlets

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

Servlets provide component-based, platform-independent methods for building Web-based applications without the performance limitations of CGI programs. Unlike proprietary server extension mechanisms (such as the Netscape Server API or Apache modules), servlets are servers as well as platform-independent. This leaves you free to select a "best of breed" strategy for your servers, platforms, and tools. Using servlets web developers can create fast and efficient server side application which can run on any servlet enabled web server. Servlets run entirely inside the Java Virtual Machine. Since the Servlet runs at server side so it does not checks the browser for compatibility.

The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the

GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

5.2 Prefuse

Prefuse is a set of software tools for creating rich interactive data visualizations. The original **prefuse** toolkit provides a visualization framework for the Java programming language. In addition, the Prefuse flare toolkit provides visualization and animation tools for ActionScript and the Adobe Flash Player.

Prefuse supports a rich set of features for data modeling, visualization, and interaction. It provides optimized data structures for tables, graphs and trees; a host of layout and visual encoding techniques, and support for animation, dynamic queries, integrated search, and database connectivity. Prefuse is written in Java, using the Java 2D graphics library and it is easily integrated into Java Swing applications or web applets. It is licensed under the terms of a BSD license, and can be freely used for both commercial and non-commercial purposes.

5.2.1 Structures in Prefuse

In Prefuse there are some different kinds of structures to represent and store the information which we want to work with. Prefuse has implemented these structures making easier the work to the programmers. The problem is to understand how they work and decide which one is better for our case.

The first structure we can find is the table. A Table organizes a collection of data into rows and columns, each row contains a data record, and each column contains data values for a named data field with a specific data type. Table data can be accessed directly using the row number and column name, or rows can be treated in an object- oriented fashion using Tuple instances that represent a single row of data in the table. As such, tables implement the TupleSet interface.

Table rows can be inserted or deleted. In any case, none of the other existing table rows are affected by an insertion or deletion. A deleted row simply becomes invalid—any subsequent attempts to access the row either directly or through a pre-existing Tuple instance will result in an exception. However, if news rows are later added to the table, the row number for previously deleted nodes can be reused. In fact, the lower row number currently unused is assigned to the new row. This results in an efficient reuse of the table rows, but carries an important side effect -- rows do not necessarily maintain the order in which they were added once deletions have occurred on the table. If not deletions occur, the ordering of table rows will reflect the order in which rows were added to the table.

Collections of table rows can be accessed using both iterators over the actual row numbers and iterators over the Tuple instances that encapsulate access to that row. Both types of iteration can also be filtered by providing a Predicate, allowing tables to be queried for specific values.

Columns (alternatively referred to as data fields) can be added to the Table using addColumn (String, Class) and a host of similar methods. This method will automatically determine the right kind of backing column instance to use. Furthermore, Table columns can be specified using a Schema instance, which describes the column names, data types, and default values. The Table class also

maintains its own internal Schema, which be accessed (in a read-only way) using the getSchema() method.

Tables also support additional structures. The ColumnMetadata class returned by the getMetadata(String) method supports calculation of different statistics for a column, including minimum and maximum values, and the number of unique data values in the column. Index instances can be created and retrieved using getIndex(String) method and retrieved without triggering creation using getIndex(String) method. An index keeps a sorted collection of all data values in a column, accelerating the creation of filtered iterators by optimizing query calculations and also providing faster computation of many of the ColumnMetadata methods. If you will be issuing a number of queries (i.e., requesting filtered iterators) dependent on the values of a given column, indexing that column may result in a significant performance increase, though at the cost of storing and maintaining the backing index structure.

I talked before about Tuple. This one is so important to understand later the composition of a person in the tree. The person will be a tuple of different values. Tuples are objects representing a row of a data table, providing a simplified interface to table data. They maintain a pointer to a corresponding row in a table. When rows are deleted, any live Tuples for that row become invalidated and any further attempts to access or set data with that Tuple will result in an exception.

The tree is a graph subclass that models a tree structure of hierarchical parent-child relationships. For each edge, the source node is considered the parent, and the target node is considered the child. For the tree structure to be valid, each node can have at most one parent, and hence only one edge for which that node is the target. In addition to the methods of the Graph class, the tree also supports methods for navigating the tree structure, such as accessing parent or children nodes and next or previous sibling nodes (siblings are children nodes with a shared parent). Unlike the graph class, the default source and target key field names are renamed to DEFAULT_SOURCE_KEY and DEFAULT_TARGET_KEY. Like the Graph class, Trees are backed by node and edge tables, and use Node and Edge instances to provide object-oriented access to nodes and edges.

The Tree class does not currently enforce that the graph structure remain a valid tree. This is to allow a chain of editing operations that may break the tree structure at some point before repairing it. Use the isValidTree() method to test the validity of a tree.

The idea of the event panel is taken from one of the demos prefuse provides. But only the idea, because I had to do a lot of changes to get the final result. The changes between the original and the last version are clearly recognizable. The first one is the way to take the information. The first one takes this information from a list of words in a txt file. The final version takes the information from a txt file but in this txt file there is a “table” created. To separate the different attributes to represent (the different events) the system creates a first row with the title of the fields. This table is separated by tabs. The next rows will be created dynamical and automatically when a person is added or edited.

The second change from the original code to the final code is the way how is changing dynamically the representation. The first one had default values in the x-

axis. Now it will start with a value (present) and then more values will be added. At the same time the events will be added, i.e. if there is a wedding a new field will appear in the x- axis (and no more fields); if after that a death is added, it will appear with the new value.

The third change is the title when the mouse is over an event. In the first one, there was a lot of information and nothing related with our case.

The fourth change is in the y-axis. The values of the original were different and nothing related to our case (there was a scale of euros and now is a scale of years). Because of that, I changed the range of zoom of this scale and of course, the values.

The fifth change was in the filter with radiobuttons. This is the same case of the x-axis. These radiobuttons are created dynamically with the information of a person. It will filter the events by years. The default value will be the present. But if some event happens in other year (i.e. in 1999), a new radiobutton appears with the value 1999 and we will be able to be used to have a new filter.

The last change is in the visualization of the events. The original only was different with the color and two forms. Now there are more forms to separate the different events and more colors to be separated by year.

In the case of the Tree Panel I started (like with the event panel) to work with one demo from Prefuse, but finally I had to dispose because it was impossible to save all the information of a person in the tree and represent the tree in the way I wanted. So, I only took the animations of the tree and I had to think how to save the information of the person. The first thing I thought, it was to create a class Person with all the attributes and then creating Nodes of person. However, what is the problem of this idea? It is the time to store the information. We wanted to store the information in an xml file with own tags. But a person is not a field it, is an object and we cannot store a person only with tags.

The second and final solution was better to store the data and to send the information to the event panel. This problem is solved creating a table in parallel with the tree saving the information of the people there (with all the attributes of the person). Why is this better to store with an xml file? Because the tree can be represented with tags but the content of these tags must be “primary classes” (Integer, String, Boolean…). I can send all this information to a txt file which will be used by the event panel.

5.2.2 Visualizations in Prefuse

Prefuse also provides some demos of visualizations. Some parts of these demos I could use for the application. But the most important part of these demos is the animations they have.

Graphics using the structure of the tree:

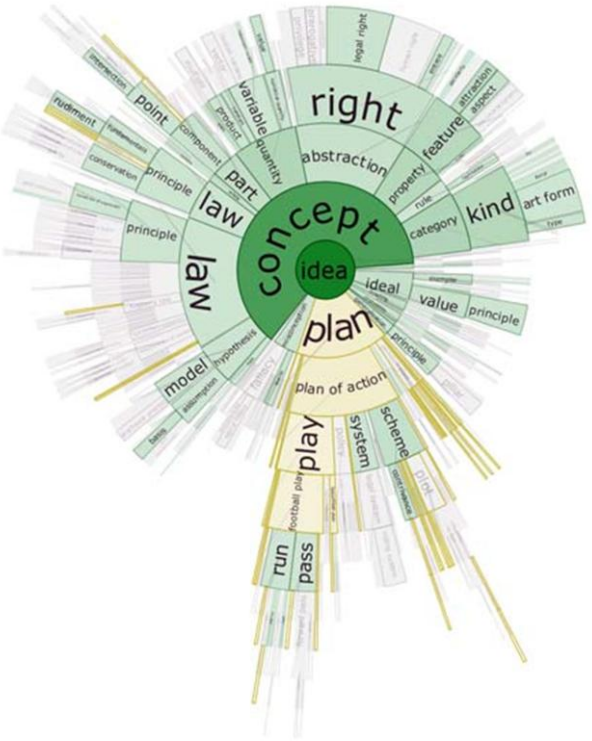

Fig 5.1 DocuBurst taken from [7]

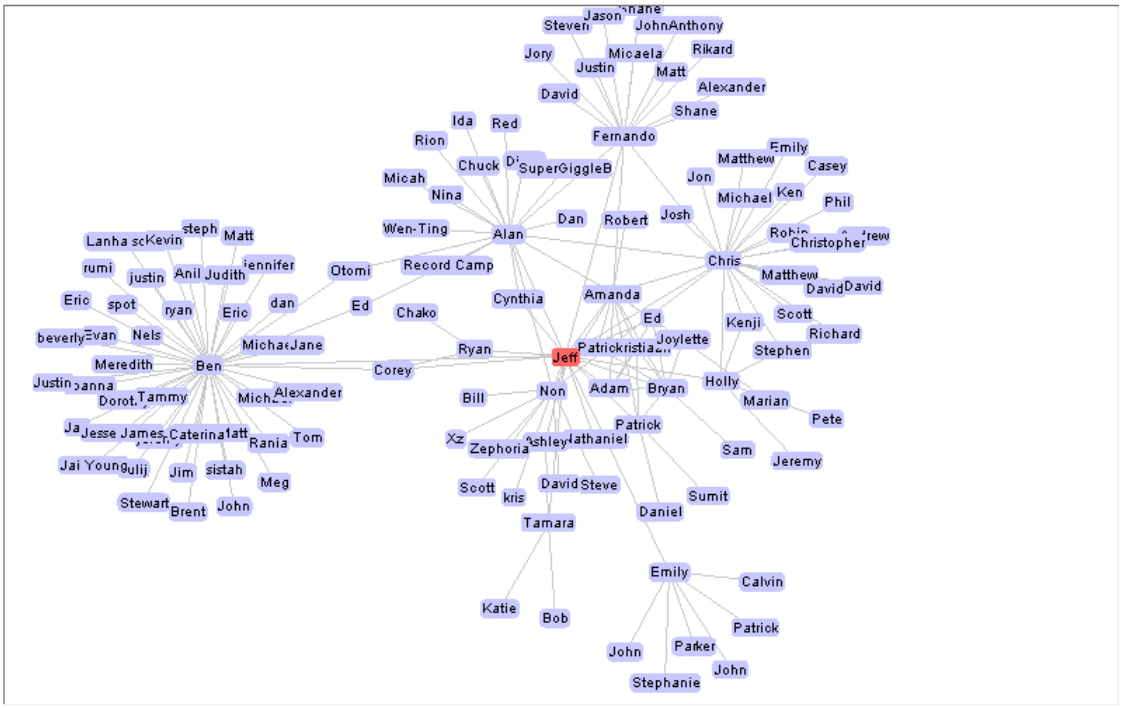

Fig 5.2 GraphView taken from [7]

Example in prefuse of structures using tables or files to get the information:

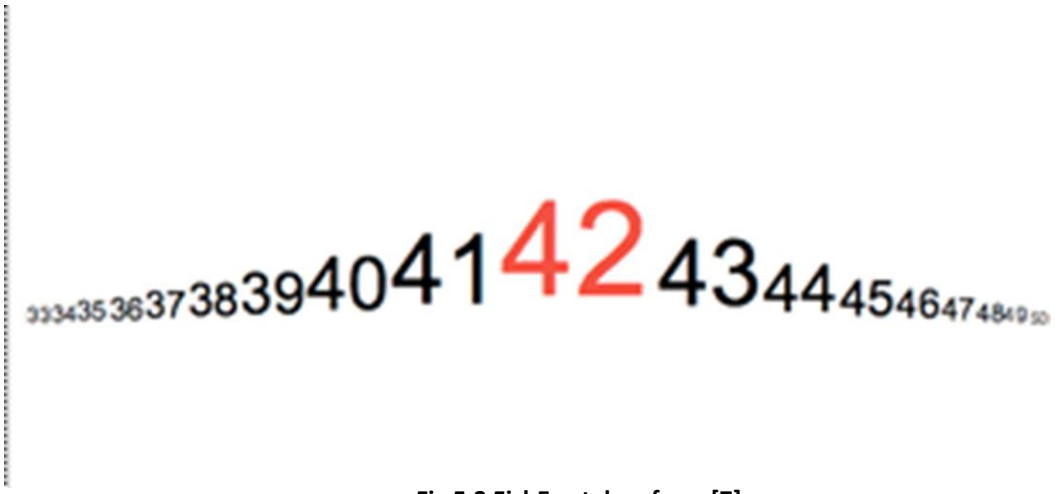

Fig 5.3 FishEye taken from [7]

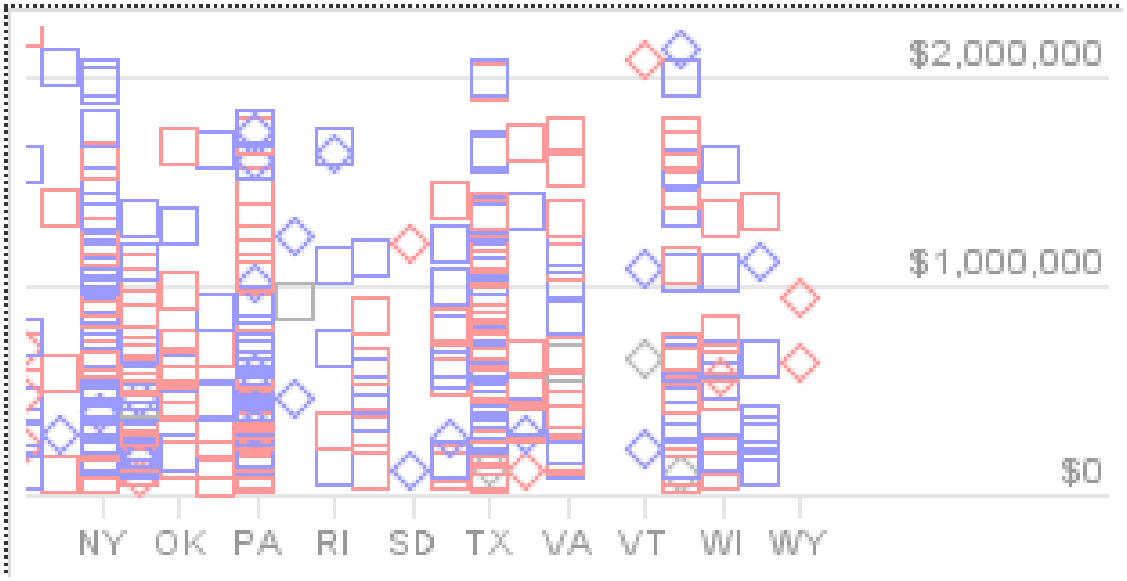

Fig 5.4 Congress taken from [7]

5.2.3 Connecting with Facebook

The first thing I did was to establish a connection to Facebook in order to create the application on Facebook. To do this, the first step is request Facebook about how to become a developer. Once you have the privileges of developer, then you register your application with a name, an icon and configure more options.

After this, Facebook provides you a public key and the secret key. With this secret key only the developer can connect it in order to develop everything on Facebook. Also, Facebook provides the AppName, the Callback URL, the Canvas page, the iFrame and the post-add URL. We will need the appName, the iFrame and the public and secret key.

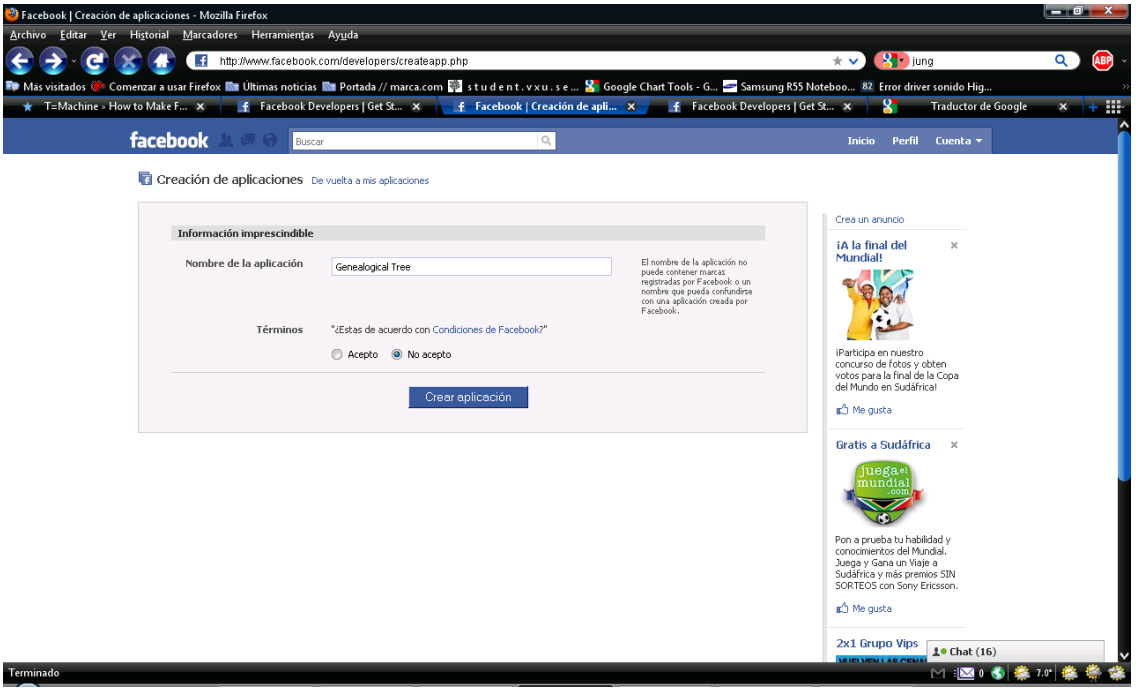

Fig 5.5 Window: name of Application from Facebook.

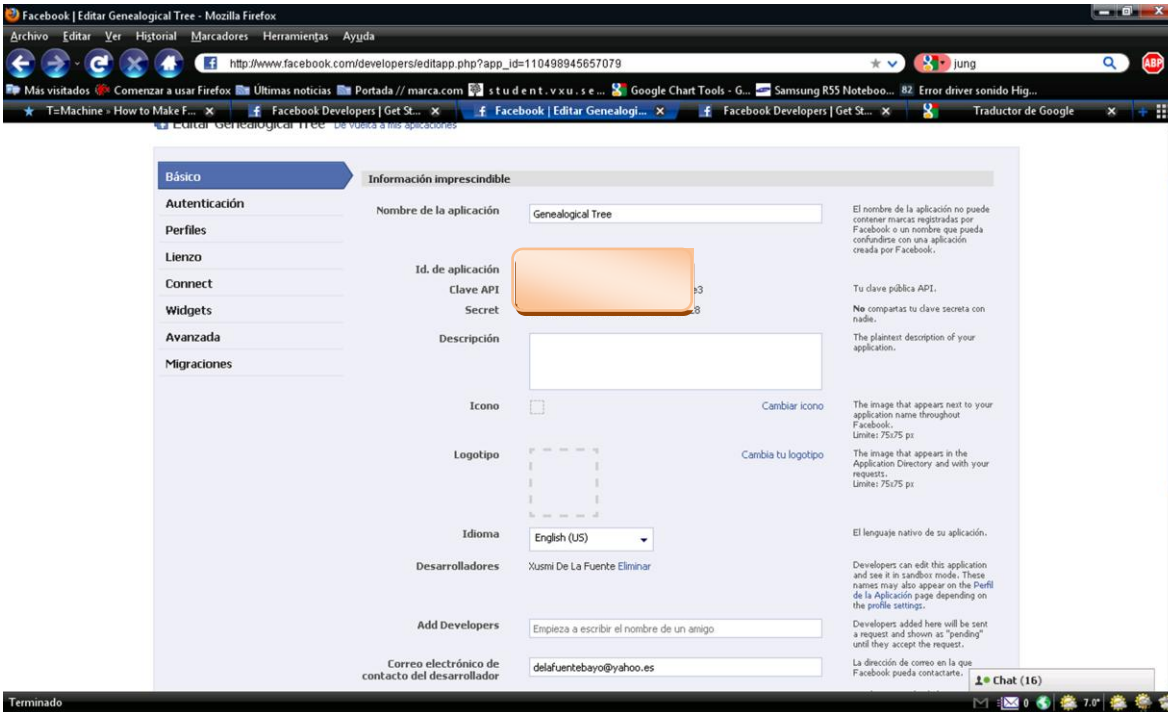

Fig 5.6 Data about the application (the keys, id,name CallbackURL…)

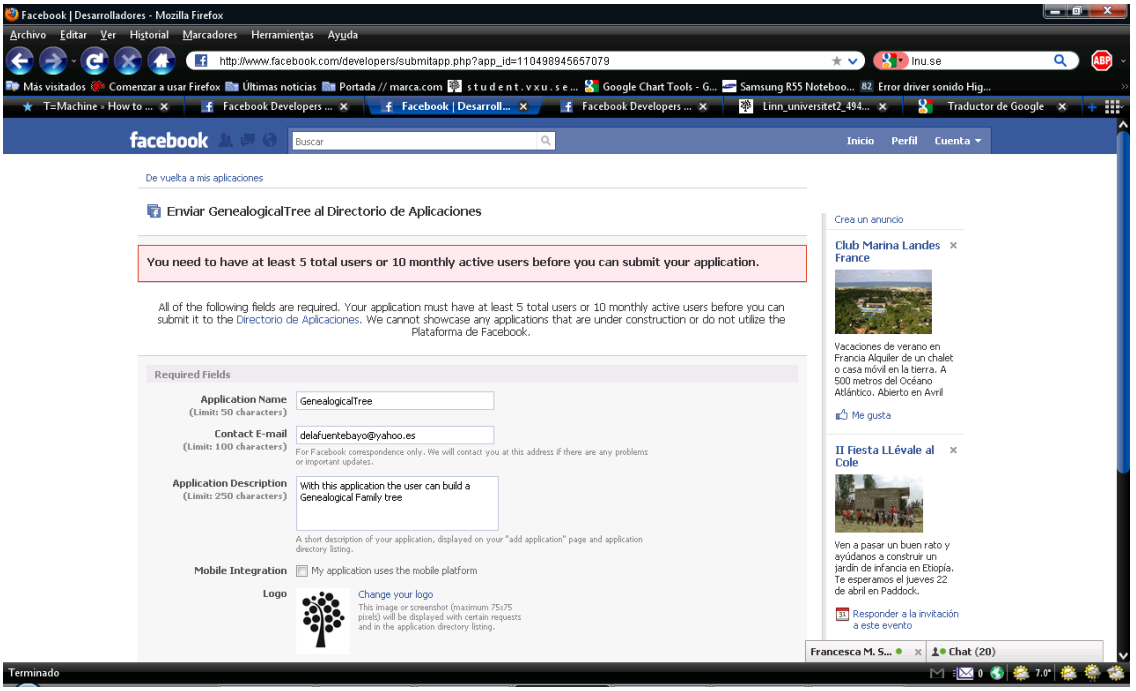

Fig 5.7 Window: Submit your application to Facebook

The next step is to download the API of Facebook to connect to our Java Application with the social network.

Then, Facebook notifies that you need to work with J2EE, necessary to connect later with Facebook.

The way Facebook works "requires" you to provide a webserver for your app, for that reason we need to find a server and install J2EE. This is like J2SE (the normal way to programming) with more libraries; we need few of these libraries. What is exactly J2EE?

Java Enterprise Edition (J2EE): J2EE is a platform-independent, Java-centric environment for developing, building and deploying Web-based enterprise applications online.J2EE includes many components of the Java Standard Edition (J2SE).The J2EE platform consists of a set of services, APIs, and protocols that provide the functionality for developing multitier, Web-based applications.

J2EE simplifies application development and decreases the need for programming and programmer training by creating standardized, reusable modular components and by enabling the tier to handle many aspects of programming automatically.

If you are an enterprise developer you need J2EE. Enterprise developers need J2EE because writing distributed business applications is not easy, and they need a high- productivity solution that allows them to focus only on writing their business logic and having a full range of enterprise-class services to rely on, like transactional distributed objects, message oriented middleware, and naming and directory services.

iFrames is important to connect Java with Facebook. This will be the "library" which allows us implement the server and interact with Facebook.

Actually, the way Facebook have designed their system, you need your own webserver anyway. In another hand, they will not let you make a Facebook App without one.

Facebook provides you with a java class, FacebookRestClient.java, which you can use to authenticate with the FB servers. Once you have authenticated an instance of this class, that class will then also provide you with access to all the features of Facebook (reading data about users, sending messages to other users, etc). Of course, this class must be implemented on the servlet.

Traditionally, when writing a J2EE app, you have multiple servlets, one for each "page" of your website, but in order to authenticate, the servlet must redirect the user back to Facebook, which will then always send the user back to your Callback URL page. That means that they will be sent to the "wrong" page.

In conclusion, Facebook only allows one servlet to authenticate with Facebook, effectively; whichever servlet you gave the address of when you filled in the "callback URL" field when creating your app on Facebook's Developer page.

5.2.4 Summary

To start with the application I had to decide about what technologies were better for my final task, because depending on the final result, it could be easier to build the system with different programming languages and different kind of trees.

The final choice was to create a Classical Hierarchical Tree, although I had to hide the data that was not directly connected with the current person in order to avoid the problem with the unused space.

The technology to create the system finally was Java. Why? It is because Prefuse and the library SWING offered a good and interesting GUI to visualize the data. Moreover programming in Java is easier for me. The other case would have been more difficult because I must have learned about that programming language.

Finally, to store the data of the tree, we will use XML because is "universal language" on the Internet and if we wanted to upload this file to a server we could visualize the information of the xml because the data is stored with tags and values.

Why did I explain the servlets if finally I have not used use them? Because this is the step I need to connect with the social networking (Facebook in our case). Without a server to install the servlet which connect the java application with Facebook to take the data of the contacts on Facebook I do not need to implement it. Even though, it is important to know how servlets are working if we have a server and we want to connect with Facebook in a future.

In the appendix I will explain the tools, languages and libraries which I was working with. Also I will explain some important classes of the application with a brief description in more detail.

6 CONCLUSIONS

In this Chapter, I will retake the goal of the thesis and give the conclusions and the experience I have had implementing this thesis. At last, the future work could be done in the application like more modules, improvements or more checking.

6.1 Conclusion

The goal of the thesis is to develop an application (for example Facebook application) that will allow users to build and share visualization of genealogical data. Additionally it should visualize time-dependant data, such as various events like marriage, etc.

Finally, the application can manage all this information and create the visualization for the user. It works properly and the main goals are fulfilled. The user will find a Facebook application, the visualization will change automatically when the user interact with the program and will be able to store and open trees created.

Taking into account that one task of this thesis is to create the application for the social network and there, users want something user-friendly and aesthetically pleasing interface, which is the biggest difference to build an application for a company (could be more specifically and complicated or do not care if has a look and feel interface) and an application for a social network.

6.2 Experience

To start with the application, it was necessary to start studying the different technologies to work with trees graphically. This is because it is necessary to have a look and feel presentation of the application to get the attention of the user. Possibly the user of the social network will not be interested to use the application without a good GUI and a good presentation of this topic. I can say that I have experience an improvement with in knowledge of the GUI in Java.

When I decided to work in technologies, I started with the implementation. I needed to study the code that Prefuse provides about trees, tables and how to store the information in a file. I decided to study this technology because Prefuse also provides friendly animations with the interface. The conclusion of this part is that I learnt about this unknown library for me and the possibilities that Prefuse provides to store and visualize the data.

The second technology I am experienced with is XML. For saving information I used xml files. I had to change some code, in order to be able to save a person and not only fields without any relation.

Finally, it has been interesting for me to see how to register the application on Facebook and having the public key, the secret key and the name of the application. If one day we have a server to install a servlet there and to interact between Java and Facebook, taking the information of the people from the social network. This part is very interesting for me because I did not know anything about the “Developers” Section of Facebook and it has provided me some different tools to implement my own application and use the information that Facebook provides to the developer.

Now when the thesis is finished, I got a new idea about how to visualize the graphics. The most important idea about the visualization of the data is how the user will understand the graphic we are representing. With the first visualization, the user must be able to understand how the application is working and what it is been

visualized exactly on the screen. Because of these reasons, I was really motivated to work on it.

Moreover, it has been a good opportunity to learn and work on this topic of the computer science, to learn more about programming, to learn about a social network interacts with the users, to work with which are unknown to me such as xml, servlets or to create an application on Facebook.

6.3 Future Work

Due to the limited time, there are some works left and other one that could have been improved or done in another way.

1. The first is implementing the connection between Facebook and Java. As I have mentioned before in the subsection 3.1.5, it is necessary a server to implement and install the servlet, to connect with Facebook.
2. The second one could be more checks up when a person is added. For example, check the dates: The date of the divorce after the wedding, the date of a birth before death and wedding…

APPENDIX

A.1 Tools, languages and libraries

Programming language	**Java J2EE**
Description	Language for developing, building and deploying Web-based enterprise applications online
Library	**JUNG**
Description	Java Universal Network/Graph Framework is a software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. Just used for some animations.
Library	**Prefuse**
Description	Prefuse is a set of software tools for creating rich interactive data visualizations. The original prefuse toolkit provides a visualization framework for the Java programming language.
Library	**Swing**
Description	Swing is a widget toolkit for Java. Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.
Store management	**XML**
Description	Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards. XML was designed to transport and store data.
Operating system	**Linux Ubuntu 10.01/Windows XP (Java is multiplatform)**
IDE	**Eclipse for Java EE Platform Version: Galileo**
Description	Eclipse is a multi-language software development platform comprising an IDE and plug-in system to extend it. It is written primarily in Java and is used to develop applications in this language

Table A.1 Tools, Languages and libraries

A.2 Class Description

In this section, I will present the detailed implementation by explain the core classes description.

A.2.1 Class Description of Visualizations.

Class	TreeGen
Description	This class has the main method and is used to visualize the main window with the 2 panels
Inner classes	AutoPanAction: The actions the tree must to do when the user is interacting with it. NodeColorAction: The action to color the node depending of some attributes of the Node. OrientAction: The actions must do the tree when the user wants to change the orientation or the visualization size of the tree
Methods	Demo (): Creates the window and the structure of the tree empty Demo (String, String): Creates the window with the structure of the tree from a file with the name of the String Demo (String, String, Boolean): Creates the same of last one but when the user chooses the option open a file. TreeGen (Tree,String): Creates the object TreeGen. AddChildToTree(): Add a new child to tree. AddDadtoTree(): Add the first node of the tree. AddWifetoTree(): Add a wife to the current node. EditChildToTree(): Edit the information of the current node. getOrientation(): Get the orientation of the tree. setOrientation (int): Set the orientation vertical/horizontal depending of the integer.
Structure Diagram	Table A.2

Table A.2 Class Description TreeGen

Fig A.1 Schema of Class TreeGen

Class	Vista
Description	This class creates the window for add a new Child to the current Node (the same cases for add a wife and add the first node).
Inner classes	OyenteBoton: The class listener for the buttons of the window. This is the important code in this class because it will send the information to the tree.
Methods	Getters and Setters of the attributes. Vista (TreeGen): Creates the Frame where there will be the fields to fill the information of the person

Table A.3 Class Description Vista

Class	Edit
Description	This class creates the window for edit the information of the person of the current Node
Inner classes	OyenteBoton: The class listener for the buttons of the window. The same of the last explained class. Here will be the important code.
Methods	Getters and Setters of the attributes. Edit (TreeGen, Node): Creates the Frame where there will be the fields to fill the information of the person

Table A.4 Class Description Edit

Class	VentanaDos
Description	This class creates the window for show the information of the person is in the current Node
Inner classes	OyenteBoton: The class listener for the buttons of the window
Methods	Getters and Setters of the attributes. VentanaDos (Node): Creates the Frame where there will be the fields with the information of the person and the picture. This one could be changed clicking on it.

Table A.5 Class Description VentanaDos

Class	Congress
Description	This class creates the event panel
Inner classes	Counter: count the total of a group
Methods	Getters and Setters of the attributes. Congress (Table): Creates the Panel of the Events taken the information from the given table. DisplayLayout (): display the format of the panel.
Structure Diagram	Figure A.2

Table A.6 Class Description Congress

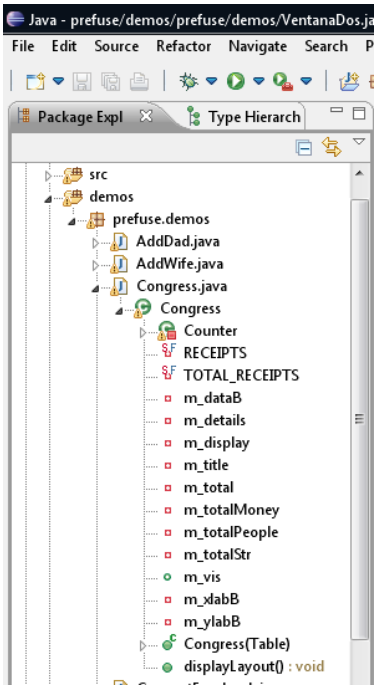

Fig A.2 Schema of class Congress

Class	**Display**
Description	This class is responsible for drawing items to the screen and providing callbacks for user interface actions such as mouse and keyboard events. A Display must be associated with a Visualization from which it pulls the items to visualize.
Inner classes	InputEventCapturer: to capture all mouse and key events on the display, detects relevant VisualItems, and informs ControlListeners. TransformActivity: Activity for conducting animated view transformations.

Methods	Getters and Setters of the attributes. Display(): Creates a new Display instance. You will need to associate with a Visualization for it to display anything. Display (Visualization): Creates a new Display associated with the given Visualization. By default, all VisualItem instances in the Visualization will be drawn by the Display. Display (Visualization,String): Creates a new Display associated with the given Visualization that draws all VisualItems in the visualization that pass the given Predicate. The precicate string will be parsed by the ExpressionParser to get a Predicate instance. Display (Visualization, Predicate): Creates a new Display associated with the given Visualization that draws all VisualItems in the visualization that pass the given predicate. AddControlLintener(Control): Adds a ControlListener to receive all impugn events on VisualItems. AddPaintListener (PaintListener): Add a PaintListener to this Display to receive notifications about paint events. AnimatePan(double, double, long): Animate a pan along the specified distance in screen (pixel) coordinates using the provided duration. Update (Graphics): Update the display. Zoom (Point2D, double): Zooms the view provided by this display by the given scale, anchoring the zoom at the specified point in absolute coordinates.
Structure Diagram	Fig A.3

Table A.7 Class Description Display

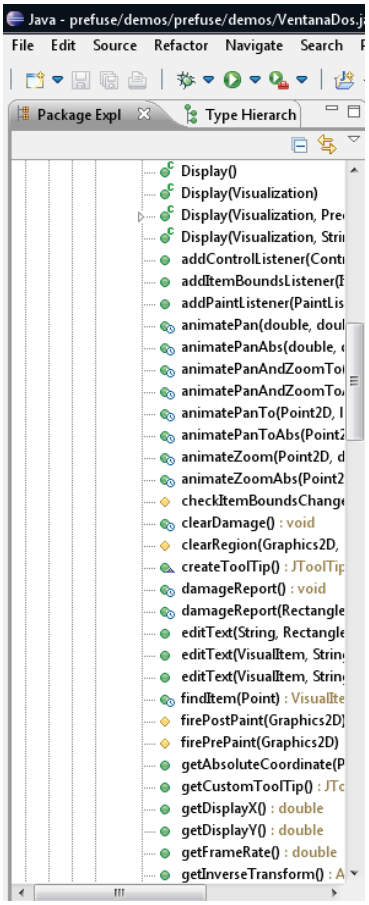

Fig A.3 Schema of class Display

Class	**Visualization**
Description	This class is responsible for managing the mappings between source data and onscreen. VisualItems, maintaining a list of display instances responsible for rendering of and interaction with the contents of this visualization, and providing a collection of named Action instances for perfoming data processing such as layout, animation, size, shape and color assignment.
Methods	Getters and Setters of the attributes. Visualization (): Creates a new, empty Visualization. Uses a DefaultRendererFactory. AddDisplay(Display): Add a display to this visualization. Called automatically by the setVisualization (Visualization) method in Display class. AddTree(String,Tree): Add a tree to this visualization, using the given data group name. A visual abstraction of the data will be created and registered with the visualization. An exception will be thrown if the group name is already in use.
Structure Diagram	Fig A.4

Table A.8 Class Description Visualization

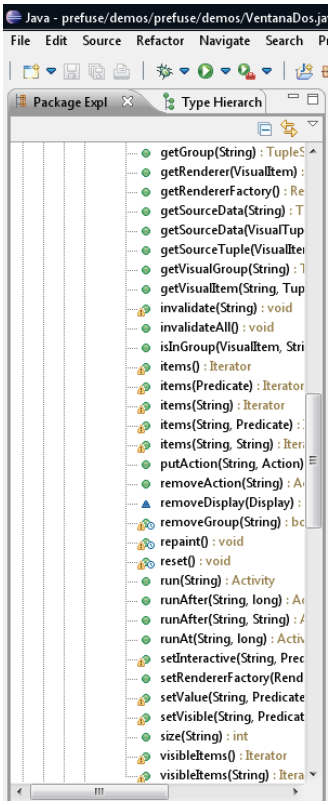

Fig A.4 Schema of Visualization

A.2.2 Class Description of Data Information.

Class	**Person**
Description	This class creates objects of person. Is the step to store this data between the user interface and the table will be finally stored in a xml file.
Methods	Getters and Setters of the attributes. Person(): Create a person by default. Empty fields. Person(String, Integer): Create a person with the given name and given year of born. toString(): Shows the information of the name of the person to visualize in the tree.
Structure Diagram	Fig A.5

Table A.9 Class Description Person

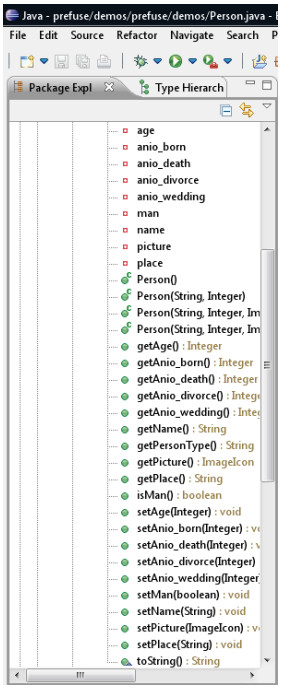

Fig A.5 Schema of Person

Class	**TreeMLReader**
Description	GraphReader instance that reads in tree-structured data in the XML-based TreeML format. TreeML is an XML format originally created for the 2003 InfoVis conference contest. A DTD (Document Type Definition) for TreeML is available online
Inner classes	TreeMLHandler: A SAX Parser for TreeML data files. Interface Tokens: String tokens used in the TreeML format
Methods	readGraph(InputStream): Read in a graph from the given InputStream.

Table A.10 Class Description TreeMLReader

Class	**TreeMLWriter**
Description	GraphWriter instance that writes a tree file formatted using the TreeML file format. TreeML is an XML format originally created for the 2003 InfoVis conference contest. A DTD (Document Type Definition) for TreeML is available online
Methods	writeGraph(Graph,OutputStream): Write a graph from the given OutputStream.

Table A.11 Class Description TreeMLWriter

Class	TableReader
Description	Interface for classes that read in Table data from a particular file format
Methods	readTable(File): Read in a table from the file at the given location. Though not required by this interface, the String is typically resolved using the method, allowing URLs, classpath references, and files on the file system to be accessed. readTable(URL): Read in a table from the given URL. readTable (File): Read in a table from the given File. readTable (InputStream): Read in a table from the given InputStream.
Structure Diagram	Fig A.6

Table A.12 Class Description TableReader

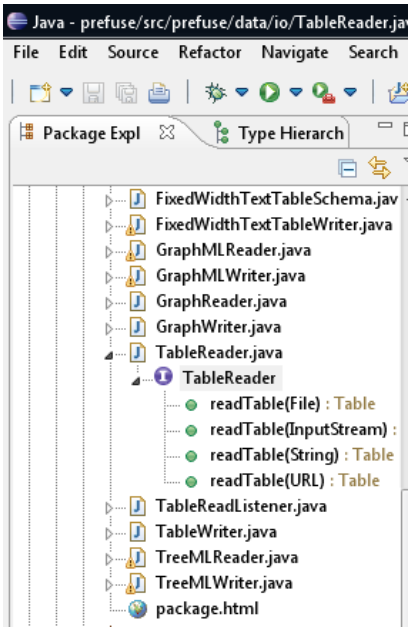

Fig A.6 Schema of TableReader

Class	DelimitedTextTableReader
Description	TableReader for delimited text files, such as tab-delimited or pipe-delimited text files. Such files typically list one row of table data per line of the file, using a designated character such as a tab or pipe to demarcate different data columns. This class allows you to select any regular expression as the column delimiter
Methods	DelimitedTextTableReader(): Create a new DelmitedTableReader for reading tab-delimited files using a default parser factory DelimitedTextTableReader(ParserFactory): Create a new DelimitedTextTableReader for reading tab-delimited files. DelimitedTextTableReader(String): Create a new DelimitedTextTableReader using a default parser factory. DelimitedTextTAbleReader (String,ParserFactory): Create a new DelmitedTextTableReader. Read(InputStream, TableReaderListener): Read the text file given.
Structure Diagram	Fig A.7

Table A.13 Class Description DelimitedTextTableReader

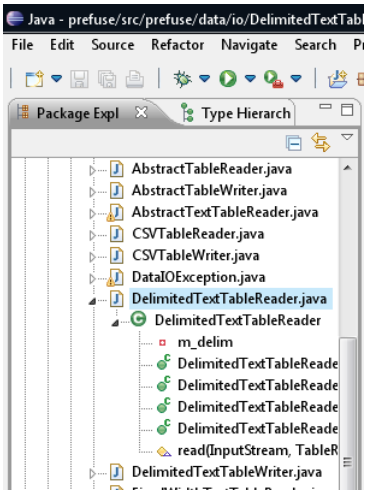

Fig A.7 Schema of DelimitedTextTableReader

Class	TableWriter
Description	Interface for classes that write Table data to a particular file format
Methods	writeTable (Table,String): Write a table to the file with the given filename. writeTable(Table, File): Write a table to the given File. writeTable (Table, OutputStream): Write a table from the given OutputStream.
Structure Diagram	Fig A.8

Table A.14 Class Description TableWriter

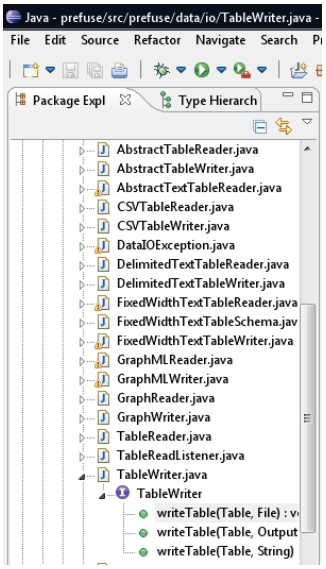

Fig A.8 Schema of TableWriter

Class	DelimitedTextTableWriter
Description	TableWriter that writes out a delimited text table, using a designated character string to demarcate data columns. By default, a header row containing the column names is included in the output.
Methods	Getters and Setters. DelimitedTextTableWriter(): Create a new DelmitedTableWriter that writes tab-delimited text files. DelimitedTextTableWriter(String): Create a new DelmitedTableWriter. DelimitedTextTableWriter(String,bool): Create a new DelmitedTableWriter. writeTable (Table, OutputStream): Write the table in the Output.

Table A.15 Class Description DelimitedTextTableWriter

REFERENCES

[1] http://www.facebook.com/note.php?note_id=107849347899 *(2010/6/10)*

[2] http://digitalcitizen.ca/2009/04/19/why-many-people-dont-believe-in-the-power-of-social-media/ *(2010/8/18)*

[3] http://www.elcomerciodigital.com/apoyos/especiales/enlacereal2/arbol_principe.htm *(2010/6/9)*

[4] http://cs.msi.vxu.se/isovis/ *(2010/6/1)*

[5] JUDITH R. BROWN …[ET AL.] *Visualization: Using computer graphics to explore data and present information (1995)* John Wiley & Sons, Inc., USA

[6] http://atalap1.blogspot.com/2009/04/arbol-genealogico.html *(2010/5/28)*

[7] http://prefuse.org/ *(2011/4/15)*

[8] http://en.wikipedia.org/wiki/Java_%28programming_language%29 *(2010/06/3)*

[9] http://download.oracle.com/javase/tutorial/ui/overview/intro.html *(2010/6/7)*

[10] http://en.wikipedia.org/wiki/Swing_%28Java%29 *(2010/7/4)*

[11] http://www.w3schools.com/XML/xml_whatis.asp *(2011/4/11)*

[12] http://en.wikipedia.org/wiki/XML *(2011/04/11)*

[13] http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets2.html *(2011/3/10)*

[14] http://www.hoppy.com/family/family.htm *(2010/4/19)*

[15] http://thebankshow.com/2008/03/26/car-_company-_family-_tree/ *(2010/4/19)*

[16] http://www.balgownie.co.uk/ *(2011/2/21)*

[17] http://www.phpgedview.net *(2010/11/14)*

[18] http://jung.sourceforge.net *(2011/04/11)*

[19] http://flare.sourceforge.net *(2010/10/23)*

[20] http://cs.lnu.se/isovis/theses/ongoing/ *(2011/4/28)*

[21] http://linnaeus.academia.edu/Departments/Computer_Science *(2010/12/12)*

[22] http://www.gramps- project.org *(2011/2/10)*

[23] http://www.phpgedview.net/ *(2010/11/14)*

[24] http://www.myheritage.com *(2010/11/14)*

[25] http://flare.prefuse.org *(2010/10/23)*

[26] TIPPING, M.E. BISHOP *Artificial Neural Networks, Fifth International Conference on (Conf. Publ. No. 440)* (1997) Neural Comput. Res. Group, Aston Univ., Birmingham

[27] MCGUFFIN, M.J. BALAKRISHNAN, R. *Interactive Visualization of Genealogical Graphs (2005)* Dept. of Computer Science, Toronto Univ., Ontario.

[28] HERMAN, G. MELANON, M.S. MARSHALL. *Graph Visualization and Navigation in Information Visualization* (2000) Centre for Mathematics and Computer Sciences (CWI)

[29] http://en.wikipedia.org/wiki/Treemapping *(2011/3/3)*

[30] MINNIE F. MICKLEY, *Our Acnestors: The Kern Family of America* (1912) NGS Quartely

[31] CLIFFORD K. SHIPTON, *"Report of the Council", Proceedings of the American Antiquarian Society* (1960, 70) Worcester

[32] ELIZABETH SHOWN MILLS, *Genealogy in the "Information Age"* (1732) NGS Quartely

[33] ANDREJ MRVAR, VLADIMIR BATAGELJ, *Relinking marriages in Genealogies* (2004) Metodoloski zvezki

[34] ANASTASIA BEZERIANOS… [ET AL.], *GeneaQuilts: A System for Exploring Large Genealogies* (2010) IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS.

[35] CLAURISSA TUTTLE… [ET AL.], *PedVis: A Structured, Space-Efficient Technique for Pedigree Visualization* (2010) IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS.

[36] NAM WOOK KIM… [ET AL.], *Tracing Genealogical Data with TimeNets* (2010) Computer Science Department, Stanford University

Linnæus University
School of Computer Science, Physics and Mathematics

SE-391 82 Kalmar / SE-351 95 Växjö
Tel +46 (0)772-28 80 00
dfm@lnu.se
Lnu.se/dfm