ChristalView, the visualization of a christal

a visualization of multiple overlapping hierarchical classification structures

Mank, Freek W.M.

Award date:
2005

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER'S THESIS

CristalView - The visualization of a Cristal

A visualization of multiple overlapping hierarchical classification structures

by
F.W.M. Mank

Supervisors:

dr. ir. E. van Bracht (Statistics Netherlands)
ir. M. Vollebregt (Statistics Netherlands)
prof. dr. ir. J.J. van Wijk (TU/e)

Eindhoven, June 2005

CristalView - The visualization of a Cristal
A visualization of multiple overlapping hierarchical classification structures
F.W.M. Mank

Department: Computer Science Voorburg
Sector: Technology, Methods and Development
Division: Technology and Methodology
Statistics Netherlands (SN)

Master's Thesis
F.W.M. Mank
Studentnumber: 472468
Department: Mathematics and Computer Science
Eindhoven University of Technology (TU/e)

Supervisors:
dr. ir. E. van Bracht (Statistics Netherlands)
ir. M. Vollebregt (Statistics Netherlands)
prof. dr. ir. J.J. van Wijk (TU/e)

Abstract

At Statistics Netherlands (SN), the Dutch national statistics institute, statistical classifications are modeled using a complex structure of multiple overlapping hierarchical data structures in an abstract model called Cristal. The basic structure of this model is an hierarchical directed acyclic graph (DAG) that models the part-whole relations between the entities in statistical classifications, extended with another DAG structure that models the element-set relations between the entities in the classifications.

In this thesis we provide a novel visualization technique composed of the treemap visualization technique and the Venn-diagram technique in combination with a force-directed layout algorithm in order to visualize the complex classification structures in a Cristal. Important aspects of the visualization are the combination of a hierarchical structure and a graph structure, as well as the usage of various perceptual cues that take advantage of the human visual perception system.

Additionally, we have created a visualization tool that implements the proposed visualization technique. To test the functionality and usability of the visualization method, we have conducted a user experiment with a number of participants.

Contents

Chapter 1

Introduction

At Statistics Netherlands (SN), a large amount of data about the Netherlands is collected, edited, estimated, aggregated and finally published in the form of tables. During the most recent years, the policy at SN has been to gather all information that has been published into one big database, called StatLine. Due to this policy, StatLine has become a database with a big collection of independent tables with information concerning all possible topics. The StatLine software development team has first searched for the most generic form in which the individual tables can be structured. As a next step, it has searched for ways in which the relations between the individual tables can be made more generic. The result is a generic model called 'Cristal', which can contain all the information of the individual tables, i.e., data and metadata, as well as the relations between the tables. The current implementation of the Cristal model only supports metadata, but support for micro and macro data will be added in the future. Section 2 gives a detailed explanation of the Cristal model as an addition to the following short description of the model.

The acronym 'Cristal' stands for **C**ubic, **R**aw or **I**ntermediate **ST**atistic**AL** information. The name shows that the model is not only meant to hold published data (often structured in 'Cubes'), but that unpublished data ('raw' and 'intermediate' data) can be structured by means of the Cristal model as well. This means that virtually all types of statistical information, such as micro data taken from questionnaires or registers, intermediate edited data, or partly aggregated data, can be modeled using the Cristal model. Note that an instance of information modeled by means of the Cristal model is called a Cristal.

The information in a Cristal is stored in a structured way. Since statistical information very often contains multiple hierarchical levels, the Cristal model allows this kind of hierarchical structure, however, the elements in a Cristal have to comply with a few basic rules. Currently there exists an implementation of the Cristal model in the form of a dynamic link library, which enforces these ba-

sic rules. An editor is available to perform basic operations on a Cristal, such as storing information extracted from a database into a Cristal, adding information by hand, storing a Cristal, etc. The editor uses a tree structure to visualize the Cristal, but this is not sufficient to show all the relations between different elements in a Cristal. The distinctive hierarchical structure of a Cristal is difficult to comprehend and, therefore, a good visualization is needed to get a clear view of the structure of a Cristal.

As is the case with any visualization tool, navigating through the data is an important factor. The human visual perception system can be a great asset in visualizing and navigation information, when carefully taken advantage of. Knowledge of the human visual perception system is, therefore, required, in combination with viewing, navigating and possibly editing the information.

The structure of the Cristal model is explained in Chapter 2. Subsequently, Chapter 3 outlines the requirements for the visualization. Chapter 4 describes different possible visualization techniques, as well as their limitations. In Chapter 5 we propose a novel visualization technique that addresses the known limitations of existing techniques, followed by an explanation of the developed visualization tool. The proposed visualization technique requires a force model, which is explained in Chapter 6. In Chapter 7 we describe the user experiment we conducted to test the visualization tool. Subsequently, Chapter 8 addresses the possibilities and the limitations of the visualization technique, as well as possible areas of further research.

Chapter 2

The Cristal model

In an effort to stay ahead of the competing statistical institutes, it is strategically important for Statistics Netherlands (SN) to focus on the special added value that national statistical offices can offer. Key factors are probably the quality and the coherence of the statistical information they supply. Hence national statistical institutes should further invest in the quality and coherence of their information to keep ahead of the competition.

For several years, SN has stored all the published statistical information in the output database. As such, this single database contains all publications of the past years. Moreover, the information in this database is freely accessible for everyone on the Internet through a system called StatLine. However, the quality and especially the coherence of the data in the output database is still moderate, due to the fact that statisticians all create their own publications that each have different structures and, therefore, can not be combined although the topics are often related.

The output database contains a heterogeneous collection of independent statistical cubes. In short, a cube is a multi-dimensional collection of information regarding the same subject. An edge of a cube contains a hierarchical classification regarding a certain subject, while the center of the cube contains the data that relates the different subjects at the edges of the cube. Clients can select subsets of cubes to display in statistical tables. The statisticians who enter the cubes in the output database are completely free to introduce their own definitions, statistical units, classifications and target variables, independent of the other cubes. Since clients become more demanding, their request for relational information increases. Clients want to combine neighboring cubes, so they can view more relations between certain areas of interest. However, since the statisticians are free to create their own definitions and classifications, often the edges of the cubes address the same subject but have a different classification structure. To glue the different cube-edges together, more information is needed about the specific hier-

archy, statistical units and definitions.

Moreover, clients of statistical information tend to ask for more background information, such as information about

- how the statistical terms are defined,
- how the statistical information relates to neighboring information in the same environment,
- how the statistical information was obtained.

The problem is that this kind of background information is insufficiently available. As with many problems, the solution is not straightforward. The Cristal model [9, 10] attempts to solve this, by providing a model for the relations between heterogeneous kinds of statistical information.

A priori statistical information

Generally, statistical information can be split into two parts: information that is known beforehand (mainly variables, classifications and values) and statistical observations that are derived from the information already known (mainly observations types and observations). The former may also be called *a priori* statistical information and the latter *a posteriori* information. The abstract Cristal model supports both types of information, but the current implementation of the model only supports *a priori* information. Therefore, we only discuss the *a priori* information, i.e., the information structure that is known beforehand, because this concerns the complex graph structures we are dealing with in this project. For a detailed explanation of the *a posteriori* information involving statistical observations, which has not been implemented yet, we refer to van Bracht [9, 10].

The basic elements of *a priori* statistical information are the *a priori categories*, and the *a priori* part-whole relations between them. However, more is needed to describe statistical information: for example, notions of sets of categories and of category trees are required. Therefore two extra objects are introduced in the model: *levels* for sets of categories and *hierarchies* for trees of categories. These different objects are explained in the following sections.

2.1 Categories

In the Cristal model, the term *category* denotes a *predicate*, e.g., something that can be asserted about an object. Examples can be the category ”Amsterdam”,

which is a predicate for the actual region of the Dutch city Amsterdam, or the category ”male”, which is a predicate for a male person or animal.

2.1.1 Category relations

The basic *a priori* relations between the categories in a Cristal are the part-whole relations. There are different kinds of part-whole systems [47], but almost all of them use the general structure of a partial ordering, which is the basic structure for category relations in the Cristal model.

In a part-whole relation, one category plays the role of the part and is called subcategory while the other plays the role of the whole and is called supercategory. The interpretation of this relation between parts and wholes is that if category c_1 is a part of category c_2, then c_1 logically implies c_2. In other words: if c_1 can be asserted about an object, then c_2 can also be asserted about it. This can be illustrated by the following examples.

For example, the predicates ”is in France” and ”is in Europe” can both be asserted about the same object and the first assertion logically implies the second. This is obviously because ”France” is considered to be a part of ”Europe”. However, other situations are also possible: ”is a person” logically implies ”is a creature”, but ”person” is not considered to be a part of ”creature”. Instead ”person” is a specialization of ”creature”. ”is in France” and ”is a person” are subcategories of ”is in Europe” and ”is a creature” respectively.

The partial ordering structure is subject to the following rules, where c_1, c_2 and c_3 are categories:

- $c_1 \geq c_1$ (Reflexivity)
  A category is always a subcategory of itself;
- if $(c_1 \geq c_2) \wedge (c_2 \geq c_3)$ then $c_1 \geq c_3$ (Transitivity)
  If a certain category has both a supercategory and a subcategory, then its subcategory is always a subcategory of its supercategory;
- if $(c_1 \geq c_2) \wedge (c_2 \geq c_1)$ then $c_1 = c_2$ (Anti-symmetry)
  If two categories are subcategory of each other then these two categories must be identical (Anti-symmetry).

The partial ordering on the categories in combination with the above rules can be regarded as a directed acyclic graph (DAG) with a strong hierarchical nature. This is because the part-whole relations between the categories induce a hierarchical structure. Strictly speaking, the structure is DAG, but because of this strong hierarchical structure, we continue to refer to the category-structure as hierarchical DAG.

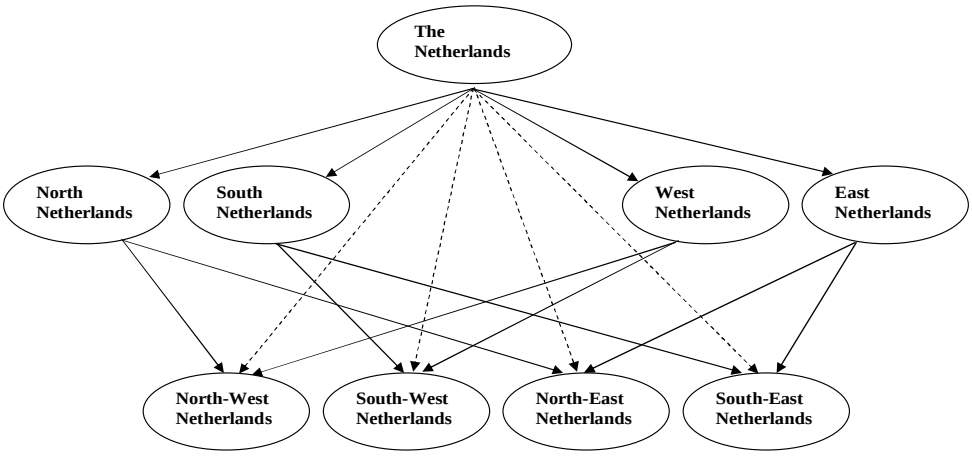


Figure 2.1: *Category subdivision of The Netherlands.*

Let's introduce a running example that is used to explain the structure of the *a priori* objects in the Cristal model. We made a classification structure of The Netherlands, i.e., we made a Cristal. We define the root category to be the predicate "The Netherlands" that denotes the region of the Dutch country, as can be seen in Figure 2.1. Subsequently we divide the total region in two halves, for which we create two additional categories "North Netherlands" and "South Netherlands". After that we divide these two halve countries into quarters of The Netherlands, for which we create "North-West Netherlands", "North-East Netherlands", "South-West Netherlands" and "South-East Netherlands". Subsequently we notice that instead of creating a north-south division, we might as well create a west-east division of The Netherlands. For this, we create two additional categories "West-Netherlands" and "East-Netherlands". These two halves of The Netherlands can be divided into the same four quarters as well.

What we see in this example is that, a category is a predicate, in this case, a predicate that asserts something about a region. Furthermore, there is are obvious part-whole relations between the categories, since "North-East Netherlands" is of course a part of "North-Netherlands". These part-whole relations are denoted by arrows from whole to part. In this case, "North-Netherlands" is the supercategory and "North-East Netherlands" is the subcategory.

Furthermore, the transitive relations are denoted by dashed arrows. Ususally we do not define these relations explicitly, but they are implicitly present in a classification structure in a Cristal. Note that we have omitted the reflexive relations in the picture, but these relations would be shown as an arrow from a category to itself.

2.2 Levels

The previous section about category relation describes the transitivity of category relations without further comment, since transitivity is a fairly obvious property of part-whole relations. However, it is possible that non-transitive relations exist between predicates, i.e., categories. For example, there is an important difference between regions and countries. The difference is that an arbitrary part of a region is again a region (transitive). However, an arbitrary part of a country is not a country again (non-transitive). This is because the countries together form a set of elements rather than a whole of parts. This means that all the regions can, but the set of all countries cannot be modeled by means part-whole relations as explained in the previous section about category relations.

These non-transitive relations are the element-set relations. For example, if a is an element of b, i.e., $a \in b$, and b is an element of c, i.e., $b \in c$, this does not necessarily mean that a is an element of b, i.e., $a \notin c$. A *level* is introduced in the Cristal model to cope with these relations.

Basically, a level is a set of categories, i.e., $L \rightarrow \{c_1, c_2, \ldots, c_N\}$. However, there are certain restrictions on levels that make them not just a set of categories, i.e.,

- $(L_1 \rightarrow \{c_1, c_2\}) \wedge (L_2 \rightarrow \{c_2, c_1\}) \not\Rightarrow L_1 = L_2$
  Two levels can be different, even if they contain exactly the same categories,

- $\forall_{c_1,c_2 \in L}(c_1 \neq c_2) \Rightarrow \neg\exists_{c_3}(c_1 \geq c_3 \wedge c_2 \geq c_3)$
  Two different categories in a level cannot overlap.

Where c_1 and c_2 denote categories and L, L_1 and L_2 denote levels. The first restriction differs from the usual set theory, because in set theory, two sets are equal if they have exactly the same elements. As for the second restriction, in set theory the set $S = \{\{a, b\}, \{b, c\}\}$ contains two sets that contain the same element b, see Figure 2.2. This is perfectly legal in set theory, but forbidden for a level in the Cristal model. The following example clarifies this second restriction.

Suppose that, in analogy with the previous set S, categories a, b, c, x, y and z are such that a and b are subcategories of x, b and c are subcategories of y and x and y are subcategories of z. This situation is shown in Figure 2.2, where the arrows are shown from the supercategories to the subcategories, i.e., from wholes to parts. In this case the set $S = \{x, y\}$ is legitimate, but the level $L \rightarrow \{x, y\}$ is not allowed because x and y overlap, i.e., they share the same subcategory b.

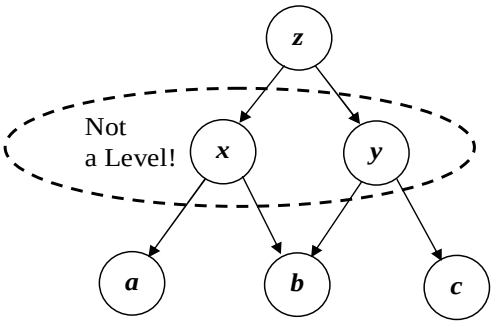


Figure 2.2: *Categories x and y cannot be in the same level because they overlap, i.e., they share category b.*

2.2.1 Level relations

A level can be a sublevel or superlevel of another level, but this is not necessarily always the case. If a level is a sublevel of another level then this level is called a refinement of the other level. This indicates that the refinement relations between levels are partially ordered as well. The refinement relations between the levels can be deduced from the partial ordering of the underlying categories according to the following simple rule: a level is a refinement of another level if every category in that level has a corresponding supercategory in the other level. Note, however, that a superlevel may have categories without any corresponding subcategories in its sublevel.

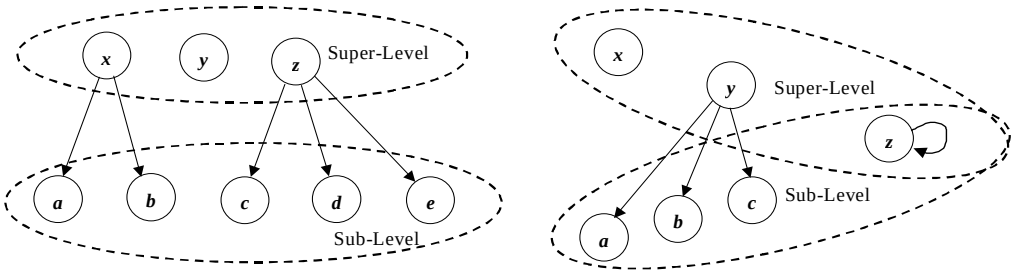


Figure 2.3: *Refinement relations between levels.*

The relations between the levels are expressed in Figure 2.3. In both structures, all the categories in the sublevel have a supercategory in the superlevel and this is precisely the reason for the relation between the sublevel and the superlevel. Note that categories can reside in multiple levels and in this case we call the levels to be overlapping. As showed in Figure 2.3, two overlapping levels can have a

superlevel-sublevel relation as well, since the reflexive relations of the categories can express the fact that a category in one level has a supercategory, i.e., the same category, in another level.

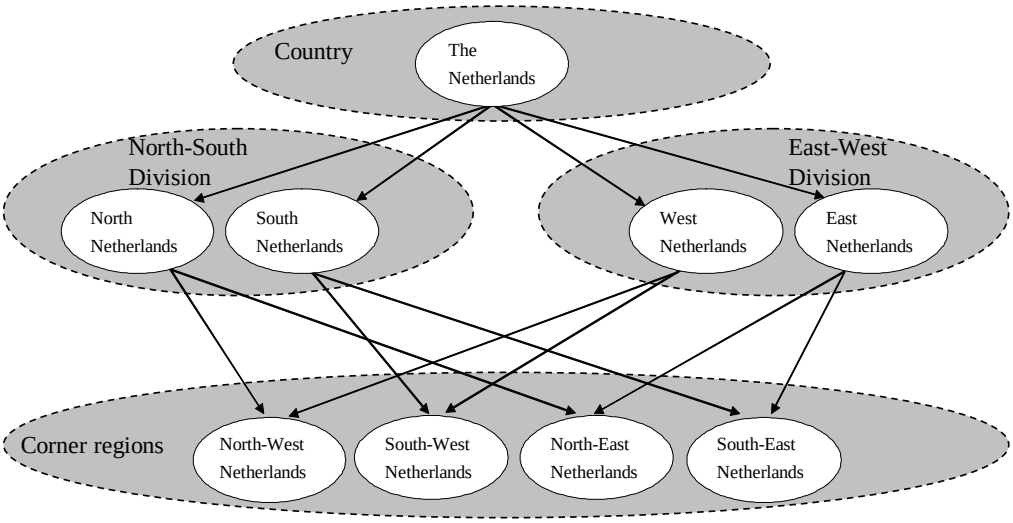


Figure 2.4: *Levels defined on the classification of The Netherlands.*

In our running example we have previously defined a category classification of The Netherlands. In Figure 2.4 we have defined levels on this classification. As stated, a level is a set of categories and in the example we have defined four levels, i.e., ”Country”, ”North-South Division”, ”East-West Division” and ”Corner regions”. Note that it is not possible to create a level that contains the categories ”North Netherlands” and ”West Netherlands”, because these two categories are overlapping, since they have subcategory ”Nort-West Netherlands” in common. As stated, categories in a level are not allowed to be overlapping and, therefore, such a level would not be allowed.

Furthermore, relations can be defined between the levels in the example that are not showed in Figure 2.4. Since all categories in level ”Corner regions” have a supercategory in level ”North-South Division”, the former level is a sublevel of the latter. At the same time, all categories in level ”Corner regions” have a supercategory in level ”East-West Division”, and, therefore, the former is also a sublevel of ”East-West Division”. This means, that the bottom level has two superlevels. Note that the level ”Country” has two sublevels, namely ”Nort-South Division” and ”East-West Division”. Note also that the relations between levels are refinement relations. A sublevel is a refinement of a superlevel.

2.3 Hierarchies

On top of the category and level structure, a hierarchy structure is defined. A single hierarchy supports storage of a tree structure of categories, but a classification structure can contain multiple hierarchies that can share their categories, so the total category structure is a graph. A hierarchy is a sequence of levels in which each next level is a refinement of the previous level, i.e., each next sublevel is different from its superlevel. The order for the levels in a hierarchy must comply with the order of the refinement relations between the levels. As such, the sequence of levels in a hierarchy is not a tree structure, but due to the restrictions on levels, the underlying category structure in a hierarchy is always a tree structure.

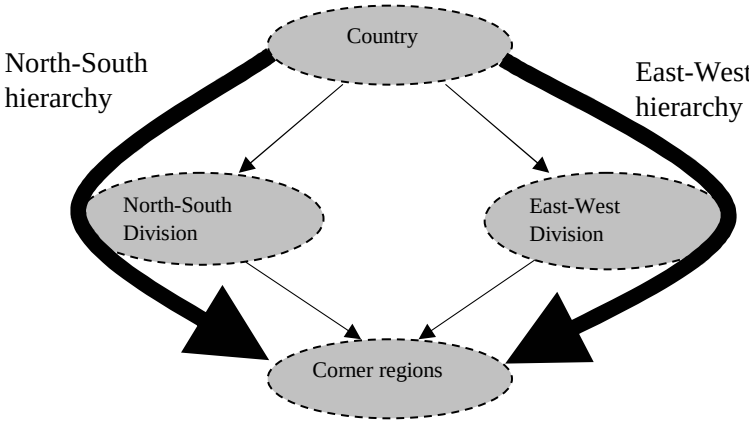


Figure 2.5: *Hierarchies defined on the classification of The Netherlands.*

Our running example has been extended with hierarchies, as shown in Figure 2.5. The levels are shown again in gray with a dashed border. This figure also includes the relations between the levels as arrows from superlevel to sublevel. We have defined two hierarchies on the classification of The Netherlands, namely, ”North-South hierarchy” and ”East-West hierarchy”. The thick black arrows denote the hierarchies as an ordered sequence of levels. Multiple hierarchies are possible, but usually we are only interested in hierarchies that range from a top level to a bottom level. In our example, only two hierarchies from top level to bottom level are possible, and they are both shown in the figure.

Note that, when the category structure is closely examined (see Figure 2.1), the graph of categories really consists of two tree structures that have been placed on top of each other. One tree structure is a division of the netherlands via north-south into four corners, while the other tree structure is a division of the netherlands via east-west into four corners. These two tree structures are represented by the two hierarchies we have defined previously. Therefore, a single hierarchy can model

a tree structure of categories, but multiple hierarchies model a graph structure of categories.

2.4 Variables

The Cristal model has two main kinds of variables: simple variables (that can be numeric or textual variables) and classification variables. A simple variable is simply based on a single set of values, while a classification variable is based on a hierarchical structure of categories with extra support for multiple levels and hierarchies.

2.4.1 Simple variables

The simple variables in the Cristal model support sets of values. These can be numerical values, such as ”1 meter”, ”2 meter” and ”3 meter” (including the measurement units), but can also be textual values, such as the addresses of each inhabitant of a country. In the current implementation of the Cristal model these simple variables are split up in numerical variables and textual variables.

The values of a simple variable are ordered, but it is assumed that the order relation is not equal to a relation between parts and wholes. In other words, even though ”14 meter” $<$ ”23 meter”, ”14 meter” is not considered a part of ”23 meter”.

Moreover, in case of numerical values with a single measurement unit, the measurement unit is often omitted to make it possible to manipulate any two values by addition, subtraction, multiplication and division. Then, usually, the result of the manipulation is again a value in the numeric simple variable.

In practice, the simple variables are often extended by extra non-numeric values like ”not a number”, ”infinity”, ”missing”, ”not possible” and so on. Unfortunately, these extra values complicate ordering, addition, subtraction, multiplication or division.

Theoretically speaking, the values of simple variables should also be treated as categories. However, since these values and their ordering relations are considered trivial, no category structure is being constructed for the simple variables.

2.4.2 Classification variables

The, for this project, more interesting variables are the classification variables. As opposed to a simple variable, a classification variable is a complete system of categories, extended with levels and hierarchies. It can contain any combination of the three entities previously described:

- A collection of partially ordered *categories*. If a category is in a classification variable, then all its subcategories are in that same classification variable as well,

- A collection of partially ordered *levels* defined on the partially ordered categories. The ordering of levels is deduced from the ordering of the categories,

- A collection of *hierarchies* defined on the partially ordered levels. Each hierarchy in the collection is a sequence of partially ordered levels, complying with the ordering of the levels.

To relate with our running example, the complete structure of categories (Figure 2.1), levels (Figure 2.4) and hierarchies (Figure 2.5) can be stored using one classification variable.

2.5 Identification and attributes

Every object in the Cristal model, i.e., a category, a level, a hierarchy and a variable, has at least a *name*, a *key*, a *description*, a *startdate*, an *enddate* and a *globally unique identification* (GUID) number. These are the basic attributes of each object. The set of attributes can be extended in two ways. They can be extended with any other kind of user defined attributes and they can be extended for their translations in any other language.

As expected, the *name*-attribute denotes a short textual representation of the object. The *description*-attribute contains a longer textual description of the object, and the *guid*-number is used as an identification number. The *key*-attribute is a special attribute, in the sense that it is used to support different versions of the same objects. Since objects are identified with their *guid*-number, this means that two objects with the same *guid*-number are exactly the same. It is possible to create a new object that represents a new version of another object and thus very much resembles this other object, but this would result in a new object with a new *guid*-number. To be able to track these different versions of objects, the *key*-attribute is used. Objects that have the same *key* are considered to represent different versions of the same object. Since only one version of an object can be valid at any given time, the *startdate* and *enddate* attributes are used to denote the time frame in which the object is valid. Note that these time frames can not overlap for objects that have the same *key*. The implementation of the Cristal model guards these restrictions and throws exceptions when these restrictions are violated.

2.6 Summary

The most interesting part(s) of the Cristal model are the classification variables. These are the entities that contain the entire complex structure. To summarize, from the top to the bottom, an instance of the Cristal model, i.e., a Cristal, can contain classification variables, these classification variables can contain hierarchies. Hierarchies are in effect ordered sequences of levels, or tree structures of categories. Levels are sets of categories that have no part-whole relation.

Speaking from bottom to top, a Cristal contains a hierarchical directed acyclic graph (DAG) of categories, on top of which the levels are defined as sets of categories. According to the partial ordering of the categories, the levels are partially ordered as well. An ordered sequence of levels is a hierarchy and multiple hierarchies can form a classification variable.

Note that although the structure of categories is a hierarchical DAG, it is fairly possible for an instances of the Cristal model that the category structure is a hierarchical tree, for instance when there are no categories with multiple supercategories.

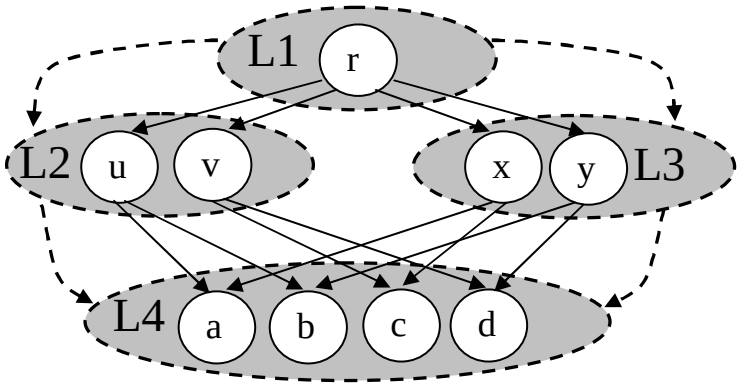


Figure 2.6: *The total structure of a classification in a Cristal.*

Additionally, Figure 2.6 schematically shows the total structure of the running example used throughout this chapter. The small white circles with the lowercase characters denote the categories with the relations between them showed by arrows. The gray ellipses with dashed border denote the levels, ranging from $L1$ to $L4$. The relations between the levels are represented by the dashed arrows. In order to make the figure not anymore complicated, a textual representation

of the hierarchies is shown below the figure. Hierarchy $H1$ represents the sequence of levels $L1, L2, L4$, while hierarchy $H2$ represents the sequence of levels $L1, L3, L4$.

This complex structure makes it possible to relate similar classifications taken from different statistical tables. Furthermore, thanks to the fact that almost all entities can store their descriptions as attributes (in possibly several languages), it is always clear how the statistical terms are defined. Moreover, through these attributes, different versions of entities can be defined, as well as followers an predecessors for entities. Since one of these attributes is an always unique *guid*-number which is used for identification, it is always clear whether two objects represent the same entity in the classification, because they have the same *guid*-number.

Chapter 3

Requirements

As stated in the Introduction (see Section 1), the program should be able to create a comprehensive visualization from the information in a Cristal, in special the information in a classification variable (see Chapter 2), because it is the most complex part of the Cristal model. Since a classification variable contains a lot of information, the visualization is not trivial, especially because the structure of the information is quite particular, as explained in Section 2. During the time the Cristal model was developed at SN, a number of people have worked with the first versions of the implementation of the model. The following requirements for a visualization tool are based on these first experiences with the Cristal model, but also on the developers' vision on the usage of the Cristal model. First we outline the requirements and the last section of this chapter (Section 3.5) gives a numbered list of the requirements including their priorities.

3.1 Elements to visualize

The most important purpose of the visualization is to clarify the relations between the categories, which has, therefore, top priority. A little less important is the visualization of the levels and their relations with the categories. Also, the visualization of the changes in the category structure throughout the time has some degree of importance. Furthermore, the visualization of simple variables has least importance. Of course, the visualization should contain as much information as possible, but too much information would only clutter the screen and does not improve comprehension. There is a certain trade-off here which favors the visualization of the categories.

From this, we can create the following requirements:

- RE1: Visualization of categories (top priority),

- RE2: Visualization of levels (medium priority),
- RE3: Visualization of hierarchies (low priority),
- RE4: Visualization of simple variables (low priority),
- RE5: Visualization of changes in time (medium priority).

The average classification contains up to 1,000 categories, but classifications with up to 20,000 or 30,000 categories are currently present in the database, therefore, the aim is to visualize this large number of categories. It is obvious that the performance will probably not be very good with this number of categories.

As stated in Chapter 2, the structure of categories is partially ordered. In practice, the path from root to leaf is never over 10 categories long. Typically, each category has about 5 subcategories, except the supercategories of the leaves. Such categories often have about 20 subcategories. When drawn as a hierarchical graph structure, the graph would be very wide, but not very deep.

The following requirements can be defined:

- RE6: Visualization of up to 1,000 categories comprehensively (top priority),
- RE7: Visualization of up to 40,000 categories (medium priority).

3.2 Usability and functionality

The target group of users will be statisticians who will use the program to get an overview of the statistical metadata they are researching. The statisticians are used to work with a computer, but, nevertheless, the user interface of the program should be easy to use. Navigating through the structure should be easy and mostly intuitive.

The following requirements can be defined from the previous:

- RU1: Easy to use for statisticians (top priority),
- RU2: Easy navigation (top priority).

The main function of the program is to visualize the contents of a Cristal, in special the contents of classification variables. The program will be mainly used by statisticians that enter and manipulate the information in the model. They have to get a detailed overview of the structure of the information to better understand it, which in turn helps them to draw conclusions about it. Currently, not all statisticians use the Cristal model to store their classification structures. To increase uniformity, the aim is to get all statisticians to work with the model. But for this

to happen, the statisticians all have to understand the rather complex model. As a result of this, the visualization should also aid in the initial understanding of the abstract Cristal model.

From this, the following requirements can be extracted:

- RF1: Aid in understanding of specific classification structure of a Cristal (top priority),
- RF2: Aid in initial understanding of Cristal model (top priority).

Following from this, it should be possible to write the images the visualization creates to disk in a widely used file format (.jpg or .bmp), for instance, to use within a presentation about the Cristal model. Furthermore, animation of the movements can be used to increase comprehension during navigation, but this is not top priority. In the case animation is added to the visualization, storing the animation in a widely used file format (.avi) would be a great asset for use in presentations about the Cristal model.

Currently, a program exists to enter the information, an in-place editor is, therefore, not required, but could be added with low priority to increase functionality and usability.

Requirements following from this:

- RF3: Use animations to increase comprehension (medium priority),
- RF4: Write images to disk in a widely used file format (top priority),
- RF5: Write animations to disk in a widely used file format (low priority).
- RF6: In-place editor (low priority).

Furthermore, the main goal of every visualization is to improve the comprehension of the displayed information. As the amount of information or the complexity of the information increases, it gets harder to obtain insight in the information and to be able to use the information in a suitable way. Speed plays an important role. When the user can obtain insight in the information set quicker, he/she is able to use the information quicker which will result in a quickly found solution for the problem.

Of course, the above is true for all visualizations. Since the designer of a visualization tool wants to influence the actions and decisions the user makes upon using the tool, we continue this section by defining requirements according to the intended actions and decisions.

The full Cristal package not only includes the Cristal model in the form of library that can be used by programmers to include in their projects, but also a

Cristal editor. This editor, the Variable Editor (see Appendix B), can be used to create and/or modify a Cristal. Since a user of the editor has to gain insight in the opened Cristal to be able to edit it, the Variable Editor contains a tree view of hierarchies, levels and categories as well as a properties grid that contains specific properties of the selected object, see Figure B.1.

The intention of the visualization tool is to help in the understanding of the Cristal model. The current tool available, the Variable Editor, has certain limitations, for example that the graph structure of categories is transformed into multiple tree structures which requires certain categories to be present in multiple trees. We, therefore, want the visualization tool to be more comprehensive to the user than the Variable Editor. This would mean that the visualization tool is an improvement, which is of course the intention of the designer.

Furthermore, we want the user to use the easiest possible data structure for their information sets, i.e., they use a tree when the information can be modeled using a tree structure, otherwise use a graph structure. Therefore, we want tree structures to look more appealing then graph structures.

Moreover, we want the user to create data structures with the least amount of errors. One type of error is the use of unnecessary objects. Since they are unnecessary, they can be left out of the information set, which makes the information structure less complex. Another error, which we want to address, is the number of objects that are invisible in the existing editing tool, the Variable Editor. They have to be pinpointed by using the visualization tool and subsequently be removed from the Cristal.

The requirements following from this:

- RF7: Improvement over Variable Editor (top priority),
- RF8: Use easiest possible data structure (top priority),
- RF9: Avoid unnecessary objects (top priority),
- RF10: Show objects invisible in Variable Editor (top priority).

3.3 System

Since the visualization tool will be used in an office environment on desktop systems, it should be aimed to run on a high-end desktop system. The current implementation of the Cristal model is in the C# programming language, so the visualization program should be implemented in C# as well. The program should run on the Windows operating system, extended with the Microsoft .NET runtime libraries. Since usability is important and the users are familiar with the Windows

operating system, the program should look and feel like a Windows program. The program should be desktop-based, not web-based.

Following from this, the next requirements can be defined:

- RS1: Visualization should run on high-end desktop system (top priority),
- RS2: Tool implemented in C# (top priority),
- RS3: Windows look and feel (top priority).

3.4 Language

It should be possible to select the language of the interface of the program. The program should support at least Dutch and English. A lot of Cristals are stored in multiple languages, so the program should also be capable of displaying the Cristal in the selected language.

The following requirements are defined from this:

- RL1: Support interface in multiple languages (top priority),
- RL2: Support display of Cristal in multiple languages (top priority).

3.5 Requirement matrix

The following table lists the previously detailed requirements. Each requirement is given a number for easy reference as well as a priority. The numbering, as well as the priority ranking is explained in Table 3.2.

Number	*Requirement*	*Priority*
RE1	Visualization of categories	1
RE2	Visualization of levels	2
RE3	Visualization of hierarchies	3
RE4	Visualization of simple variables	3
RE5	Visualization of changes in time	2
RE6	Visualize up to 1,000 categories comprehensively	1
RE7	Visualize up to 40,000 categories	2
RU1	Easy to use for statisticians	1
RU2	Easy navigation	1
RF1	Aid in understanding of specific structure of a Cristal	1
RF2	Aid in initial understanding of Cristal model	1
RF3	Use animations to increase comprehension	2
RF4	Write images to disk in a widely used file format	1
RF5	Write animations to disk in a widely used file format	3
RF6	In-place editor	3
RF7	Improvement over Variable Editor	1
RF8	Use easiest possible data structure	1
RF9	Avoid unnecessary objects	1
RF10	Show objects invisible in Variable Editor	1
RS1	Visualization should run on a high-end desktop system	1
RS2	Implemented in C#	1
RS3	Windows look and feel	1
RL4	Support interface in multiple languages	1
RL5	Support display of Cristal in multiple languages	1

Table 3.1: *List of requirements.*

Number	*Type of requirement*
RE	Requirements on elements to visualize
RU	Usability requirements
RF	Functionality requirements
RS	System requirements
RL	Language requirements

Priority	*Explanation*
1	Top priority
2	Medium priority
3	Low priority

Table 3.2: *Legend for list of requirements (Table 3.1).*

Chapter 4

General information visualization techniques

As stated in Chapter 3 in combination with Chapter 2, the structure we want visualize is a hierarchical directed acyclic graph (DAG), namely the graph of categories. On top of that, we want to visualize different relations between the categories, namely the levels, and we want to visualize the changes in time for the categories. To comply with these requirements, we want to create a new visualization tool or enhance an existing visualization tool.

Therefore, we first try to define the important factors of a visualization in combination with the type and structure of the information we want to visualize. After that, we discuss a few existing visualizations that are able to visualize the multiple overlapping hierarchical classification structures we are looking for, followed by a conclusion.

4.1 Introduction

In general, data visualization is used to gain insight in data, where data is defined in the broadest sense possible. Wherever information is being handled, visualizing this information can be a great asset to be able to comprehend the matter. Data visualization is applicable to a large number of areas, from medical appliances to physical data models, from a file system to statistical data, for example at SN. Usually visualization of data with an inherent geometry (like molecular or scanned medical data) is called scientific visualization; visualization of abstract data like tables, trees, graphs and also Cristals, is called information visualization.

There are three important factors in a visualization, namely the human visual perception, the display itself and the interaction technique. These three components have to be chosen carefully to get the greatest revenue out of the visual-

ization. Of course, the type of information that has to be visualized plays an important role in the choice of these three components. To serve as an overview, Gershon et al. [23] have created a wide-ranging taxonomy on topics about human visual perception, display techniques and interaction techniques. Often, the boundaries between these three topics are fuzzy, because in a typical visualization, these three components need to work closely together to build up the visualization. The tutorial of Gershon et al. partitions the display technique category into sub-categories such as *hierarchies*, *node-and-link* and *other*, while the interaction techniques can be classified into the categories *focusing*, *filtering* and *linking*. Since this project concerns the visualization of multi-dimensional hierarchical graph structures, we focus on the *hierarchies* and *node-link* categories.

4.2 Human visual perception

As stated above, an important part of a visualization is the human factor, more specifically, the properties of human perception of visual information. Perceptual cues all have a different way of affecting the human visual perception system. Since these perceptual cues are important for humans, visualization can make use of these cues to their advantage. A lot of research has, therefore, been done [51] that provides experimental evidence on which visualization techniques are, or should be, based. Most of the literature regarding this subject concentrates on the effects of motion and color.

The power of basic visual attributes such as color, size and brightness comes from the fact that they are part of so-called *pre-attentive cues*. These cues are visual attributes that immediately ”jump out” of the image or scene. The human visual processing system unconsciously marks these elements. Visual properties can, therefore, be split into two categories, one with the pre-attentive cues and the other with the visual properties that require an effort to be recognized. The two most important pre-attentive cues are motion and color, where color is defined in the broadest sense of the word, including texture, hue, intensity and contrast. The human object-detection system is mainly based on motion-detection, in combination with detection of colors. In general, when something with a distinct color than the surrounding moves with a different speed than the surrounding, we humans mark that something as an object. Therefore, motion and color are two important cues in a visualization, which are elucidated in the following sections.

4.2.1 Motion

Since motion is an important cue in human life, various studies have investigated the ability of motion to increase the power and comprehension of information vi-

sualizations. For example, Hubona et al. [34] and Ware and Franck [52] have investigated the possibilities of using motion as an aid for dis-entangling otherwise ambiguous 3D graph structures. Their findings show that rotation of a 3D network made its structure much clearer than with static representations or even stereoscopic-depth views of the static structure.

Fundamental research into the properties of motion has been done by Bartram [2], who has explored motion as a technique for encoding further dimensions of an information set. Bartram states that perceptual properties such as harmonic oscillation, blinking and frequency of movement are suitable for pre-attentively representating some types of information. Later, Bartram expanded this by theorizing that objects oscillating in a common phase have an ability to stand out as a group from other objects [3].

Finally, Bederson and Boltman [7] show that for some simple tasks, animating between viewpoints in an abstract information display improves the ability of users to reconstruct that information space. In effect, animation allowed the users to clarify their mental representations of the information without any diminishing performance, which reflects the work of Ware and Franck [52] for 3D structures.

4.2.2 Color

Color is very important in identifying groups of objects and, therefore, using the right color for a visualization is an important topic in information visualization. For example, the background color can easily affect the perception of foreground colors, therefore, a neutral background should be chosen when possible.

Color itself can be regarded as three dimensional. These dimensions can be red, green and blue (RGB), or hue, saturation and value (HSV). The dimensions might as well be obtained from other color spaces, such as the LUV color space. The LUV space is designed to be a perceptually uniform color space, where the L value encodes perceived luminance, i.e., the lightness, and combinations of the U and V values define chromacity, i.e., the actual color. Within any of these color spaces, a specific color will map to a specific point in the 3D space.

Since the RGB color space is perceived by humans as non-linear, Levkowitz and Herman [39] describe a color scale that correct this non-linearity of the RGB scale in human vision. The distance of two colors on their linearized color scale is proportional to the perceived difference between the two colors.

Other perceptual and physiological advantages and pitfalls of color were described by MacDonald [40]. Since the human eye is less sensitive to blue than it is to green or red, fine detail should not be displayed in blue, according to MacDonald. Color, like other perceptual cues, can be misleading when it is used without proper consideration of its properties.

Furthemore, experimental studies by Healey et al. [30, 31] have demonstrated the conflicting effects of several perceptual visual cues, such as color, shape and motion. They found that the use of color tends to dominate and effectively mask information displayed by shape. Such perceptual conflicts should, of course, be avoided as much as possible.

These experimental findings provide a basis for developing information visualization systems that take advantage of the properties of the human visual system. However, one of the criticisms of visualization is that its researchers and practitioners don't take the perception research sufficiently into account. Mackinlay [41] underscribes this and states that understanding and taking advantage of the interaction between perception and cognition will be a major topic for information visualization in the future.

4.3 Display techniques

Usually information is abstract, but it is often structured in a certain way, for example a network or a hierarchy. Different display techniques deal with the size, layout and legibility on the screen. Since we want to visualize multiple overlapping hierarchy structures, we do not follow Gershon et al.'s categorization, but we focus on the three most relevant information types for this type of data structures: hierarchies, graphs and networks (node-link data), and multi-dimensional information.

4.3.1 Hierarchies

Since the graph of categories is a hierarchical structure, a relevant research area is the area of hierarchy visualization. A hierarchical structure is an efficient way of storing, classifying and manipulating objects. It requires the objects to be grouped and sub-grouped according to common attributes. In practice, information is very often hierarchically structured, because a lot of information can be grouped and sub-grouped. Therefore, we find hierarchical information in file systems, document classifications, taxonomies, organizational structures and sports league structures, for example. As a result, a great deal of effort has been put into the visualization of these hierarchical data structures in the field of information visualization.

Hierarchies tend to be strictly trees, i.e., structures in which an object, or node, has exactly one parent object/node (except the root, which has no parent). However, this is not always the case, as some more complex structures can be hierarchical too, such as the classification structure in a Cristal. Furthermore, these structures occur with multiple taxonomies on the same subject, for example in the

botanical world. When taxonomists study and classify organisms to create a classification hierarchy that shows the organisms natural relationships, they group the organisms into different taxa, i.e., categories. These lower level taxa are placed in higher level taxa according to some criteria, and this structure forms a hierarchy. Different taxonomists create different taxa, but at the lowest level they are all classifying the same organisms. When the taxonomies of different taxonomists are combined, a clear hierarchical structure is formed, but it is certainly not a tree structure. In short, a hierarchy can also contain elements or nodes that have more than one parent. However, we discuss representations suitable for hierarchical trees first.

The two classic approaches to drawing tree structures are the node-link and nested box representations. The node-link representation generally visualizes nodes within the tree structure as small boxes, connected by lines that reflect the logical connections, or links, between the nodes. The nested-box representation, on the other hand, represents trees by drawing ”child” nodes as smaller boxes within a larger box representing the ”parent” node that contains it. This process is then carried out recursively, dividing the smaller nodes to be able to fit their child nodes and so on. The latter is often called a ”tree-map” representation.

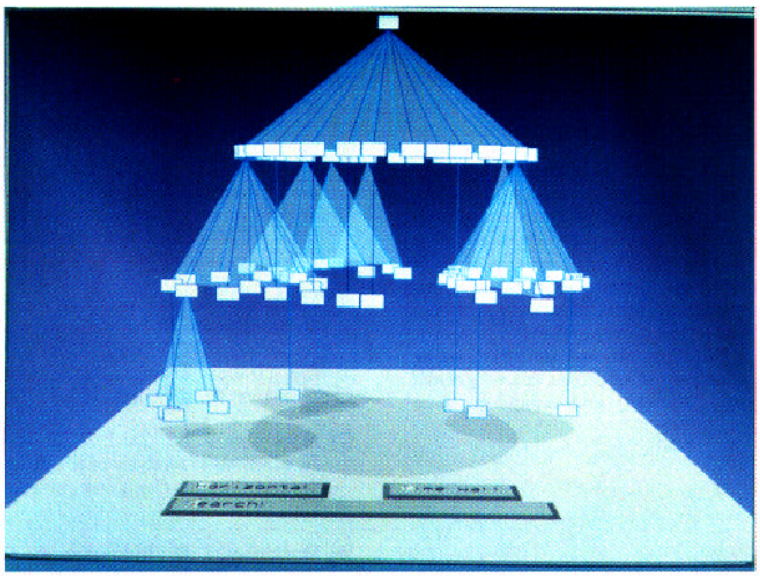


Figure 4.1: *Cone tree by Robertson, Mackinlay and Card [44].*

An early hierarchical tree information visualization was called ”Cone Trees” by Robertson, Mackinlay and Card in 1991 [44], developed as part of the Information Visualizer project at Xerox PARC [12]. A tree structure is displayed in three dimensions in an attempt to increase the number of nodes that can be presented

on the screen, as shown in Figure 4.1. The links from a particular node to its child nodes form a translucent cone. The child nodes are arranged uniformly around the cone's base. Selecting any node brings that node to the front of the view of the cone tree in an animated sequence. The use of animation preserves the users' mental model of the visualization while the repositioning of the nodes takes place. As stated in Section 4.2.1, an animation is by far superior to the alternative abrupt moving to the final positions. However, cone trees suffer from viewing problems caused by occlusion [1] in the 3D representation, as seen in the figure. Still, its appearance and possibilities gave other researchers inspiration to investigate how this visualization method could be improved or extended. Jeong [36] adapted the cone trees to tackle the occlusion problem by using reconfigurable discs instead of cones. Others, namely Tversky et al. [48] and later Carrière and Kazman [13], studied and began to resolve the problems of enhancing the perceptual cues used in cone trees, as well as advanced filtering and focusing mechanisms.

Aware of the fact that traditional node-link tree diagrams waste approximately half of the screen space, Johnson and Shneiderman developed Treemaps [37] as a space-efficient approach to increase the size of trees that could be displayed on the screen.

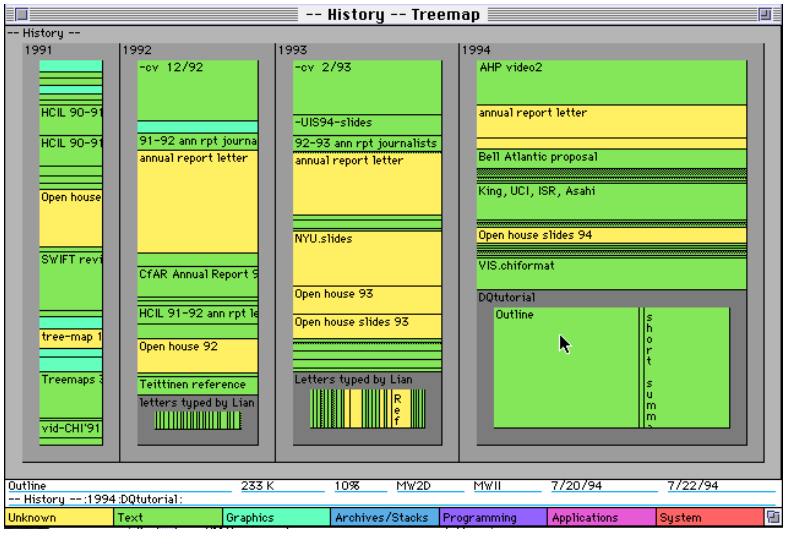


Figure 4.2: *Nested tree-map layout by Johnson and Shneiderman [37].*

Tree-maps, an example of which is shown in Figure 4.2, use a representation with nested boxes in which a specified screen area is divided according to the

[1] Occlusion occurs when two objects in 3D space are projected on top of each other on a 2D screen. The nearest object is occluding, i.e., blocking the view to the farthest object, and is, therefore, called the occluder.

number of root nodes in the tree. These root node areas are then sub-divided according to the number of children of each node. This process continues recursively down to the leaf nodes. The relative size of each area is calculated according to a property of the data. The original application was used to visualize a file system structure, and used the file and directory sizes to calculate the size of each area. This approach is extremely space-efficient, because it uses all of the space available in the original area, but the leaf nodes are very prominently visible as opposed to the internal structure of the data. To solve this problem, the internal nodes can be given borders to make it more clear that they reside underneath their child nodes, but this obviously reduces the space available for the display of the child nodes.

Figure 4.3: *Cushion tree-map layout by van Wijk and van de Wetering [53].*

Tree-maps have proven to be, like cone trees, successful enough to unleash further investigations into its usage. The use of tree-maps in other data domains has been explored, for instance in analytical decision charts by Asahi et al. [1]. Various extensions and refinements to the original design have been developed, such as using 3D effects to improve the perception of the structural depth by van Wijk and van de Wetering [53], and improvements on the aspect ratios of displayed leaves by Bruls et al. [11]. Lately, research has focused on layout algorithms that combine stability and pleasing aspect ratios between the nodes in a tree-map [45] and layout of objects of bounded minimal size [6]. A pleasing aspect ratio is a height/width ratio for the rectangles in a tree-map that is close to one, such that the rectangles are close to squares. Recently, Engdahl, Köksal and Marsden [18]

use tree-maps to visualize threaded discussion forums on PDAs.

4.3.2 Node-link structures

As stated we want to visualize the graph of categories. A graph structure is usually known as a node-link structure. The nodes, as with trees, represent individual objects and the links represent the relations between them. Many different types of graphs exist. The least complex graphs are the trees, which have their own specialized visualization approaches as previously described. A little more complex are the Directed Acyclic Graphs (DAG) relevant in this project. Multigraphs and full, general graphs are the most complex graphs. Herman, Melançon and Marshall [33] describe the common graph structures in detail, including their associated visualizations, in their survey paper.

The visualization of general types of graph structures is difficult compared to visualizing trees and simple hierarchies. In general, the problem lies in the fact that a graph is not a simple hierarchy, and, therefore, is not lending itself for a hierarchical layout. To overcome these difficulties, attempts have been made to transform the graph to a type of a tree, which is much easier to visualize. For example, Hao et al. [28] use invisible links between nodes to avoid the links cluttering the screen. They make a number of links in the graph invisible, such that instead of the full graph, a hierarchical tree structure is always displayed, which is a lot more comprehensive. Furthermore, Munzner [42] uses a technique that computes the spanning tree of a graph. A layout for this spanning tree is subsequently calculated using the cone tree method (see also Section 4.3.1) and displayed in 3D hyperbolic space. Using these methods, screen clutter is largely avoided to create a more understandable visual layout.

Another option is to restrict the graph such that it has properties that lend themselves to be visualized more understandable. Bartram et al. developed the Ztree [4] that displays a tree-like graph with the tree portion in a nested tree-map style and the other relations displayed as cross-links between the appropriate nodes (or sets of nodes) in the hierarchical structure. The display of the Ztree can be described as a combination of a pure tree layout, where further properties are revealed through user interaction (this is explained in Section 4.4).

Finally, another main display technique in information visualization for drawing a full graph is to use a self-organising system that positions the nodes and links of the whole graph according to a set of rules. These rules are defined with the idea to produce visually pleasing layouts, but obviously, the final judgment of this is down to the user, not the algorithm. Harel and Koren [29] focus on issues such as reducing visual edge-crossings and promoting visual symmetry in the case the graph has symmetrical properties, to make complex visualizations more understandable.

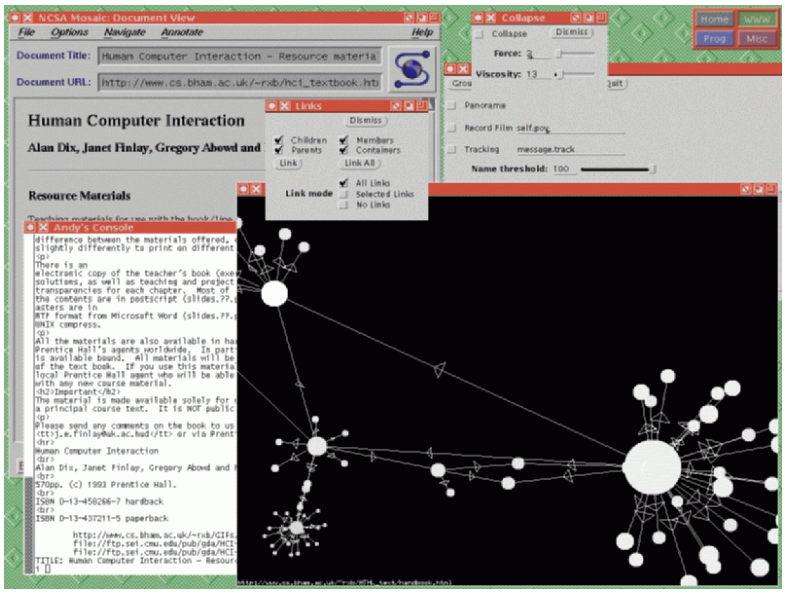


Figure 4.4: *The HyperSpace web viewer [54].*

An example of a graph visualization that is applied to a specific domain is HyperSpace [54] which is displayed in Figure 4.4. HyperSpace visualizes the hypermedia structure of the World Wide Web (WWW) as a graph. Individual pages are shown as circles and links between pages are represented by lines between the nodes in the visual representation. HyperSpace visualizes the graph by letting the user select an area of interest, after which the related pages move closer together and the unrelated pages move further away. In the end a graph with clusters of related pages is formed and displayed.

The type of self-organising structure HyperSpace uses is called a force-directed algorithm that uses a spring-mass model (see the next section). Such layout algorithms are used in many systems such as HyperSpace's successor Narcissus [32]. The effect of this clustering is analogous to the concept of grouping in drop-down menus. Similar items are grouped together and the user recognizes that they are sharing common attributes, because they are positioned closely together.

Force-directed layout

A force-directed layout algorithm calculates the positions of a group of entities according to the links between them. The original method was introduced by Eades [16]. In short, the algorithm works as follows: nodes are replaced by steel rings and the links between them depict springs that have a certain ”strength” or ”stiffness”, as showed in Figure 4.5.

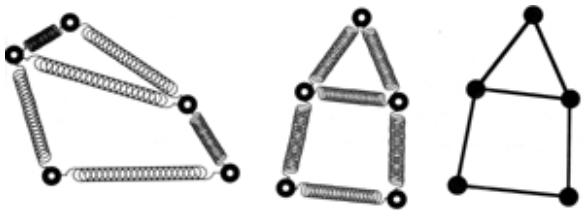


Figure 4.5: *Physical spring model.*

The algorithm iteratively calculates the forces modeled by the springs and the nodes are moved in the direction of the calculated forces to minimize the overall energy of the system, as listed in Section 6.1. This is exactly the way an actual physical system would move. For example, two distant nodes connected by a strong spring would be moved faster and closer toward each other, because of the effects of the spring. Two close nodes move away from each other, since the spring will ”push” instead of ”pull”. Also, often a repelling force between all nodes is used, typically according to the electrostatic forces working on particles with similar charge. These repulsion forces usually help to spread out the nodes for a more pleasing layout and help to avoid overlapping node representations.

The spring-mass model has often been subject to refinements, including those of Fruchterman and Reingold [20], Kumar and Fowler [38] and Gansner [22]. Fruchterman and Reingold have enhanced the algorithm with uniform edge lengths as well as an increased degree of symmetry in the layout, Kumar and Fowler extended the algorithm to three dimensions and Gansner tackled the problem of nodes occluding each other by using unfavorable positioning. Another type of force-directed model is based on a process known as simulated annealing [15]. It is more costly in algorithmic terms, but allows nodes to move in a direction opposing the local force gradient. This allows layouts to escape from positions in which the classic spring layout would be in a local minimum and head toward layouts in which the total energy is closer to the global minimum.

4.3.3 Multi-dimensional information

Since we want to visualize hierarchical multi-dimensional information structures, we discuss solutions from the relevant research area. Multi-dimensional information is information whose objects can have many shared attributes or dimensions. One of the main difficulties regarding visualization of multi-dimensional information is the mapping of these many dimensions to the two dimensions displayable on a computer screen.

A three dimensional projection can help to display one extra dimension of information and virtual reality techniques can increase the depth perception of 3D

visualizations, but, as stated, this only gives one extra spatial dimension. Since multi-dimensional data can have many dimensions, this is often not a big advantage. Moreover, a 3D perspective introduces its own set of problems, such as occlusion and effective use of depth cues. These issues can be tackled using transparency and motion, but, since these drawbacks can not be tackled completely, consideration should be taken whether or not the drawbacks outweigh the benefits while using a 3D visualization.

A straightforward solution to the multi-dimensional problem is to represent information entities as objects on the screen. The dimensions can, subsequently, be mapped to visual properties of these objects on the screen such as position, rotation, color, brightness, transparency and shape. With this technique, a very high number of dimensions can be visualized. An example is the music visualizer by Graves et al. [26]. A problem with this approach lies in deciding which dimensions to assign to which visual property of the displayed objects. This can only be decided by analysing the information in accordance with the user's tasks and deciding what aspects of the information the user is most likely searching for. Another problem is the perceptual ordering, i.e., the human visual perception system marks visual cues as color and motion before other visual cues. It seems, therefore, obvious to map important dimensions to color and motion cues. However, it is not trivial to know what the important dimensions of an information structure are. A solution can be to let the users themselves dynamically assign dimensions to certain visual properties according to the task at hand.

Another approach initially developed by Feiner and Beshers [19] consists of nesting coordinate systems within the points of other coordinate systems. This way only a subset of the actual dimensions or objects present in the information set is viewed at a time. A further approach is to divide the screen into sub-areas in which pairs of dimensions can be compared against each other. This approach uses a number of different viewports where a number of dimensions is displayed in each viewport. The problem of linking the information in the different viewports arises, which is one of the interaction techniques described in the next section.

Inselberg and Dimsdale [35] introduced Parallel Coordinates, a system whereby a number of dimensions are mapped one-to-one to an equal number of parallel axes on-screen. An object in the information set is mapped to a series of points, one per axis, with the position of each point on the axis being dependent on their value in the dimension that is represented by the axis. The points are connected with line segments between neighboring axes, forming a "poly-line" across the set of axes. This process is repeated for each object in the information set.

Using this method, similar objects have similar lines. In later applications using this technique, the axes can be moved about to enable the user to order the dimensions as they see fit. However, one problem is that if two objects share the same value in a particular dimension, they share the same point on the correspond-

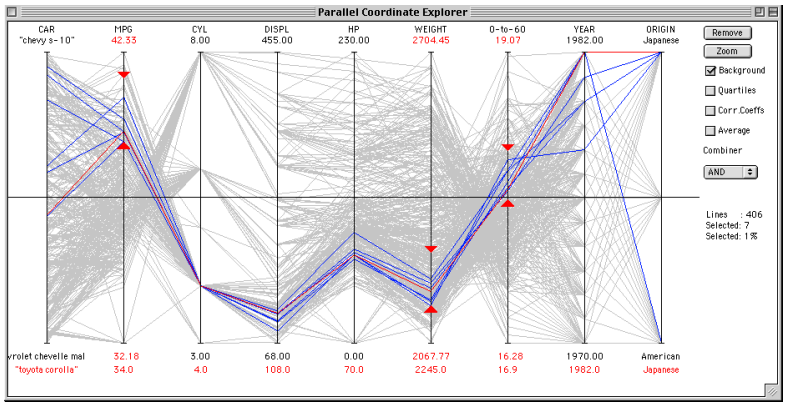


Figure 4.6: *Parallel coordinates as visualized by Siirtola [46].*

ing axis, which results in two polylines that appear to merge and then separate again. Without additional cues, such as color, it is impossible to determine which line is which after the merge and separation effect. Such a situation can be seen clearly in the third axis from the left, labeled ”CYL” in Figure 4.6.

Even while using the previous techniques, there is still a limit on the number of dimensions that can be represented through dimensional nesting, parallel coordinates or simple one-to-one mappings of color, hue, shape, positioning etc. Some high-dimensional datasets can have hundreds of different dimensions. To solve this, Young [50] reduces the amount of dimensions through a technique called Multi-dimensional scaling (MDS). Using this technique, a distance function has to be defined that describes the distance between the information entities over all the dimensions. The information set can subsequently be displayed on a two dimensional screen with the distances between the individual objects resulting from the distance function. To calculate a placement of the objects such that the distances in 2D are as close as possible to the result of this distance function, often a force-directed layout algorithm is used (see Section 4.3.2). Variations exist that reduce the number of dimensions to three instead of two, and subsequently create a two dimensional projection on the screen.

4.4 Interaction techniques

The third topic in Gershon, Card and Eick’s tutorial [23] was ”interaction techniques”. In this topic, they classified the general interaction styles that have been used with visualization. The aims of these techniques are to simplify and aid the user in finding patterns or locating specific details in the information. Of course, these techniques may be implemented differently in different visualizations, but

many share similar underlying principles. Gershon, Card and Eick grouped the interaction techniques into three categories *focusing*, *filtering* and *linking*.

4.4.1 Focusing techniques

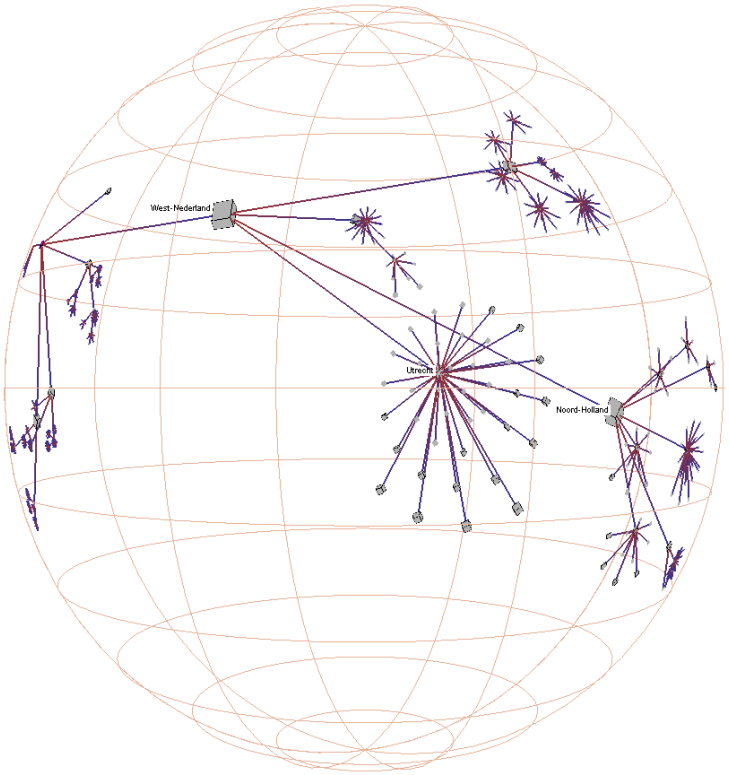


Figure 4.7: *Hyperbolic graph visualization [42].*

Focusing techniques are concerned with the magnification of the display area on the screen. Through the use of focusing techniques, certain areas of the visualization space can be given prominence which in turn increases the detail and comprehension of whatever is displayed there. Furthermore, focusing techniques allow navigation and overview of large information sets while simultaneously allowing detailed inspection of specific pieces of information. In general, focusing techniques resemble the effects of looking through a fish-eye lens or a magnifying glass. The fish-eye view uses the effects of a gradual magnification effect that is centered on a focal point. The magnifying glass, on the other hand, uniformly increases the magnification, which is often called zooming. Also, a combination of both techniques is fairly possible.

A different class of lens viewers take advantage of a non-Euclidean space, namely hyperbolic space. Hyperbolic space has the unintuitive property that infinity can be projected back to a finite point in Euclidean space. As such, all points in a structure in hyperbolic space can be projected back to a bounded area of ”normal” space. In addition, a visually pleasing fish-eye effect is generated with the object at the origin in hyperbolic space displayed at the origin in normal space. However, the equations involved are not trivial and in practice many objects are reduced to such small size that they are not displayed. Munzner [42] has applied hyperbolic lenses to network structures, while extending the hyperbolic distortion to three dimensions (see Figure 4.7).

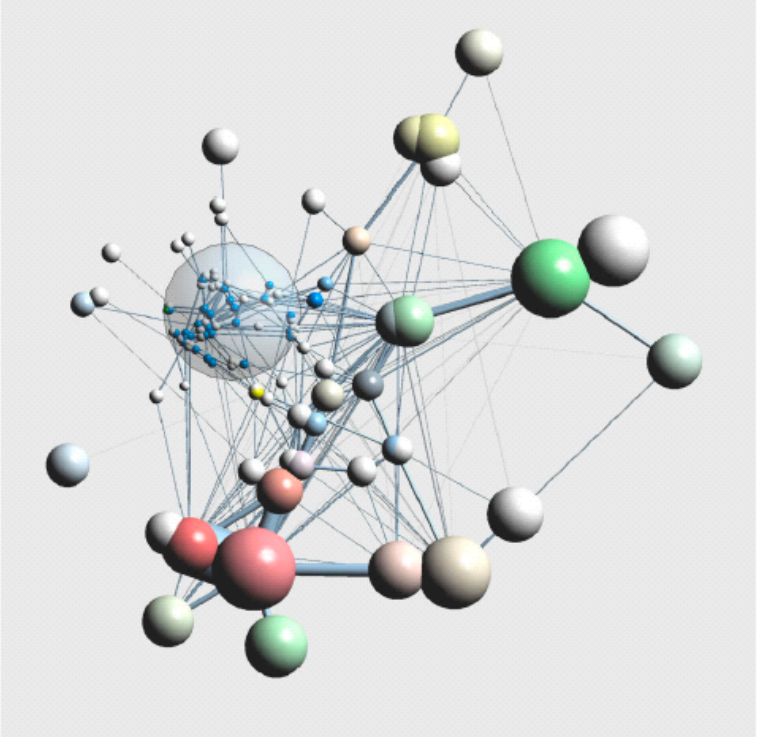


Figure 4.8: *Fish-eye view using a lens [27].*

The second class of focusing techniques are the zoom methods. Opposite to fish-eye views, the entire window is always at the same level of magnification while using zoom methods. However, the effect is still to focus on a particular piece of information to gain detailed insight in that subset of the data, but now the context is lost. Separate overview and detail windows can be linked together which partially solves this problem, but it is less elegant than the lens techniques and requires the user to mentally link the information in the two windows. Bed-

erson and Hollan [8] introduced a semantic zoom in their Pad++ interface. Using this approach, they extended the standard zooming method by changing the level of detail when zooming in or out.

These lens-based focusing techniques rely on transformations on the used coordinate spaces. Another focusing method, where an object's absolute coordinates depend on their neighbors position rather than a global transformation function, relies on functions called Degrees Of Interest (DOI). Furnas [21] describes a DOI as a function that "assigns to each point in the structure a number telling how interested the user is in seeing that point, given the current task". His approach was then to compare these values with a threshold value and thus deciding whether or not an object was shown. An example, which uses an actual lens that represents the area of interest is given by van Ham and van Wijk [27]. Figure 4.8 shows a graph that displays less detail at the periphery while the level of detail increases in the direction of the lens. Under the lens, the level of detail is maximized.

4.4.2 Filtering techniques

Filtering techniques are used when the user wishes to get detailed insight in information that has common attributes or values. Usually the subset of the data that corresponds to the selected conditions is highlighted, or non-corresponding information is removed from the visualization.

Eick [17] describes sliders that define a range in a particular dimension or set of dimensions. The filter subsequently accepts objects with values within the given ranges and rejects other objects. Only the information in the current filter is visualized on the screen. By using the filter, unwanted information is removed from the visualization, which results in a clearer view on the information the user is interested in. Furthermore, it is possible to filter out unwanted information by using transparency and blur effects.

Filtering can also be used on structured information sets, such as a tree structure. For example, the user can set filter conditions that result in the removal of certain subtrees in the data set. What remains are the parts of the tree where the interest of the user lies in. This creates a more comprehensible overview picture of the data when a lot of subtrees have been removed and gives the user the ability to expand the subtree of interest.

Filtering techniques are similar to focusing methods in their aim to increase detail in the area of interest, while reducing the amount of irrelevant and distracting information. In other words, filtering techniques focus a user's attention. But they are different from focusing methods because they affect the display attributes of the visualized objects directly rather than the space in which the objects are visualized. In addition, filters are different from focusing techniques, because they can also be applied to other perceptual cues, such as color, size, visibility and

transparency.

4.4.3 Linking techniques

The third main interaction technique is linking. Usually, linking is important when the visualization involves multiple views on the information set. It is important that an action carried out in one of the views results in a mirroring of that action's results in the other views. This is, for example, the case when the user selects an object or region in one view, which has to result in the selection of that same object or region in the other views. Often, linking is used in combination with multiple scatterplots, as can bee seen in Figure 4.9 from Tweedie et al. [49]. The figure shows the selection of different value ranges on the left that have to correspond with the yellow boxes on the right. A change of selection in any of these views results in an update of the selection in the other views. As with other interaction techniques, this update must be rapid to be useful.

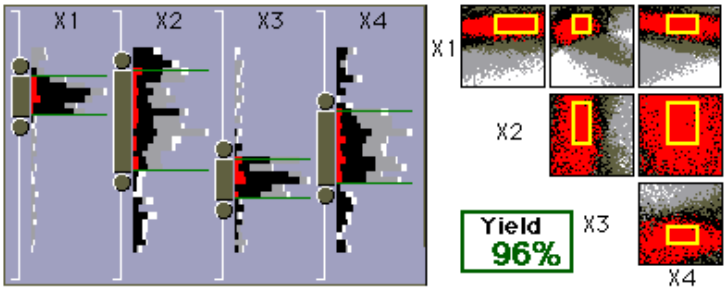


Figure 4.9: *Linking value ranges and scatterplots [49].*

Additionally, North and Shneiderman [43] describe a taxonomy of possible linking techniques between two views. Selection of items in one view results in the same items being selected in another view. Navigation of a view is mirrored in the navigation of another view. When an item is selected, the item is automatically selected in the other views as well and the other views automatically navigate the information set such that the selected item becomes visible. All these tasks can be performed on views on the same data set, or on different data sets.

To conclude, linking techniques can be used for splitting visually crowded visualizations of complex data sets into multiple more comprehensible views. The combination of data and perceptual cues between these many views will allow the user to comprehend the information piece by piece. Furthermore, the users only need to correlate as many views as necessary to acquire the information they need. In the case linking techniques are not applied correctly, it is obvious that the use of multiple views does not lead to an increase of comprehension of the information set.

4.5 Hierarchical graph visualizations

The previous sections describe more general methods for graph visualization, this section describes related work that covers the more specific multi-dimensional hierarchical structures we try to visualize. This specific type of information structure is, for example, found in the botanic world, where taxonomists create classifications that categorise different species of plants and organisms. These classifications are hierarchical structures where specimens are grouped into different taxa. These taxa are then placed in higher level taxa according to some criteria, e.g., DNA relationships or simply shape. The taxa are also assigned to ranks that specify the level of a taxon in a classification hierarchy, such as ”familia” or ”genus”. Taxonomists may use different combinations of these existing ranks, or even develop new ranks, in their taxonomies.

When a taxonomist has created a taxonomy and has published it, the taxonomy is considered a valid classification. If other taxonomists disagree with the classification, they have to create and publish a new classification that reflects their viewpoint. This means that, over time, some specimens may end up classified in different groups in various classifications. These different classifications are all valid, since taxonomists do not have the concept of ”correct classification”. All published taxonomies are considered valid viewpoints.

In the following paragraphs, we discuss a graph-based and a set-based visualization of this kind of multi-dimensional hierarchical graphs from the botanical world, followed by a Venn-diagram visualization method that is used to display categorized files from a file system.

4.5.1 Graph-based hierarchy visualization

Graham et al. [24] have made a comparison between set-based and graph-based visualizations of overlapping classification hierarchies in the botanical field as described above. The first prototype they have created to use in the comparison is a directed acyclic graph (DAG) (see Figure 4.10) that combines the individual hierarchical taxonomies. This could be done, because the information in the hierarchies has a high degree of correlation. Different hierarchies share a large number of categories, so the nodes in the graph remain relatively constant. Mainly the links between the nodes and thus the organisation of the nodes involve major changes between hierarchies.

The program visualizes the DAG structure using the spring-mass metaphor that is commonly used in network visualizations (see also Section 4.3.2). The edges of the graph represent springs between the nodes that move the nodes to their positions. Unlike most spring-mass model based systems, these positions are not final and are constantly recalculated by the system. This is because the

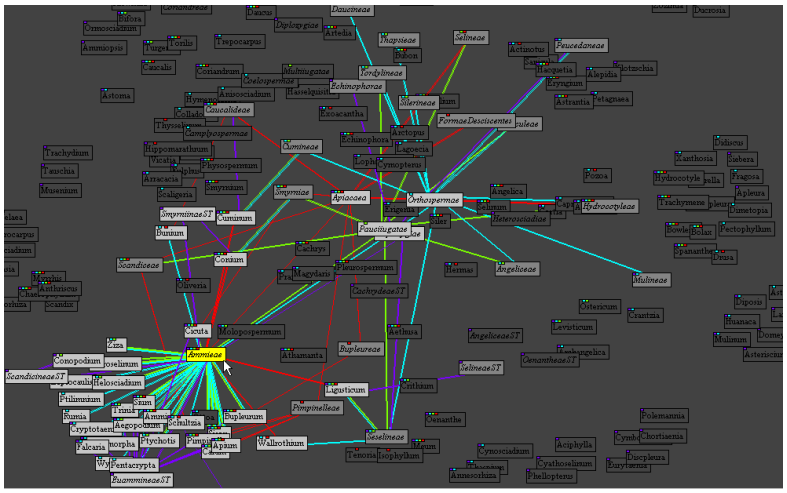


Figure 4.10: *Graph visualization of overlapping classification hierarchies [24].*

user can switch on and off the display of individual hierarchies that require the visualization to readjust itself to suit only the hierarchies that are displayed.

The main advantage of this approach is the total integration of all the hierarchies. All the hierarchies can be seen at once in a single visualization. Therefore, limitations on linking objects (see Section 4.4.3) are completely avoided, since there is only one visualization area that contains all the information. However, problems arise when the user wants to filter out certain hierarchies to get a closer look. As can be seen in the figure, the total picture looks quite crowded with information. This is because all the nodes are visible, as well as all the links between them. Graham et al. use coloring to display the current filter, i.e., links between nodes in the selected hierarchies are displayed using colors with light hue and links between nodes that are not selected are not displayed at all. Furthermore, the nodes in the selected hierarchies are assigned a light color and the nodes that are not selected are colored black. This way, the selected hierarchies draw the attention of the user, which is a desired effect, because the user has selected these hierarchies, which indicates the user's interest in these hierarchies.

To differentiate between hierarchies in the graph, the visualization displays the links between the nodes with a different color for each hierarchy. The colors are from a linear color scale (see Levkowitz and Herman [39]) in which the perceived difference between two colors is proportional to their distance on the color scale. The colors used are evenly spread along the color scale, so they appear to be evenly spread along the color range, which makes differentiating between the colors easier.

Furthermore, the nodes are displayed individually as labeled rectangles. Each node is colored using a gray-value that indicates its distance from the root nodes

(a hierarchical DAG can have multiple root nodes). The lighter the node, the closer it is to the root nodes. Note, however, that the nodes that are furthest away from the root nodes are still colored a little lighter than the nodes that are not selected, which are colored black. In addition to the colored links, each node has a small set of colored glyphs along the top of the labeled rectangle that indicates the hierarchies the node is a member of. These colored glyphs are even displayed when the node is not selected, which enables the user to notice membership of particular classifications even when the links are not displayed.

Graham et al. state that the reasoning behind the use of gray and color indicators is that these two different color scales are perceived differently. Gray scales are perceived as being ordinal scales, indicating a quantitative measure, while color scales are generally seen as only indicating membership or some other qualitative measure. Therefore, Graham et al. use color to indicate membership of a particular hierarchy structure, which is a qualitative property, and use the gray scale to show distance from the root, which is a quantitative metric.

To conclude, the real drawback in using this visualization method is that it is difficult to see the hierarchical structure of the graph, compared to tree-map visualizations where the hierarchical structure is by far more comprehensible (see Figure 4.3). Of course, the user can select and deselect hierarchies, but differentiating between different paths from the root node down to the node of interest remains difficult, even with the specific use of coloring. This remains an intrinsic problem when using a fairly standard force-directed graph drawing algorithm. However, caution needs to be taken when comparing this graph-based visualization to tree-map visualizations, because a tree-map visualization can only visualize a hierarchical tree structure which is significantly different from visualizing a graph structure.

Graham et al. ruled out this option of hierarchical graph visualization, because of two reasons. Their prototype had a limit of displaying and updating roughly 250 node positions at a rate of 4/5 refreshes per second. This was due to the fact that the graphical update on-screen took a considerable amount of time, which they could not reduce. Secondly, force-directed placement algorithms have an intrinsic high complexity. They have investigated a method to reduce this complexity to speed up the prototype that uses some degree of stochastic sampling to move the nodes rapidly to their final positions. This method introduced an unacceptable amount of visual jittering whenever changes were made to the spring-mass model. For these reasons, they ruled out this visualization prototype.

4.5.2 Set-based hierarchy visualization

The second approach of Graham et al. [24] is a set-based visualization. Since a categorization of objects, such as a botanical taxonomy, consists of sets and

subsets, it seemed logical to exploit this feature in the visualization.

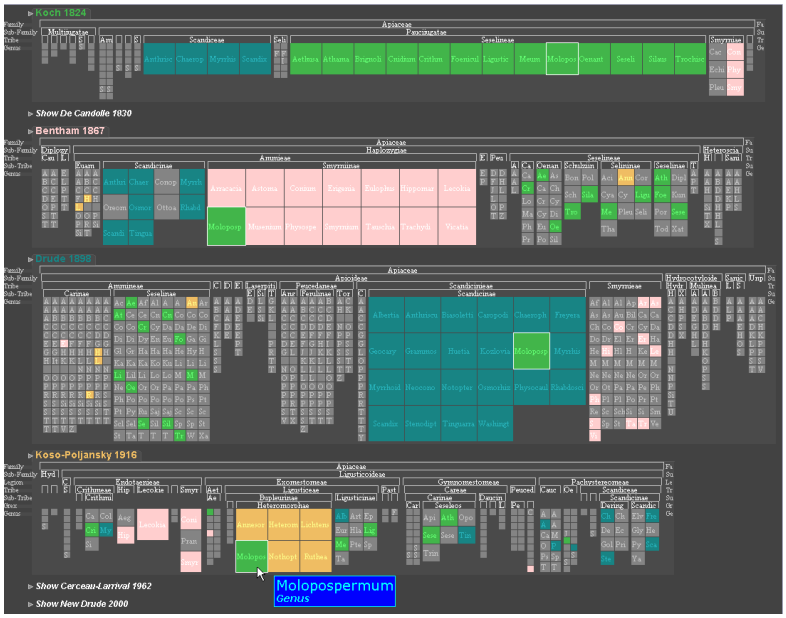


Figure 4.11: *Set visualization of overlapping classification hierarchies [24].*

As explained in the introductory text to this chapter, a single classification is a categorization made by one taxonomist. As a result, this single classification is a tree structure. When multiple taxonomists create a taxonomy on the same subject, multiple tree structures are formed that share a large number of objects. Since the set-based solution requires tree structures, the complete taxonomy is split into the single classifications made by the single taxonomists. These classifications are subsequently displayed separately using a tree visualization method, see Figure 4.11.

Each horizontal system of squares and rectangles represents a single classification by a single taxonomist. The leaf nodes are represented by small rectangles, while their parent nodes are represented by rectangles that are placed above the leaf nodes. To reduce screen space problems, the leaf nodes are arranged in a grid formation, as opposed to the normal style of a linear layout for each level of a hierarchy. Furthermore, each node contains the label of the category it represents, but again to reduce screen space problems, only the first character of the label of the leaf nodes is displayed.

Interaction with the data is performed using linking and highlighting. Since certain nodes are present in multiple hierarchies, correct and comprehensible linking of nodes between hierarchies is important. This is solved by letting the user select a node, which is subsequently highlighted using the same color in all hier-

archies. Each node, including the large parent nodes, can be shared in multiple hierarchies. Each selected node has a different color, but equal nodes in different hierarchies have the same color. For no apparent reason, the used colors are not as bright as in the graph-based visualization.

In addition to just selecting a node, the node's rectangle is expanded to hold the full title of the object. This occurs in each hierarchy the node is present. A drawback is that upon selecting a node, the overall picture changes very much, because all the hierarchies have to be readjusted to fit the enlarged node. After selecting a node, it takes some time for the user to get acquainted again with the new image on the screen.

To conclude, the set-based system is very much capable of displaying the different hierarchical parts of the total structure of the information set. However, the hierarchical structure per classification may get clear to the user, the overall structure is not comprehensible at all. This is because a single object that is incidentally shared is displayed multiple times, in each hierarchy, while, in fact, it is only one single object. The price to be paid for this (partially) clear hierarchical structure is the difficult linking of nodes between hierarchies. The main problem is that only by highlighting nodes, the structure can become visible. Moreover, the readjusting of the nodes upon selection does not permit fast linking through highlighting, because every time the picture changes, the user needs some time to get acquainted with the new image.

Graham et al. conducted a user experiment with taxonomists to evaluate the visualization prototype. Since the hierarchical structure of the data set was more important to the taxonomists than the graph structure, the taxonomists positively responded to the set-based prototype as opposed to the graph-based prototype described in the previous section. The taxonomists stated that the multiple tree effect and the grouping of sets was closer to how they viewed classifications when working with taxonomic data, rather than a node-link diagram as seen in the graph-based prototype. However, there were still problems with selecting nodes and mentally linking the same nodes divided over multiple hierarchies, as well as the problems involving the changing of the layout each time a node was selected. But Graham et al. thought they could overcome these problems and continued developing this visualization prototype.

4.5.3 Venn-diagram visualization

As technology advances and file systems get bigger and bigger, traditional hierarchical file systems have the limitation that files can only be structured according to one attribute. In general, a file system is a hierarchical classification of files in the form of a tree structure. In this structure, the files are the leaf nodes that are placed in a directory (a parent node) that represents a common attribute for all

the files in that directory. Since disk storage continues to get cheaper while the amount of storage space increases, it is not uncommon to have a large amount of files on a storage disk that could be structured in a number of different ways. For example, the same files could be structured according to different criteria, such as ”place”, ”date”, ”subject” etc. Such a classification of files would create a graph structure for a file system, which is currently not supported by the mainstream file systems. De Chiara et al. [14] provide a visualization method that is capable of displaying such a graph structure by using a technique similar to Venn diagrams.

Strictly speaking, a Venn diagram is a special case of an Euler diagram. An Euler diagram uses shapes to denote sets. When shapes overlap, the overlapping part denotes the intersection of two or more sets. Only non-empty sets are represented by a shape in an Euler diagram. A Venn diagram is a special Euler diagram in which all possible intersections of sets are non-empty, and, therefore, all possible intersection of sets are denoted by a region. Since a number of intersections between directories in a file system are empty, the visualization method uses Euler diagrams. But, since De Chiara et al. have named their visualization method VennFS (after Venn File System), and since Venn diagrams are more widely known than Euler diagram, we refer to this sort of diagrams as Venn diagrams.

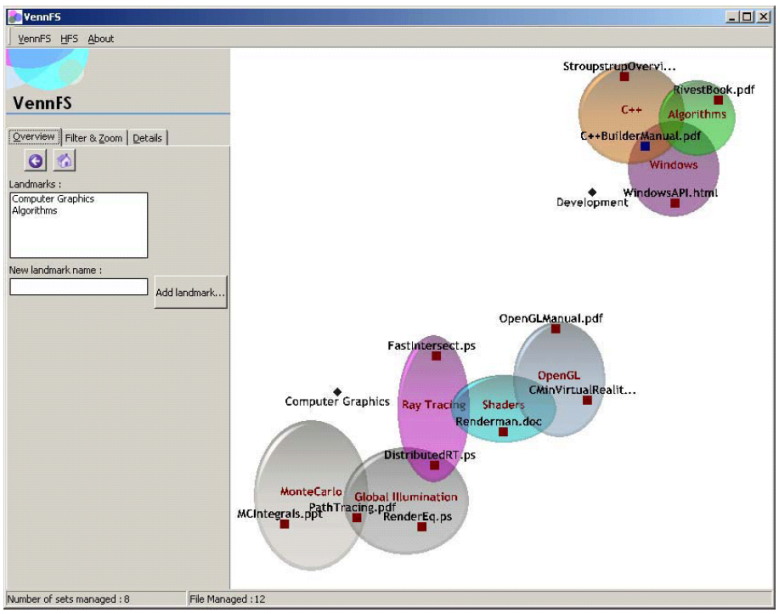


Figure 4.12: *Venn-diagram visualization of file categorization.*

As shown in Figure 4.12, files are classified into categories, according to different criteria. Each category is represented by a colored ellipse, while the files are represented as small labeled squares. Since a file belongs to at least one category,

the ellipse encloses the files in that category. In the case a file belongs to multiple categories, the colored ellipses representing these categories overlap analogous to the set representation style known as Venn diagrams.

Interaction with the data is performed using filtering. The user is able to select a filter on the shown files according to file size, or modification date. Furthermore, the user can select a file which results in the display of more detailed information on the selected file.

A drawback of this approach is that only two levels in the hierarchy can be shown, which results from the intrinsic non-hierarchical properties of Venn diagrams. The notion of parent nodes and child nodes is sparsely supported, i.e., only one layer of parent nodes and one layer of child nodes is present in the visualization, while in most hierarchical information structures, a number of levels are present that have to be visualized.

Furthermore, since all categories and files are labeled, the screen is cluttered with labels. This is very distracting when the user wants to get a quick overview of the structure of his storage space. On the other hand, it is perfectly clear without user interaction which glyph represents which file. Moreover, the coloring of the nodes is quite arbitrary. The user has to select a color for each category in the file system and is, of course, very well capable of selecting an unintuitive coloring.

To conclude, the approach of using a Venn diagram style visualization certainly has potential, because opposite to a graph visualization, there is no screen clutter due to links between nodes. Furthermore, since there is only one visualization area, limitations of linking are completely avoided (see Section 4.4.3). On behalf of navigating and filtering, fairly standard methods are used. However, this prototype is only capable of displaying two levels in the hierarchy at once, because Venn diagrams are intrinsically non-hierarchical. Moreover, standard Venn diagrams represent for a number of sets all possible combinations of unions and intersections. This way all possible subsets are created that overlap each other. Since these complete Venn diagrams do not look appealing when constructed from more than 4 sets, De Chiara et al. have implemented a visualization technique that resembles Venn diagrams.

4.6 Conclusion

So far we have seen a number of ways to visualize hierarchical graph structures. Each visualization technique has certain positive points as well as certain drawbacks. So far, there is no out-of-the-box solution that is capable of visualizing the hierarchical DAG structures from a classification in a Cristal.

However, we have seen a number of interesting properties we would like to see combined in a visualization of a classification from a Cristal, i.e.,

- *Hierarchical structures*
  Tree-map layouts offer intrinsic support for hierarchical structures using inclusion. An interesting property is the fact that transitive relations between objects are natively supported,
- *Flexible force-directed layout algorithm*
  Node-link diagrams often use force-directed layout algorithms that are very flexible and allow for many adjustments,
- *Overlapping structures*
  Venn-diagram style visualizations that offer support for overlapping structures such as graph structures,
- *Advantages using human visual perception*
  Visualizations that take advantage of the human visual perception system by using motion and coloring as well as other perceptual cues such as shape or size. These perceptual cues should not only be used in the visualization technique, but also in the interaction technique.

Of course there are also a number of properties we would like to avoid, because they can be regarded as drawbacks on the visualization of the hierarchical DAG structures of a Cristal, i.e.,

- *Node-link diagrams show no hierarchical structure*
  Node-link diagrams do not provide a clear overview of the hierarchical structure of a graph,
- *No duplication of nodes*
  Visualizations such as Graham et al.'s set-based visualization (see Section 4.5.2) duplicate nodes which introduces linking problems that we want to avoid,
- *Multi-dimensional techniques show no hierarchical structure*
  Multi-dimensional information visualization techniques such as Parallel Coordinates (see Section 4.3.3) that do not provide an overview of the hierarchical structure.

Since the hierarchical structure of the classification structure in a Cristal is the most important to visualize, as well as the graph-structure, we propose a novel visualization technique in the next chapter that is a combination of the tree-map technique and the Venn-diagram technique. In combination with a flexible force-directed layout algorithm we are able to show the hierarchical structure and the graph structure without the need to duplicate nodes.

Chapter 5

Visualization of Cristal

As stated at the end of the previous chapter, we will propose a novel visualization method in this chapter. First we state our approach, subsequently we explain the way we are going to visualize the different aspects of the classification structure in a Cristal. Finally we explain the interaction techniques used in the prototype.

5.1 Approach

As stated at the end of Chapter 4, we want to create a visualization method that can show both the hierarchical structure and the graph structure of a hierarchical DAG. As we have seen, probably the best way to visualize a hierarchical structure is the use of tree-maps (see Figures 4.2 and 4.3), since the inclusive shapes immediately induce the percept of part-whole relations. A welcome extra aspect of this kind of visualization is the intrinsic visualization of the transitive part-whole relations present in the structure. However, the tree-map technique can only visualize tree structures and the category structure we want to visualize in the first place is a hierarchical DAG.

An essential difference between a DAG and a tree structure is the fact that the former allows nodes to have multiple supernodes, while the latter only allows nodes to have at most one supernode. One of the main ideas behind a tree-map visualization is that each node encloses all its subnodes while maximizing screen space usage, which is of course only possible when each node has at most one supernode. In our case, we have a DAG structure where nodes can have multiple supernodes. Using the tree-map paradigm, this would result in overlapping nodes in the case these nodes share the same subnodes. Moreover, this kind of overlapping entities is exactly the same as the way a Venn-diagram is constructed. In a Venn-diagram, two sets are depicted by two overlapping circles while the overlapping part of the two circles represents the intersection of the two sets. When

we also keep in mind that the relations between categories in a classification in a Cristal are part-whole relations, a category is essentially a collection of its subcategories, which complies with the Venn-diagram paradigm of sets and their intersections. We, therefore, propose to combine a Venn-diagram style visualization method with the hierarchical inclusion paradigm found in tree-map visualizations.

Our initial approach is to represent the categories of the classification structures in a Cristal by shapes, such that categories actually reside in their supercategories. In the case there are categories with multiple supercategories, these categories should reside in all of their supercategories, which results in overlapping shapes.

Furthermore, we have to find a placement of the categories, such that the supercategories can be drawn as enclosing shapes around them. Since overlapping shapes have a meaning, it is important that shapes that do not have overlap, are drawn non-overlapping. Only categories that have overlap should show overlapping regions on the screen. For this, the categories should be placed on the screen in such a way that this is achieved. This is not trivial as opposed to a naive tree-map visualization. We will use a force-directed layout algorithm to determine positions for the categories that allow the supercategories to be drawn as enclosing shapes around them. We choose a force-directed algorithm because such algorithms are relatively easy to adjust and, therefore, very flexible and suited for multiple purposes, such as the proposed layout of categories. This algorithm is explained in the next chapter, Chapter 6. Using this approach, we have to give up the tree-map's capabilities of maximizing the screen space usage, but this is inevitable and not considered really problematic.

We continue this chapter by detailing the way we visualize the categories, the levels and the time properties of a classification structure of a Cristal.

5.2 Visualization of categories

To visualize the graph of categories, we represent each category with a shape. As stated, these shapes have to include each other to depict a category-subcategory relation, but they also have to overlap each other to depict a category with multiple supercategories. We have investigated the use of a number of different shapes, which we explain next. We started out with simple rectangles as can be seen in Figures 5.1 and 5.2.

Initially, we chose rectangles, because they are very easy to implement and capable of showing the desired inclusion and overlap. Figure 5.1 shows a small tree structure where the categories completely enclose their subcategories. The leaf categories of the graph, which we call atoms, i.e., categories that have no subcategories, are special categories which are denoted by a small square. The

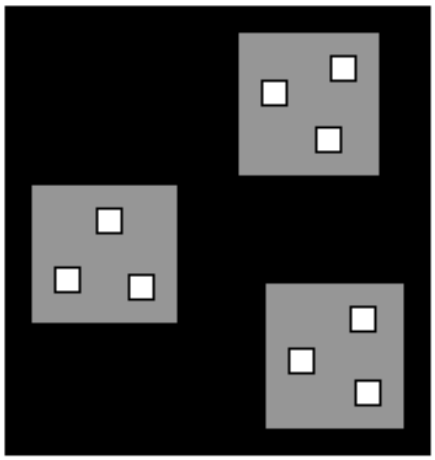


Figure 5.1: *Tree structure with three atoms inside a category, and three categories inside a rootcategory.*

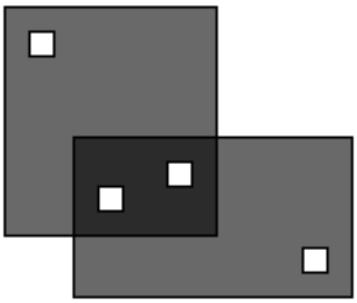


Figure 5.2: *Two overlapping categories with two atoms in the overlapping region.*

inclusing rectangles in the figure clearly induce the visual percept of subcategories actually being in their respective supercategories.

Subsequently, Figure 5.2 shows a small graph structure in which two categories have the same two subcategories in common. In essence, this means that the supercategories overlap each other, therefore, we draw the rectangles such that they are overlapping with the two subcategories (atoms in this case) in the overlapping part. The overlapping rectangles in the figure clearly induce the visual percept of subcategories actually being in multiple supercategories.

There is, however, a problem with using the proposed technique with rectangles to display the hierarchical DAG structures. Because of the fact that the rectangles are not allowed to rotate, it is not always possible to display a graph using rectangles. Figure 5.3(a) shows a graph structure with at the bottom row four atom categories. The middle row show the categories that represent all combinations of two atoms, while the top row displays the root category. This particular structure can not be comprehensively shown using rectangles, as can be seen in Figure 5.3(b). This figure shows a lot of overlapping shapes, but it is not clear exactly which rectangles are overlapping which.

Therefore, we investigated the use of blob shapes. Figure 5.3(c) shows the same graph structure visualized with blobs. Obviously, the structure is more clear while using blobs, however, rectangles have a very interesting property that blobs lack. In the case only a part of a rectangle can be seen, the human visual perception system can quite easily predict the rest of the rectangle. This is because the brain

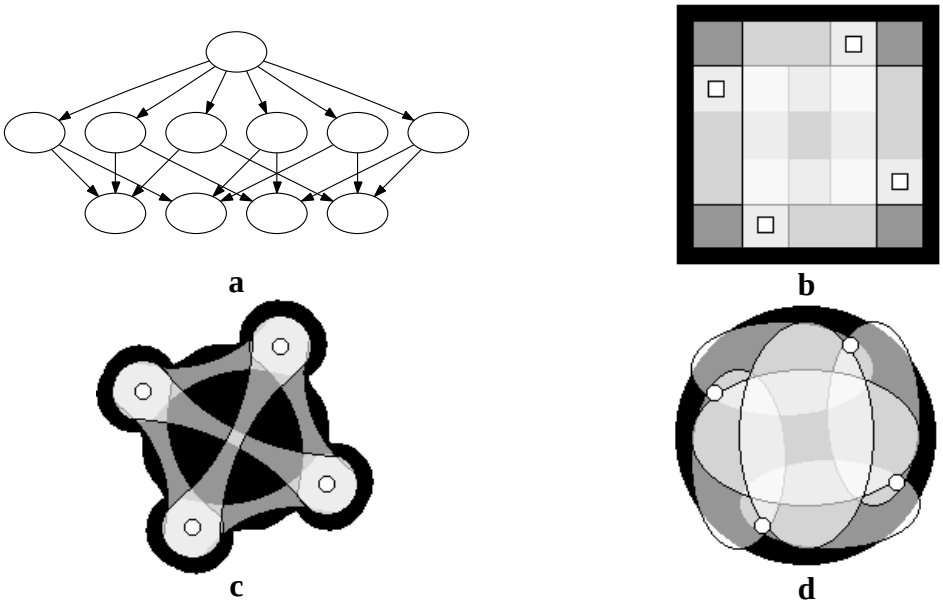


Figure 5.3: *Category shapes:* ***(a)*** *Graph with all combinations of two atoms as categories,* ***(b)*** *Visualization of* ***a*** *using rectangles,* ***(c)*** *Visualization of* ***a*** *using blob shapes,* ***(d)*** *Visualization of* ***a*** *using ellipses.*

only has to see two edges of a rectangle to mentally visualize where these edges cross. Blob shapes lack this fundamental property, because since a blob can take on any shape, the visual perception system can not make a mental image of an unseen part of a blob. This property does not manifest itself when viewing small structures such as Figures 5.3(b) and (c), but we found out that larger structures are a lot clearer when visualized with rectangles than with blobs.

The third shape we have investigated is the ellipse. Ellipses have the same property as rectangles, where the human visual perception system can make a mental image of the complete ellipse quite easily when a part of the ellipse is not shown. Figure 5.3(d) shows the graph structure visualized using ellipses. As can be seen, the structure is more clear than with rectangles, but not as clear as with blobs. Furthermore, the restriction that a supercategory completely encloses its subcategories is quite difficult to guarantee. The inclusion of the ellipses is even in the figure not perfect. Moreover, when the stated restriction can be guaranteed, it is fairly possible that a supercategory will be denoted by a very large ellipse, because of an awkward positioning of its subcategories. This happens more frequently when the path from rootcategory to atom is relatively long and results in very large ellipses with large empty regions. Therefore, we chose to stick with the relatively easy rectangular shapes. Rectangles are easy to implement as well as easy to mentally comprehend in most cases which results in a clear total picture.

Furthermore, rectangles are compared to ellipses very space efficient. The limitations in the number of graph structures we can visualize using rectangles do not outweigh the advantages rectangles have compared to blobs and ellipses, since in practice, such graph structures are very rare.

Another important aspect of the visualization is the notion of depth in the graph. All categories have a certain distance from the rootcategory. When this distance is somehow visible, this would enhance the perception of the hierarchical structure. Of course, the depth of each category is inherently visualized using the proposed method of inclusion and overlap, but to enhance this, we want to use coloring. Section 4.2.2 states the importance of color, moreover Section 4.5.1 states that a gray scale is perceived as an ordinal scale, indicating a quantitative measure, such as the distance of a category to the root category. A color scale is perceived as a qualitative measure, indicating membership of a set, such as a category's membership of a level.

In analogy to the previous, we use gray values to denote the distance of a category to the root category. The atoms are colored white and each supercategory is colored a little darker up to the rootcategories that are the darkest. A few simple tests showed that the other way around, i.e., black atoms and light root categories, does not indicate the depth of a category in the graph as well as the proposed coloring. This is because we use transparency to denote overlapping rectangles resulting in the overlapping region being darker which, in turn, results in difficult distinction between an overlapping region and a subcategory, because both are darker than the surrounding supercategories. Since the proposed coloring uses only gray values, the hue can be used to display other aspects of the classification structures, such as levels.

5.2.1 Ghost categories

As stated, the general approach is that a subcategory, which is semantically part of a supercategory is represented by a rectangle which is actually 'in' the rectangle of the supercategory. This means that supercategories which have one or more subcategories in common must have an overlapping region.

However, it is possible that the set of subcategories of a category is a part of a set of subcategories of another category, see Figure 5.4(a). Category 1 and category 2 have two subcategories in common, namely, category 4 and category 5. This should show up as an overlapping region in the visualization (see Figure 5.4(b)), but this figure suggests a tree-like structure as can be seen in Figure 5.4(c), where category 2 is suddenly a subcategory of category 1. In the original structure, this relation was not present, so category 2 should not be entirely included in category 1. Following from this, there is a part of category 2 that is different from category 1, but this part happens to be empty, i.e., not contain-

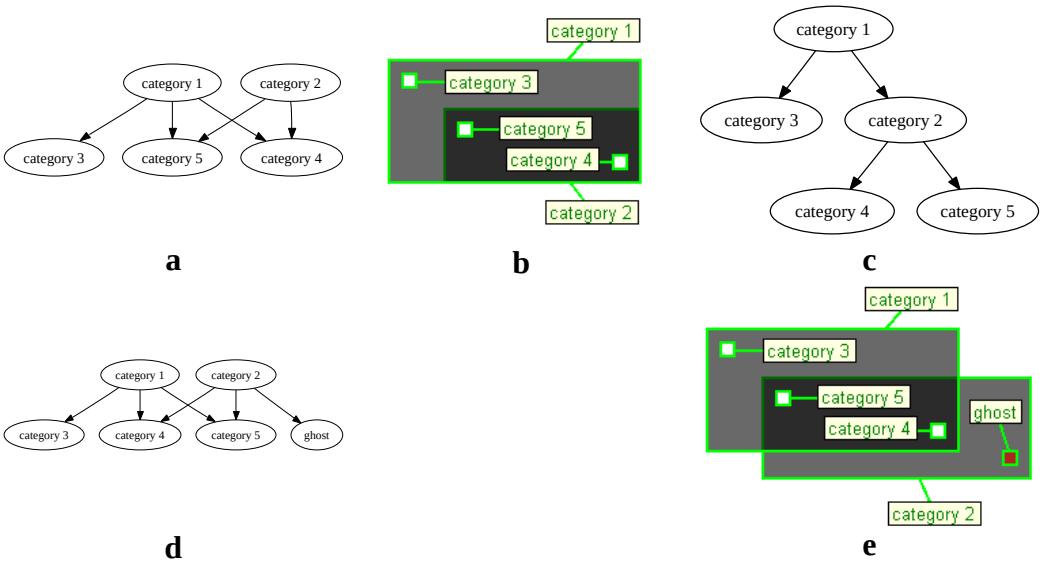


Figure 5.4: *Ghost categories:* ***(a)*** *Two categories with two subcategories in common,* ***(b)*** *Visualization of* ***a*** *which is actually* ***c****,* ***(c)*** *The graph structure has changed into a tree structure,* ***(d)*** *Ghost category added to category 2,* ***(e)*** *Visualization of* ***d*** *with the ghost category shown in red.*

ing any subcategories. Therefore, there should be a non-overlapping region that shows this part, but this is not possible, because a supercategory is always drawn as an enclosing rectangle around its subcategories.

A solution to this problem is to add so called ghost categories to the graph which are used to fill up these empty non-overlapping regions such that these regions become visible. In the example, a ghost category should be added to category 2. Addition of this ghost category fills up the empty part of category 2 and creates the graph of Figure 5.4(d) which is visualized as shown in Figure 5.4(e). Note that the ghost categories are actually not visible in the visualization.

Formally speaking, we have a directed graph of categories $G = (V, E)$, where V is the collection of categories in the graph, i.e., the collection of vertices, and E is the collection of direct relations between these categories, i.e., the collection of edges between the vertices. Now let $sub(v)$ be the collection of direct subcategories of category $v \in V$, i.e.,

$$sub(v) = \{u \in V | (v, u) \in E\} \tag{5.1}$$

Then, in analogy with the previous example, a ghost category should be added to c_2 when the set of subcategories of c_2 is completely contained in the set of subcategories of c_1. A category is completely contained in another category when all

subcategories of the former category are also subcategories of the latter category, i.e.,

$$sub(c_2) \subseteq sub(c_1) \Rightarrow sub(c_2)' \leftarrow sub(c_2) \cup ghost \tag{5.2}$$

5.3 Visualization of levels

As explained in Chapter 2, a level is a collection of categories that have no direct relation in the category-graph and no subcategory in common. Levels represent element-set relations in contrast to categories that represent part-whole relations. Furthermore, levels are partially ordered and form a hierarchical DAG, such as the categories. Moreover, it is possible that categories reside in multiple levels, thus levels can overlap each other, with the shared categories in the overlapping region. However, since the categories in a level do not have a direct relation in the category-graph and no subcategory in common, the rectangles representing the categories in a level can not be overlapping.

To visualize the graph of levels, it is fairly possible to use the same visualization method as has been used for the categories, since the structure of the level-graph is similar to the structure of the category-graph. However, this would render two images that look alike, one for the levels and one for the categories, but represent two things that are semantically completely different. On top of that, the levels and categories are related to each other, since levels are sets of categories. The relation between the levels and the categories would not be clear from the two resembling images. Furthermore, it is not possible to overlay the visualization of categories with the visualization of levels, because this would create two hierarchical structures on top of each other, which would not show the relations between a category and the level(s) it resides in.

We chose, therefore, to visualize a level by selecting it from a list and subsequently highlighting all categories that make up this level. Since the categories in a level do not have a parent-child relation, the categories in a level should not be overlapping in the highlighted set of categories. This property can, therefore, be easily verified by the user. Furthermore, since we are already using gray values to denote the depth of categories in the graph, we use hue to highlight the categories in a level, because the color scale is better suited for a qualitative measure such as the indication of membership of a set.

Figure 5.5 shows the selection of two levels using the running example of Chapter 2 regarding a classification structure of The Netherlands. The left part of the figure shows the category structure of the classification while level ”East-West Division” is selected. The categories ”East Netherlands” and ”West Netherlands” belong to this level and are colored orange. The right part of the figure shows

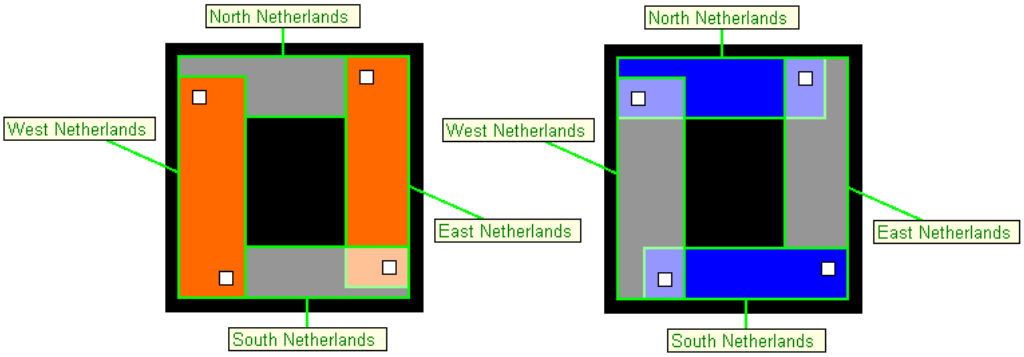


Figure 5.5: *Two different levels.*

the visualization while level ”North-South Division” is selected. Because the categories ”North-Netherlands” and ”South-Netherlands” make up this level, these categories are colored blue.

The two parts of the figure also show that the atoms in the classification structure, i.e., the small white squares in the corners, reside in multiple levels. When both levels would be selected at the same time, the red and blue categories would overlap because of this, therefore, we only allow the user to select one level at a time. Furthermore, the categories in a level do not overlap, as can be seen in the figure, because they have no direct relation and no common subcategories, which is a restriction of a level.

5.4 Visualization of time

As stated in Chapter 2, each level and category, and even stronger, each object in a classification variable of a Cristal, has a start date and an end date. The object is valid, i.e., existing, in the time between those two dates and is not valid, i.e., not existing, outside the time frame denoted by those two dates. It seems, therefore, obvious to let the user select a time frame by selecting a start date and an end date, and only show the objects that are valid in this period of time. According to this definition, there are five types of categories:

1. Categories that are valid during the entire selected time frame.
2. Categories that become valid somewhere in the selected time frame.
3. Categories that stop being valid somewhere in the selected time frame.

4. Categories that become valid and stop being valid somewhere in the selected time frame.

5. Categories that are not valid during the entire selected time frame.

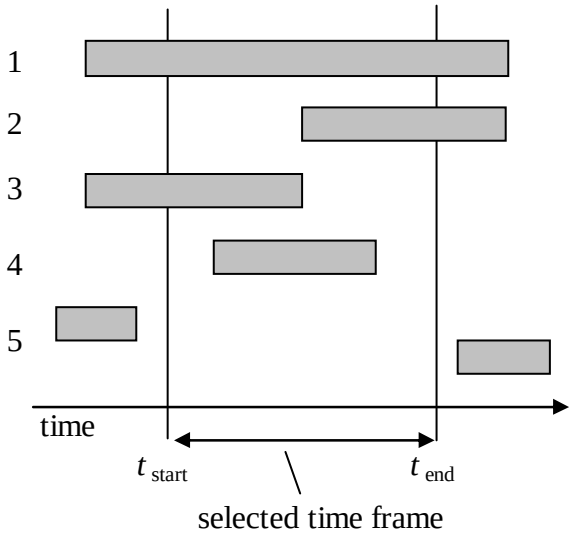


Figure 5.6: *Different types of validity.*

Figure 5.6 schematically shows the five different validity types for the categories. Note that type 5 consists of categories that are either valid before the time frame or after the time frame, but not both at the same time.

A possible way to show these different types of categories is to color them using different colors according to the type of validity following from the selected time frame. Using a slider, the user can change the time frame by sliding the starting date (t_{start}) or the end date (t_{end}) of the time frame. However, since we are already coloring categories to show the levels, additional coloring of categories to denote their type of validity would probably result in a visual chaos. We, therefore, try to find another way to visualize time.

In addition to defining starting and end dates for objects, the Cristal model offers support for tracking changes in objects using a combination of the *key*-attribute and the starting date and end date, as explained in Section 2.5. This results in categories having predecessors and successors. To be able to show a combination of the validity type of a category as well as its predecessors and successors, we propose to show only the categories that are valid somewhere in the selected time frame (types 1, 2, 3 and 4 of Figure 5.6). Categories that are not valid in the selected time frame are hidden (type 5 of Figure 5.6). Furthermore,

we show the changes in time of a category by drawing arrows from predecessor to successor, as shown in Figure 5.7.

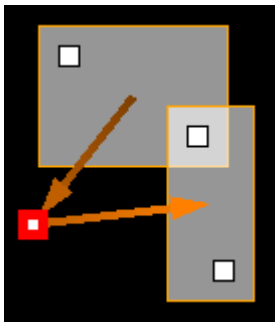


Figure 5.7: *Arrows denote changes from predecessor to successor.*

The figure shows a category with two subcategories at some point in time changing into a category with no subcategories. After this, the category changes again into a category with two subcategories again. Using this method, a sequence of successors is denoted as a sequence of arrows. Note that we gradiently color the arrows from dark to light. We use a distinctive color to make the human visual perception system mark these arrows as special and, therefore, draw attention to them.

5.5 Interaction and usability

As outlined in Chapter 4, perceptual cues and interaction are an important part of a visualization tool. Of course, we included a number of such, mainly visual, cues in our visualization. The following sections give a detailed description of the incorporated perceptual cues, as well as the different interaction techniques.

5.5.1 Perceptual cues

The most important perceptual cue used in the visualization is the inclusion in combination with the overlap of shapes. A shape drawn on top of another shape induces the percept of inclusion when the topmost shape is smaller than the underlaying shape. We use this percept to display the relation between a category and its subcategories, because this relation is indeed an inclusive relation. The subcategory is a part of its supercategory. Additionally, two overlapping shapes induce the percept of a common part. We use this percept to display categories that have multiple supercategories. Indeed, in this case, the subcategory is at the

same time part of both supercategories, thus the supercategories have something in common, which is exactly the subcategory.

Furthermore, to display the depth of a category in the graph, we color them from black (root categories) to white (leaf categories) along a gray scale. The gray scale is perceived as a linear scale and thus suited to display quantitative measures (see also Section 4.5.1). The depth of a category is typically a quantitative measure. Moreover, this leaves other colors to display different properties.

One of these properties is the validity of a category, in combination with its predecessors or followers. As stated before, an arrow is drawn from predecessor to follower to denote the change over time of the category. Perceptually, an arrow denotes a certain movement, which induces a start position and an end position. The start position is the predecessor-category that changes into the follower-category, which is the end position. Furthermore, the start position of the arrow is colored darker than the end position of the arrow which results in a gradient coloring of the arrow. This greatly enhances the percept of movement.

Coloring is also used to display levels. Levels are groups of categories that cannot have overlap. The user is enabled to select a particular level from a list of levels and subsequently all the categories belonging to that level are colored. As stated, we use non-gray colors to display the categories in a level in order to make a clear distinction between the categories in the selected level and the other categories.

Furthermore, motion is next to coloring one of the most important aspects of a visualization. The iterative layout algorithm using the spring-embedder model is inherently capable of displaying motion. The positions of the categories change after each iteration step of the layout algorithm, therefore, a redraw of the entire graph structure on the screen is all that is required to create an animation. Furthermore, motion is incorporated in combination with user interaction, as described in the next section.

5.5.2 Interacting

In combination with the perceptual cues described in the previous section, interaction with the dataset is also an important factor. Of course, interaction techniques should aid in the comprehension of the information set, and, therefore, make use of perceptual cues as well.

Labeling is an important factor of a visualization. Upon hovering a category with the mouse, a label displays the contents of the ”name”-attribute of the category. Clicking a category while holding the shift-button adds the clicked category to the multi-selection, i.e., a group of selected categories. The categories in the multi-selection show a green border around them to distinguish them from other categories. Furthermore, these categories show their label permanently.

As stated in the previous section, motion is next to coloring one of the most important aspects of a visualization. Movement of categories is achieved through interaction with the information set and through the inherent motion of the force directed layout algorithm (which is explained in Section 6). Since the layout algorithm is an iterative algorithm, the results after each iteration of the algorithm can be displayed, which results in an animation. We have chosen to show this animation on the screen to give the user an idea of the way the layout is being constructed.

Furthermore, the user can interact with the dataset by dragging categories to another position, which of course induces motion. By dragging a supercategory, for example, it is clearly visible what subcategories belong to the dragged supercategory, because the human visual system associates objects moving in the same direction with the same speed as somehow connected together.

It is possible to double-click a category to get a properties-window in which it is possible to navigate upward and downward in the graph of categories by clicking a supercategory or subcategory from the currently selected category. The properties-window always shows the currently selected category, while the graph shows the currently selected category with a red bounding box. This way, the user can cognitively link the object in the graph visualization with the object in the properties-window.

5.6 Prototype

The prototype, which is called ”CristalView”, contains the main visualization area, where the graph of categories is showed. A screenshot of the application is shown in Figure 5.8.

The categories, i.e., nodes in the graph, are represented by rectangles and squares. A supercategory is represented by a rectangle, surrounding its subcategories. The atoms, i.e., leaf nodes, are represented by small squares, because they are different from the other categories, since they have no subcategories.

Furthermore, the right of the screen displays the list of levels present in the Cristal. Each level has an associated color. Upon selecting a level, the categories belonging to that level are appropriately colored to show the contents of the level. Since categories in a level can not have overlap, this property can be easily visually verified by the user.

Additionally, since each category has a certain period of time in which it is valid, a time slider is displayed in which the user can set a time frame by selecting the start-date and end-date. Subsequently, only those categories that are valid somewhere in the selected time frame are shown. Moreover, in combination with the validity of a category, it is possible to denote the predecessors and the follow-

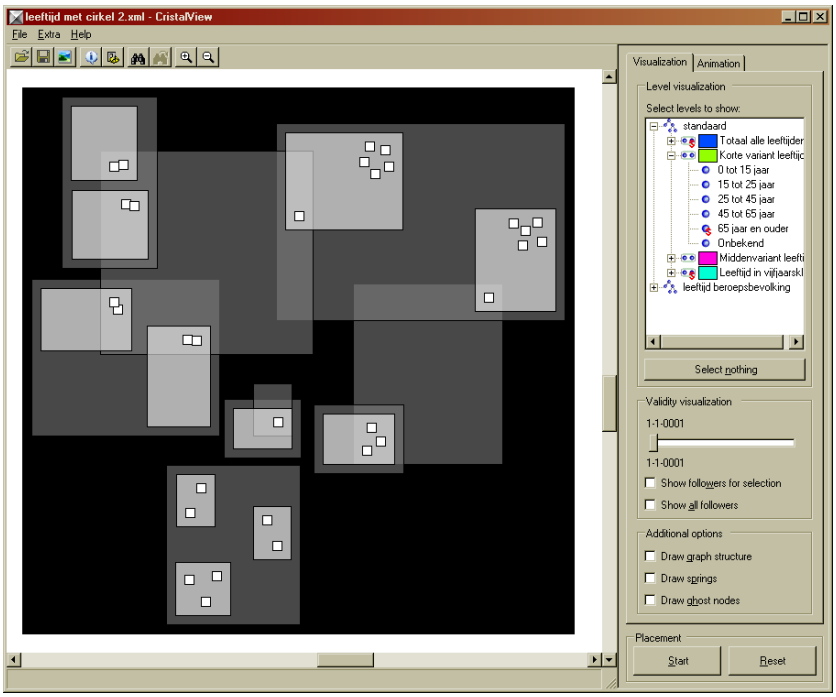


Figure 5.8: *Screenshot of the final program.*

ers of a category. An arrow is drawn from a predecessor to a follower to display the change of that particular category over time.

Also, it is possible to double-click a category, which brings up a properties-dialog. This dialog shows the attributes of the category, as described in Chapter 2. In addition, these static properties can be edited. Also, the supercategories and subcategories of the selected category are shown, and can be clicked to navigate upward and downward in the graph of categories.

5.7 Final application

A number of participants of the user experiment explained in Chapter 7, had remarks about the visualization tool. Of course, this is not very surprising, as they were testing the prototype. These participant remarks are listed and discussed in Section 7.3. The final application is basically the same as the prototype, but according to these remarks, we have added a few things to the visualization tool. Figure 5.8 shows the final application.

A search functionality was added to the tool which enables the user to search for a certain category. When the user enters a number of keywords, the tool searches for the first occurrence in the information set of a category that matches

any one of the keywords. When a category is found, it is added to the multi-selection, meaning that the category gets a green border as well as a label. Furthermore, it is possible to search again using the same keywords, which results in more categories be added to the multi-selection.

An important user remark was that the folding of categories was too easy. Only the right mouse button had to be pressed and the category the mouse was hovering would be folded. This often resulted in surprised participants, because they did not know what they had done. Therefore, we have made it a little more difficult to fold a category, i.e., we have added a pop-up menu that pops up when the right mouse button is pressed over a category. A number of options are revealed, such as showing the properties-dialog, add the hovered category to the multi-selection, deselect all, fold the hovered category. We chose for this solution, since the prototype had no way to deselect all categories at once. Each category that was in the multi-selection had to be individually removed from the multi-selection. Furthermore, we thought a pop-up menu was a nice visual control unit, as opposed to the obscure key combinations that had to be pressed in the prototype.

Furthermore, we have implemented a toolbar with easy access buttons to enhance the usability of the user interface. Typical functions that can be reached from the toolbar are loading and saving of Cristals, but also zooming options and search options are represented by a button on the toolbar.

The results of the user experiment show that the visualization of levels is more effective with the Variable Editor. To get the best of both worlds, we have changed the list from which the levels can be selected such that it resembles the hierarchy-tree in the Variable Editor. Therefore, the list shows the list of hierarchies, with accompanying hierarchy-icon. A hierarchy can be expanded to show the list of levels in that hierarchy. Each level has a level-icon and a color associated with it. Each level can subsequently be expanded to show the categories in that level. The categories are also accompanied with a category-icon. Note that these icons are taken from the Variable Editor, and are listed in Appendix B.

5.7.1 Implementation

The prototype as well as the final application were developed using C#.NET. This was a requirement, since the complete Cristal model was implemented in C#. The language C# does very much resemble Java. Like Java, C# supports a garbage collector that frees resources when the system runs low on memory. There are, however, subtle differences. Java programs, for example, compile to byte-code that require to be executed on a Java Virtual Machine. C# programs, on the other hand, compile to an intermediate language that can be interpreted by the Microsoft .NET framework. The .NET framework translates the intermediate language to instructions the processor can execute. When a .NET program is run

for the first time, the .NET framework simply translates the intermediate language code to CPU instruction, but during execution, the .NET framework keeps track of code that is executed many times, such as repetitions. The framework subsequently optimizes these code-bits, so they execute faster. The framework does not optimize all the code, because this would be unprofitable, since it costs relatively much time to optimize. For pieces of code that get executed many times, optimizing them is profitable, because the added optimization time does not outweigh the increase in execution time.

To give a small statistical overview, the final application consists of a little less than 15,000 lines of code, split over 42 classes. The complete project is made up from 44 files.

Chapter 6

Force model

As stated in the previous chapters, we want to visualize a hierarchical graph as a collection of shapes, e.g., rectangles, that can have overlap, see Figure 5.2. In order to visualize the hierarchical structure of the graph, we draw a supercategory as an enclosing rectangle around it subcategories, which induces the percept of subcategories being actually in their supercategories. Furthermore, we use a force directed placement algorithm to find positions for the categories such that the structure of the graph is clear to the viewer. In this chapter we discuss this algorithm.

6.1 Overview

The graph of categories is a Directed Acyclic Graph (DAG) which can be represented by a graph $G = (V, E)$ where V is the collection of vertices and E is the collection of edges connecting the vertices. The vertices represent the categories and the edges represent the parent-child relations between the categories and are, therefore, directed from parent to child. Note that we do not take into account the edges that represent the transitive or reflexive relations between categories.

Force directed algorithms model this graph as a physical system and then try to find the positions $\mathbf{p}_v$ of all vertices v such that the total energy in the system is minimal. Standard force directed algorithms (Battista et al. [5]) model the edges of the graph as springs, as also shown in Figure 4.5. Pseudo code of such a standard algorithm is given in Figure 6.1.

In general, this model is used to draw undirected, possibly cyclic graphs, because all vertices in the graph are treated the same. This means that the hierarchical structure is lost. As stated, we want to draw supercategories as enclosing rectangles around their subcategories. We want the categories that have a close

```
1: const A, B {Strength of attraction, strength of repulsion}

2: procedure LAYOUT(Graph G)
3:     { Place the nodes of G at random positions }
4:     for all v ∈ V(G) do
5:         p_v ← p_random
6:     end for
7:     repeat
8:         { Calculate force vector F_v for each node v }
9:         for all v ∈ V(G) do
10:            F_s ← 0, F_r ← 0
11:            for all u ≠ v do
12:                p_uv ← |p_u − p_v|
13:                p_uv→ ← (p_u − p_v)/p_uv
14:                { Add repulsive force vector of u on v }
15:                F_r ← F_r + B/(p_uv)^2 · p_uv→
16:                if u ∈ neighbors(v) then
17:                    { Add attractive force vector of u on v }
18:                    F_s ← F_s + A · (p_uv − x_0) · p_uv→
19:                end if
20:            end for
21:            F_v ← F_s − F_r
22:        end for
23:        { Move each node v according to its force vector F_v }
24:        for all v ∈ V(G) do
25:            p'_v ← p_v + F_v · Δt
26:        end for
27:    until (End condition reached)
28: end procedure
```

Figure 6.1: *Standard force-directed algorithm.*

relation to be positioned close to each other as well. To accomplish this, we do not define springs on the edges of the graph, but between sibling-categories, as explained in the following section. Subsequently, Section 6 explains the force model we use followed by multiple sections that explain the additional parameters of our model.

6.2 Springs between siblings

We define categories that have a supercategory in common to have a close relation, moreover, two categories that have a direct supercategory in common are called *siblings*. We want to emphasize that siblings are closely related and form clusters, hence, we use springs between sibling-categories. A formal definition of sibling-categories is given below. Figure 6.2 shows a graph in which the categories are represented by steel rings. Additionally, springs are defined between sibling-categories. The arrows in the figure denote the edges in the graph, i.e., the super-subcategory relations between the categories.

Note that there are no springs between a supercategory and its subcategory. Since a supercategory is drawn as an enclosing rectangle around its subcategories, the force exerted on a supercategory is, in effect, a force on each of its subcategories. Therefore, the forces on a supercategory are passed on to its subcategories. This ensures that supercategories, which are clusters of subcategories, are connected with springs as a group and receive their force vectors as a group.

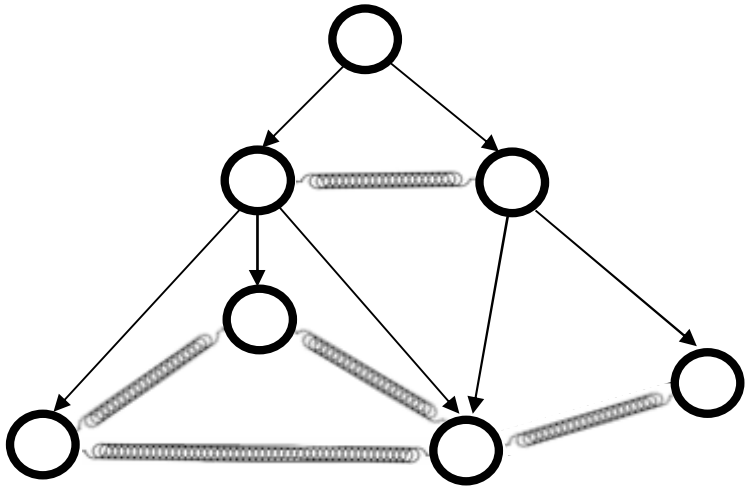


Figure 6.2: *Graph with springs between siblings.*

Since the graph is directed and acyclic, every category has zero or more su-

percategories as well as zero or more subcategories. Let $super(v)$ and $sub(v)$ be the collections of supercategories, respectively subcategories, of category $v \in V$, i.e.,

$$super(v) = \{u \in V | (u, v) \in E\} \tag{6.1}$$

$$sub(v) = \{u \in V | (v, u) \in E\} \tag{6.2}$$

Using these definitions, the two types of categories, supercategories and atoms can be defined as follows. An atom is a category v with $sub(v) = \text{ø}$ and a supercategory is a category u with $sub(u) \neq \text{ø}$.

The siblings of category $v \in V$ are defined by the categories that share a supercategory with category v, i.e.,

$$sibling(v) = \{s \in V | super(v) \cap super(s) \neq \text{ø}\} \setminus \{v\} \tag{6.3}$$

As stated, we define springs between the siblings in graph G. The force on a category depends on the attractive forces exerted by the springs between the category and its siblings. Details on the spring forces are denoted in the following section.

6.3 Force function

Each spring in the model has, in analogy with a physical spring, a spring function. Function $f_{uv}(x)$ describes the attractive spring-force for a spring between categories u and v and follows typically physical analogies, such as Hooke's spring law:

$$f_{uv}(x) = A_{uv} \cdot (x - x_0^{uv}) \tag{6.4}$$

where x_0^{uv} states the zero energy length of the spring between categories u and v, see Section 6.7 and A_{uv} the stiffness of the spring between categories u and v, see Section 6.6.

The force is a function of the distance between the categories. We can write the total force exerted on category v as the sum of the forces exerted by the springs connected to v and the total force on the supercategory of v, i.e.,

$$F(v) = \sum_{u \in sibling(v)} f_{uv}(p_{uv}) \overrightarrow{\mathbf{p}_{uv}} + \sum_{u \in super(v)} F(u) \tag{6.5}$$

with distance $p_{uv} = |\mathbf{p}_u - \mathbf{p}_v|$ and force direction $\overrightarrow{\mathbf{p}_{uv}} = (\mathbf{p}_u - \mathbf{p}_v)/p_{uv}$, where $\mathbf{p}_v$ is defined as the position of category v. Since an atom is only a small square, the position of an atom is trivially defined as the actual coordinate of the square,

where the position of a supercategory is defined by the rectangle that encloses its subcategories. We use the center point of the rectangle as the connection point of the springs that are connected to this supercategory.

The potential energy in the springs connected to category v can be expressed as:

$$P(v) = \frac{1}{2} \sum_{u \in sibling(v)} A_{uv} \cdot (p_{uv} - x_0^{uv})^2 + \sum_{u \in super(v)} P(u) \tag{6.6}$$

The total potential energy of the system can be expressed as:

$$P = \sum_{v \in V} P(v) \tag{6.7}$$

In the optimal configuration, the total potential energy is minimal. Due to the non-linearity of the problem, it is not feasible to find a minimum directly, hence an iterative approach has to be used. A commonly used method is to start from a random configuration. The categories are subsequently moved in the direction of the forces exerted on them, such that in the end the total force on each category is zero, or, in other words, a minimum energy state is reached. This will often be only a local minimum, but the result is visually pleasant enough.

6.4 Repulsion forces

Standard force directed algorithms also use repulsion forces between categories. The result of the choice to define springs between siblings is that the springs between siblings form a fully connected graph. Each category in this subgraph of the overall spring system is connected with a spring to all the other categories in this subgraph. Therefore, the structure resembles a very strong crystal-like structure which is not easily affected by a repulsion force on one of its categories. In practice, the difference between the algorithm with repulsion forces is negligible opposed to the algorithm without repulsion forces. Moreover, in the case the repulsion forces would make a difference, this would have a negative effect on the symmetry. Categories that have an equal number of subcategories look exactly the same, but this is not always the case when repulsion forces are used. Furthermore, since the calculation of the repulsion forces is of order $O(N^2)$ per iteration in the number of categories, the performance gained by omitting them would be significant. Therefore, we omit the repulsion forces. More information on the complexity of the force algorithm is given in Section 6.10.

6.4.1 Bumper springs

However, and we make a small sidestep here, the negligible effect of the repulsion forces could be caused by a bad ratio between attractive and repulsive forces. To investigate this, we have introduced a new type of springs between all the nodes, i.e., bumper springs. These springs do not exert an attractive force, but only a repulsive force, similar to repulsion forces. The idea is that these bumper springs make sure that the categories push themselves away from other categories.

As stated, the bumper springs only try to reach their zero energy length in the case they are too short. A bumper spring that is too long does not try to reach a shorter length. Therefore, the bumper springs do not follow Hooke's spring law as defined in Equation 6.4, but an alternative spring law:

$$f_{bumper}(x) = \begin{cases} A \cdot (x - x_0) & \text{when } x < x_0 \\ 0 & \text{otherwise} \end{cases} \tag{6.8}$$

As could be expected, the effect of the bumper springs was also negligible, for the same reasons as we could neglect the repulsion forces. Furthermore, the additional $O(N^2)$ springs asked a lot more computation power. Therefore, we might as well use repulsion forces, but since we have rejected those, we reject the bumper springs as well.

6.5 Time step

At each iteration, the position $\mathbf{p}_v$ of a category v is updated according to:

$$\mathbf{p}'_v = \mathbf{p}_v + F(v) \cdot \Delta t \tag{6.9}$$

where Δt is the time step. Initially, the time step is set to a relatively high value to let the really stretched out springs do their work, which results in a fast placement which is not yet perfect. Through time, the time step is diminished to make sure the springs can reach their minimum energy state. A small time step removes the oscillating effect which occurs when the spring is traversing its zero energy length each iteration of the calculation process. As the time step gets smaller and smaller, the graph is fine tuned. Specifically, we use:

$$\Delta t_{i+1} = C \cdot \Delta t_i \tag{6.10}$$

where i is the number of the iteration and C is a constant with $0 \leq C \leq 1$.

6.6 Stiffness

The stiffness of a spring reflects the rate at which the spring tries to reach its zero energy length. The stiffer the spring, the faster the spring tries to reach this length. As stated, we define springs between the sibling-categories in the graph, where siblings are defined as categories with the same supercategory, see Equation 6.3. It is, therefore, possible for categories to be siblings through multiple supercategories. This occurs, for example, when a set of subcategories have the same set of supercategories, such as in Figure 5.2. The middle two categories have the same set of supercategories, namely, the two supercategories shown. According to the definition (see Equation 6.3), we would create multiple springs between these categories, because they are siblings over multiple supercategories. To decrease the computational complexity, at most one spring is created between any two categories. Therefore, to reflect the existence of multiple springs, the one existing spring's stiffness is multiplied by the number of springs it represents.

In Figure 6.3, the categories c_4 and c_5 are siblings through multiple supercategories, e.g., c_1 and c_2. According to the definition, there should be two springs between these two categories, but we only place one spring with double stiffness between these two categories.

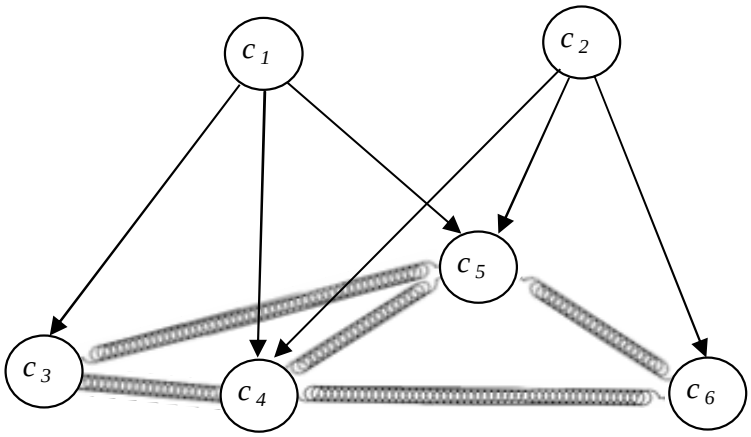


Figure 6.3: *Spring between c_4 and c_5 has increased stiffness, while c_3 and c_6 are considered lonely categories.*

Another problem that can be solved with increased spring stiffness is the behavior of 'lonely categories'. Categories that have only one supercategory, and siblings with multiple supercategories, are called 'lonely categories'. Formally speaking, a category $v \in V$ is considered a lonely category when $v \in Lonely(V)$, where the collection of lonely categories is defined by:

$$Lonely(V) = \{v \in V | \#super(v) = 1 \ \wedge \ \forall_{s \in sibling(v)}(\#super(s) > 1)\} \quad (6.11)$$

where $\#C$ denotes the number of elements in collection C. Since the siblings of these lonely siblings receive their force vectors from multiple supercategories, and these supercategories move often in opposite directions, the total force vector of these siblings is significantly smaller than the force vectors of the lonely categories. The lonely categories, therefore, tend to move away from the siblings while enlarging the rectangle of the supercategory. To cope with this unwanted behavior, the stiffness of the springs attached to the lonely categories is doubled to keep them close to their siblings with multiple supercategories. Note that ghost categories (see Section 5.2.1) can also be seen as lonely categories, and, therefore, have stiffer springs attached to them.

In Figure 6.3, the categories c_3 and c_6 are considered lonely categories, because they each have only one supercategory, e.g., c_1 and c_2 respectively, but siblings with multiple supercategories, e.g., c_4 and c_5. Note that in practice there usually are more categories like c_4 and c_5, so the difference in spring force between lonely nodes and their siblings is large.

6.7 Change spring lengths

Since two categories that are connected by a spring should not overlap when they do not have an overlapping region, it is important to correctly set the zero energy length of the spring between these two categories. Each spring in the force model tries to get as closely to its zero energy length as possible, while following the spring law (see Equation 6.4). Since the spring is connected to the center point of the shape which represents the category, the length of the spring should be at least the sum of the radii of the two shapes if we want one length for arbitrary relative positions.

Initially, the exact size of the shape of a category is not know, since initially the atoms are placed randomly across the screen. This means that the supercategories initially have very large shapes and they are all overlapping. The actual radius of these shapes is, thus, very large and, therefore, we approximate the radius by using the square root of the weight of a category. Therefore, the zero energy length x_0^{uv} of a spring between two categories u and v that are not overlapping is initially defined as a function on the weights of u and v, i.e.,

$$x_0^{uv} = B \cdot (\sqrt{w_u} + \sqrt{w_v}) + m \quad (6.12)$$

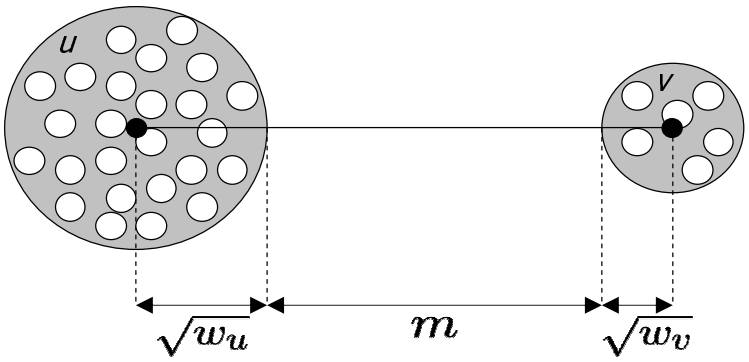


Figure 6.4: *Initial spring length.*

where w_x is defined as the total number of atoms under category x, i.e., the number of subcategories which have no subcategories. Constant B denotes the radius of an atom. Term m denotes a margin, as can be seen in Figure 6.4.

We take the square roots of the weights, because the square root of a number of items is generally a good estimate for the number of items in horizontal direction, and the number of items in vertical direction. Of course, this relies on the assumption that the items are uniformly distributed and lie in a square or a circle. In our case, the subcategories in a supercategory are not uniformly distributed and the shape of the supercategory is not a square or a circle, so we only use this definition as an initial zero energy spring length.

Therefore, during the placement process, the lengths of the springs are adjusted each step of the iteration to better reflect the actual distance between the two endcategories of a spring. To avoid unwanted overlap between categories, the length of the spring should at least be the sum of the radii of the two endcategories. First we define the radius to be half the diagonal of the rectangle that represents the category. Then, let r_u and r_v be the radius of categories u and v respectively, then:

$$x_0^{uv} = r_u + r_v + m \tag{6.13}$$

where x_0^{uv} is the zero energy length of the spring between two categories u and v. Term m denotes a margin, i.e., the distance between the two rectangles that represent the categories, as denoted by Figure 6.5a.

It is fairly possible for two categories that are connected to a spring, to have an overlapping region. In accordance with the Cristal model, this overlap means that the two categories have at least one subcategory in common. The length of the spring should account for the overlapping space. This means that the zero energy length of the spring should reflect the sum of the radii of the two categories minus

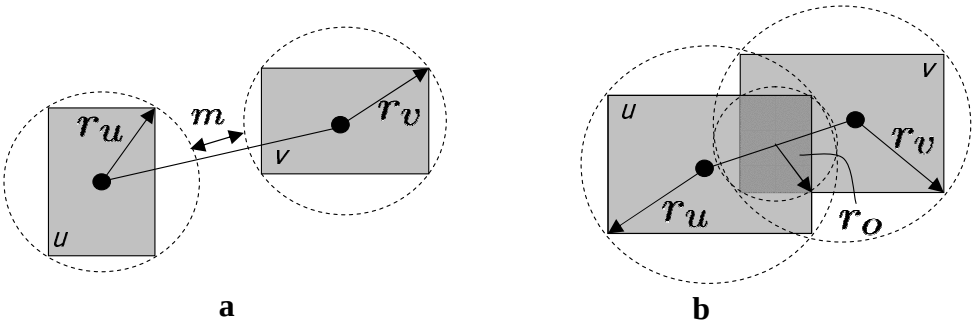


Figure 6.5: *Spring lengths:* ***(a)*** *Spring length when the two categories do not overlap,* ***(b)*** *Spring length when the two categories overlap.*

the diameter of the overlapping region. Since it is difficult to determine the actual diameter of the overlapping region, we approximate this diameter by taking twice the square root of the weight of the overlapping region. Let r_u and r_v be the radius of categories u and v respectively, then:

$$x_0^{uv} = r_u + r_v - 2r_o - m \tag{6.14}$$

where x_0^{uv} is the zero energy length of the spring between two categories u and v, r_o denotes the radius of the overlapping region of u and v, and term m denotes a margin, as denoted by Figure 6.5b. The radius of the overlapping region is defined as half the length of the actual diagonal of the rectangle that encloses the categories that make up the overlapping region.

6.7.1 Other spring length options

We described the best working zero energy spring lengths above. We have experimented with a number of different ways to change the spring lengths to decrease unwanted overlap. As a sidestep we give a number of alternatives and the reasons why they have been ruled out:

- *Change spring length according to overlap.*
  When the sum of the radii of two categories that are connected with a spring is greater than the distance between the two center points of the categories, the two categories are overlapping. In this case, the length of the spring is increased, otherwise, the length of the spring is decreased, i.e.,

$$l_s = \begin{cases} l_s \cdot (1+e) & \text{when } r_u + r_v > |\mathbf{p}_u - \mathbf{p}_v| \\ l_s \cdot (1-e) & \text{otherwise} \end{cases} \tag{6.15}$$

Another option is to increase the length of the spring between two categories when the actual shapes of the categories are overlapping and decrease the length of the spring otherwise, i.e.,

$$l_s = \begin{cases} l_s \cdot (1+e) & \text{when } u \text{ is overlapping with } v \\ l_s \cdot (1-e) & \text{otherwise} \end{cases} \tag{6.16}$$

where l_s is the length of the spring between categories u and v, r_u and r_v are the radii of category u, respectively category v, $\mathbf{p}_u$ and $\mathbf{p}_v$ are the center positions of category u, respectively category v, and e is a small constant. These spring length change functions have been ruled out because the spring lengths just change too dynamically. The change of one spring's length leads to the change of another spring's length and so on. It is, therefore, fairly possible that the length of the springs only increase, which leads to a graph that grows unbounded.

- *Change power of the weight according to the radius per spring.*
  This option uses the initial length of the springs, which is defined by Equation 6.12. In short, the spring length is set to the sum of the square roots of the number of atoms below the two categories. Since taking the square root of a number is the same as raising that number to the power of 0.5, we can change that power to alter the spring length. More specifically, when the power is increased, this means that categories with small weights, i.e., little atoms, have substantially longer springs connected to them. Decreasing the power results in categories with big weights, i.e., many atoms, to have substantially longer springs connected to them.

$$a = \begin{cases} a \cdot (1+e) & \text{when } r_u + r_v > |\mathbf{p}_u - \mathbf{p}_v| \\ a \cdot (1-e) & \text{otherwise} \end{cases} \tag{6.17}$$

As with the previous option, we can also check to see whether the actual rectangles of the categories are overlapping and adjust the power accordingly.

$$a = \begin{cases} a \cdot (1+e) & \text{when } u \text{ is overlapping with } v \\ a \cdot (1-e) & \text{otherwise} \end{cases} \tag{6.18}$$

$$l_s = B \cdot ((w_u)^a + (w_v)^a) + m \tag{6.19}$$

where a is the power which is used in the latter equation, r_u and r_v are the radii of category u, respectively category v, $\mathbf{p}_u$ and $\mathbf{p}_v$ are the center positions of category u, respectively category v, e is a small constant expression, l_s is the length of the spring between categories u and v, B is the stiffness

of the spring, w_u and w_v are the weights, i.e., the number of atoms, of categories u, respectively category v, and m is a margin.
It turns out that the springs between atoms remain too long and the springs between supercategories are relatively too short. Furthermore, there is still a lot of overlap which is not wanted, because the lengths of the springs are not yet optimal. However, the system is a lot less dynamic, and does not lead to a graph that grows unbounded.

- *Change overall power of weight according to the radius.*
  The previous options describe a way to change the power which is used in accordance with the number of atoms below a category to resemble the radius of that category, which in turn is used to determine the length of the spring. In the previous options, this power is determined per spring. Of course, it is also possible to determine this power for all the springs at the same time. This means that all radii of all categories are calculated using the same power.

$$a = \begin{cases} a \cdot (1+e) & \text{when at least two categories are overlapping} \\ a \cdot (1-e) & \text{otherwise} \end{cases} \tag{6.20}$$

$$l_s = B \cdot ((w_u)^a + (w_v)^a) + m \tag{6.21}$$

  where a is the power which is used in the latter equation, e is a small constant expression, l_s is the length of a spring between two categories u and v, B is the stiffness of the spring, w_u and w_v are the weights, i.e., the number of atoms, of categories u, respectively category v, and m is a margin. Note that all the zero energy lengths l_s of all the springs s are updated with the new power a.
  The initial results were appealing while using this option, but we soon found out that it is fairly possible for two categories to be constantly overlapping. The result is that the power is constantly increased, while the overlap is not reduced. Therefore, the lengths of the springs only increase, which leads, again, to a graph that grows unbounded.

6.8 Multi Dimensional Scaling

A different approach we have investigated originated from the area of multi dimensional scaling (MDS). In essence, the main problem is to find a hierarchical placement for the categories in which categories that have a close relation are placed closer to each other than categories which have no relation. In our case, a

close relation is defined between categories that have the same supercategory, i.e., siblings.

Standard MDS algorithms use a distance function between different objects and try to place the objects on the screen with respect to their relative distances. This distance function is, in essence, a mapping from a multi dimensional space to a two dimensional space, or in some cases a three dimensional space. In our case, the distance function should return a relatively low distance for categories that are siblings and a high distance for categories that have nothing in common. To denote the distance between two categories u and v, we count the number of edges in the shortest path p from u to v, with the restriction that the closest common supercategory of u and v is a member or path p.

Let $G = (V, E)$ be a directed acyclic graph where V is the collection of categories, i.e., the collection of vertices and E is the collection of relations between categories, i.e., the collection of edges between the vertices. Thus, each vertex in the graph denotes a category.

First define a distance function between two categories that counts the edges between two categories:

$$d(u, v) = \text{number of edges on the shortest path from } u \text{ to } v \tag{6.22}$$

Then, the set of closest common supercategories of two categories u and v is defined by:

$$CCS(u, v) = \{x \in V | \forall_{y \in V}(d(x, u) + d(x, v) < d(y, u) + d(y, v))\} \tag{6.23}$$

Finally, the distance function used for this MDS algorithm with $ccs_{uv} \in CCS(u, v)$ is defined as the length of the path from u to v over the smallest common supernode ccs_{uv}, i.e.,

$$d_{mds}(u, v) = d(ccs_{uv}, u) + d(ccs_{uv}, v) \tag{6.24}$$

Since the positions of the atoms dictate the positions of the supercategories, we define a fully connected graph of springs between the atoms. The zero energy spring length of these springs is defined by the distance function $d_{mds}(u, v)$.

As stated, the MDS approach defines a full graph of springs between the atoms. After a few iterations of the force algorithm, this results in a somewhat circular representation of atoms. The proposed distance function as function that describes the spring length between two atoms proved to be a good estimate for the actual distance between the atoms. There is, however, no way of ensuring that the supercategories can be drawn as enclosing rectangles around their subcategories. For example, since the atoms are laid out in a circular structure, the

supercategories of the atoms that happen to lie on the outside of the circle tend to grow very large and overlap each other, causing a lot of unwanted overlap. It is also possible that two supercategories with each two atoms form a cross and, thus, create a lot of unwanted overlap. In an MDS algorithm there is no way we know of that enforces better placement of the atoms such that these drawbacks are avoided. Therefore, since the MDS algorithm does not take the hierarchical structure into account, and because it generates circular structures that do not comply with the proposed rectangular inclusive shapes, we had to rule out the MDS approach.

6.9 Summary

For reference, Appendix A lists the pseudocode for our force directed algorithm. To summarize, the following steps are used in the force model:

- *Add ghost categories*
  We add ghost categories to the graph to show the implicit relation of a category being contained in another category, which otherwise would be shown as an explicit relation.

- *Define springs between siblings*
  To create clusterings of categories, we define springs between sibling-categories as opposed to the general force directed algorithms with springs defined on the edges of the graph. Siblings are defined as categories that share the same direct supercategory.

- *Increase spring-stiffness to resemble multiple springs*
  A spring between siblings that have the same set of supercategories has increased stiffness to resemble multiple springs. We only define one spring to decrease computational complexity.

- *Increase spring-stiffness for 'lonely categories'*
  Springs of 'lonely categories' have increased stiffness to avoid the tendency of these 'lonely categories' moving away from the rest of the graph, resulting in large rectangles for their supercategories.

- *Initial zero energy spring length*
  Initial zero energy spring length between two categories u and v:
  $x_0^{uv} = B \cdot (\sqrt{w_u} + \sqrt{w_v}) + m$

- *Iterate over springs*
  Repeatedly iterate over all the springs to calculate the force vectors for each node, in order to minimize the total energy function. Execute the following steps each iteration:

- *Calculate force vectors*
  Calculate the total force vector for each category v, according to the springs connected to that category.

- *Move categories*
  Move the categories to their new positions, according to the calculated force vectors.

- *Change spring lengths*
  Because the square root is not a good estimate for the radius of a group of categories, we change the zero energy spring length x_0^{uv} of the spring between two categories u and v every step of the iteration as follows:

  - No wanted overlap: $x_0^{uv} = r_u + r_v + m$
  - Wanted overlap: $x_0^{uv} = r_u + r_v - 2r_o - m$

6.10 Complexity

In this chapter, we have defined an iterative force directed algorithm. Each iteration of the algorithm, the force vectors that result from the springs between the categories have to be calculated. Furthermore, each category has to be moved according to the resulting force vector from its spring. Subsequently, the zero energy length of each spring has to be adjusted. As a result, the number of springs determines the complexity of the algorithm.

In the worst case, a graph with N categories consists of one root-category and the other $N-1$ categories are subcategories of the root-category. This means that there are $N-1$ siblings, each with the root-category as supercategory, resulting in $\sum_{i=1}^{N}(N-1-i)$ springs, which is of order $O(N^2)$.

Each iteration, the forces resulting from these springs need to be calculated. The more categories the classification contains, the more iterations are needed to calculate an appealing layout. Therefore, the number of iterations is of order $O(N)$. Furthermore, moving N categories to their new positions is an $O(N)$ operation. Subsequently iterating over all springs to change their zero energy spring lengths is an $O(N^2)$ operation. To sum up, the total complexity is $O(N) \cdot (O(N^2) + O(N) + O(N^2))$. Which evaluates to $O(N^3)$.

As stated in Section 6.4 we have omitted the repulsion forces for one reason because they require an additional $O(N^2)$ calculation per iteration. Since the total complexity per iteration is already of $O(N^2)$, this does not seem to matter very much. However, the complexity of calculating the repulsion forces requires exactly N^2 steps, because for all categories, all categories have to be examined to acquire the repulsion between them, while the iteration complexity of $O(N^2)$ is

only a worst case running time. On average, for a number of real world information sets, the amount of springs is about $10N$, which is $O(N)$. This results in a total average case complexity of $O(N) \cdot (O(N) + O(N) + O(N)) = O(N^2)$. Therefore, omitting the additional $O(N^2)$ for calculating the repulsion forces per iteration does reduce the complexity of the total algorithm.

6.11 Results

The algorithm is capable of showing graph structures with up to 1,000 categories on a high-end desktop computer. To give an impression on the running time, Table 6.1 shows the running times for different classification structures on an Intel Pentium 4 2.8 with 512MB RAM.

# Categories:	Running time:
50	8 sec
100	14 sec
358	49 sec
986	2:36 min

Table 6.1: *Running times of algorithm.*

We found that narrow graph (or tree) structures tend to cause less unwanted overlap. In practice, the classification structures at SN are narrow structures, i.e., graph structure in which the categories near the root category have little subcategories. Unfortunately, however, the algorithm was not capable of removing all unwanted overlap. In these cases, the user has to move the categories around with the mouse to be able to see the exact structure of the information set.

A tree structure with 100 categories is shown in Figure 6.6. The algorithm is fairly capable of showing tree structures, as can be seen in the figure.

A graph structure with 52 categories is shown in Figure 6.7. The portion inside the red circle shows some overlap that we could not avoid.

Figure 6.8 shows a graph structure with 986 categories. The two red circles again denote some unwanted overlap.

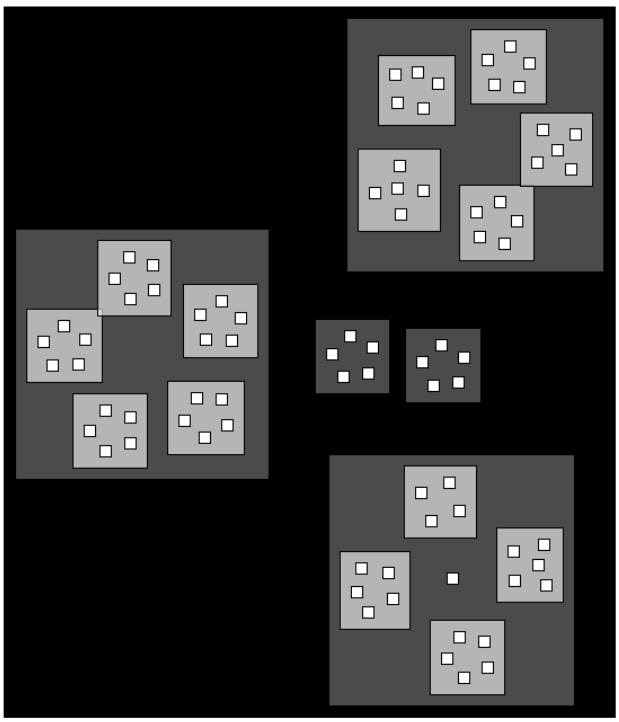


Figure 6.6: *Visualization of tree structure with 100 categories.*

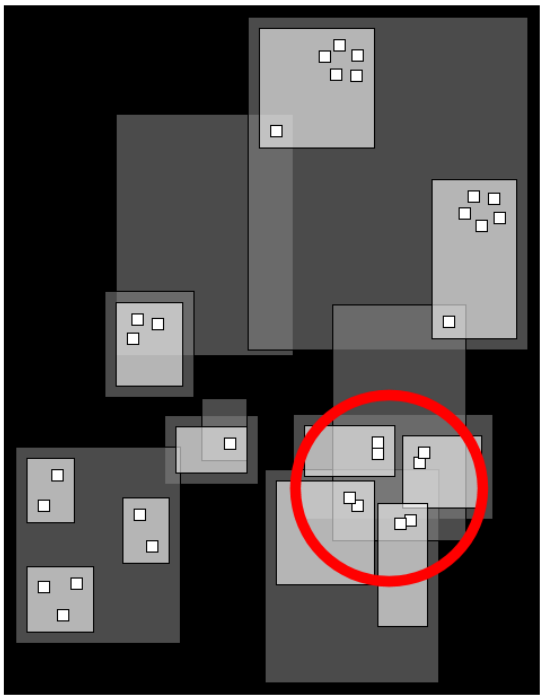


Figure 6.7: *Visualization of graph structure with 52 categories.*

Figure 6.8: *Visualization of graph structure with 986 categories.*

Chapter 7

Evaluation

Once the initial prototype was developed, it was necessary to determine the capabilities and the potentials of the visualization technique and the visualization tool, CristalView. This judgment needed to be performed such that both the good points and drawbacks of the visualization were revealed.

In Chapter 3 we have defined a number of requirements for a visualization of the classification structures of a Cristal. Subsequently, we have come up with a visualization method and a prototype of our visualization tool, CristalView (see Chapter 5). In this chapter we evaluate the visualization tool CristalView. For this, we have conducted a user experiment with a number of participants, which is described in the following sections.

7.1 Methods of testing

In this section we describe the test methods for the requirements we evaluate during the user experiment. The test setup is explained in the next section. We number the test methods starting with E1 and the tested requirements (see Chapter 3) are denoted between brackets.

- E1: *Improvement over Variable Editor* (RF7, RF1, RF2)
  The most important factor that has to be tested is whether or not the visualization tool is an improvement over the previous visualization tool, the Variable Editor. To this end, a comparison has to be made between the two visualization tools. We, therefore, ask the user several questions about the structure of the information set and measure the time needed to answer the questions using the two visualization tools. The questions have to be different for the two visualization tools to avoid the effect of memory, but the insight required to answer them has to be the same. Also, the questions have to address all kinds of possibilities the Cristal model offers, to test whether

the user understands the full capabilities of the Cristal model. Since some participants have no experience with the Cristal model at all, the initial understanding of the model is tested by these questions as well.

- E2: *Pinpoint deficiencies* (RF8, RF9, RF10)
  Another intention is to enable the user to create data without unnecessary objects. Also, the chosen data structure, e.g., a tree or graph structure, has to be the simplest structure possible. To test this, the users are confronted with a Cristal that has deficiencies in it. They are asked to identify the deficiencies and how they would fix them. Also they are asked questions on whether or not they would have created the Cristal the same way if they had to do it themselves. Of course, it is necessary that the users have some kind of experience with the visualization tool, so this part of the test should be carried out after the user has gained some experience with the tool. Note that the required time per action is not really an issue here, because it is more important that certain deficiencies are identified.

- E3: *Ease of use* (RU1, RU2)
  Finally, the program has to be easy to use. This is examined through a concluding survey with questions on the user interface, the ease of navigating, etc. Also, the users can describe certain improvements on the program they would like to see, as well as general remarks.

7.2 Test setup

We start the user experiment with a short explanation of the visualization tool CristalView. The basic functions are explained and demonstrated shortly to give the user an impression of the program. In case the user is not familiar with the Variable Editor, the functionality of this editor is explained shortly to the user. Of course, the contents of the user test is explained to the user.

The user experiment consists of three parts. The first part is a comparison between the Variable Editor and CristalView. In the second part the participant has to pinpoint certain strange objects in the dataset. The third and final part of the test consists of a survey that asks the user's opinion on the visualization tool CristalView.

In the first part of the test (see E1) the user is asked to answer some questions about the information set on the screen.

To be able to test the visualization of different aspects of the information set, three types of questions were asked, i.e.,

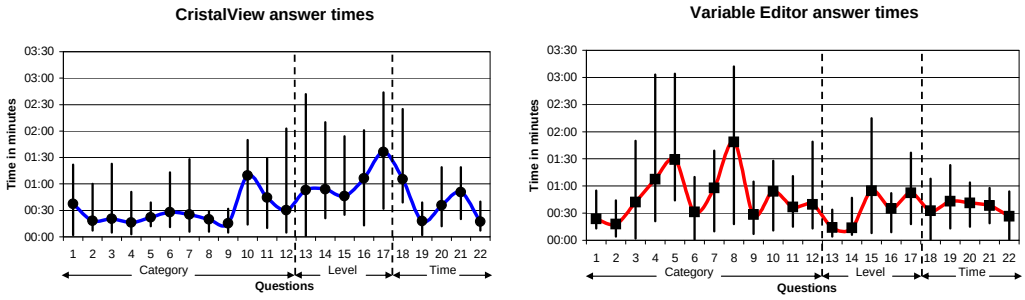
- Questions regarding categories
- Questions regarding levels
- Questions regarding time

The time required to answer the questions is measured. This is done for the Variable Editor as well as CristalView, which allows us to compare the measured times and draw conclusions on the comprehension of the information structure. The information we use is not real world data, but a specially constructed information set with descriptive category-names. Since we are letting the participants use the Variable Editor as well as CristalView and we want to compare the two, we use two information sets with exactly the same structure, except that the descriptive names are different. This allows us to ask different questions for each information set which prevents the effects of participants memorizing questions, but in fact we are asking the same questions, because the structure of the two information sets is the same. The complete structure of the information set, as well as the questions asked about it can be found in Appendix C.

In the second part of the test (see E2), the user was asked to look at the current Cristal and see if there are any strange objects on the screen. If the user identified such objects, the user was asked to precisely describe what is wrong. After that the user was asked a couple of questions on whether or not he/she would have created the same Cristal given the data. We used a specially constructed dataset with a number of deficiencies in it. By counting the number of deficiencies found by the participant, we can draw conclusions on whether or not the user is capable of using CristalView to pinpoint errors and unnecessary objects in the data. The information set we used is described in Appendix C.

The last part of the test (see E3) asks the user to give his opinion about certain aspects of the program, e.g., navigation, ease of use etc. This is done through a small survey in which the user can propose certain improvements and give general remarks.

Since these methods of testing (also see Section 7.1) require an extensive test which takes a few hours, we could not use a large amount of participants. On the other hand, to be able to draw well-funded conclusions, we should test the program with as many participants as possible. Therefore, we use a group of about ten participants, but each test is slimmed down to around one hour.



a **b**

Figure 7.1: ***(a)*** *CristalView average answer times. Vertical lines denote minimum and maximum answer times.* ***(b)*** *Variable Editor average answer times. Vertical lines denote minimum and maximum answer times.*

7.3 Results

Since the user experiment is divided into three parts, we discuss the results of the three parts separately before we draw conclusions.

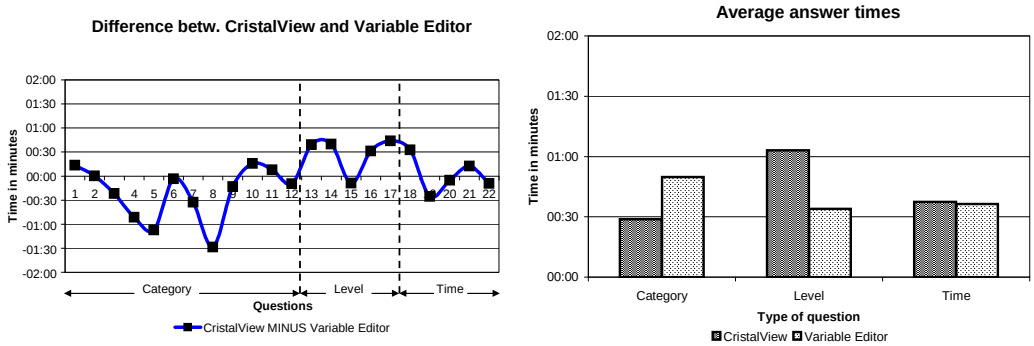
Part one - Comparison

The participants were divided into two groups. The first group tested CristalView first and subsequently the Variable Editor, while the second group started out with the Variable Editor.

Figure 7.1 shows the averages of the measured answer times for the two tools. The test consisted of 22 questions regarding the three aspects of the visualization, see Appendix C. The three different types of questions are denoted below the horizontal axis of the graphs. For each question, the vertical line denotes the maximum and minimum answer times, while the trend denotes the average answer time per question.

Figure 7.1(a) shows the answer times for CristalView. It shows that the category-questions are answered quite quickly, which means they were relatively easy to answer. The level-questions were a bit harder to answer, so they took more time. The question regarding time were a little easier to answer, but still more difficult than the category-questions. Furthermore, the difference in answering times is quite significant, as shown by the vertical lines. Some participants required considerable more time to answer questions than other participants. This effect is most noticeable regarding the level-questions, probably because these questions were considered to be harder to answer than the category-questions.

Figure 7.1(b) shows the answer times for the Variable Editor. Opposite to



a **b**

Figure 7.2: ***(a)*** *Difference between CristalView and Variable Editor average answer times,* ***(b)*** *Average answer times per type of question.*

the previous graph, this graph shows that the category-questions are quite hard to answer, while the level- and time-questions are relatively easy to answer. Again, the difference in answering times is the largest with the toughest questions, which can in this case be found amongst the category-questions.

When we take the difference between the two graphs, we acquire Figure 7.2. Figure 7.2(a) shows the CristalView average answer times minus the Variable Editor average answer times. This means that the participants answered the questions faster in CristalView when the trend is below 0, and that the questions were answered quicker in the Variable Editor when the trend is above 0. Again, the figure shows that the usage of CristalView is considerably more efficient regarding Categories, but when it comes to Levels, the Variable Editor is more efficient. For questions regarding Time, both tools are equally efficient. To clarify this statement, consider Figure 7.2(b). This figure shows the difference in average answer times per type of question between the two tools. It clearly follows from this figure that CristalView clarifies the category-structure, but that the Variable Editor is more effective regarding the level-structure. When it comes to questions regarding time, both tools are equally effective.

Part two - Deficiencies

As stated in Section 7.2, the user is asked to look at a classification structure of a Cristal and see if there are any strange objects on the screen. Moreover, the participant is asked to precisely describe what is wrong. We used an information set containing a classification structure of regions of The Netherlands, see Appendix C. The dataset contains all municipalities of The Netherlands at the lowest

level. These municipalities are contained in 42 groups of municipalities. These groups of municipalities are contained in 12 provinces, where the provinces are contained in 4 quarters, one for each wind of the compass. These 4 quarters are contained in the root category ”Total”.

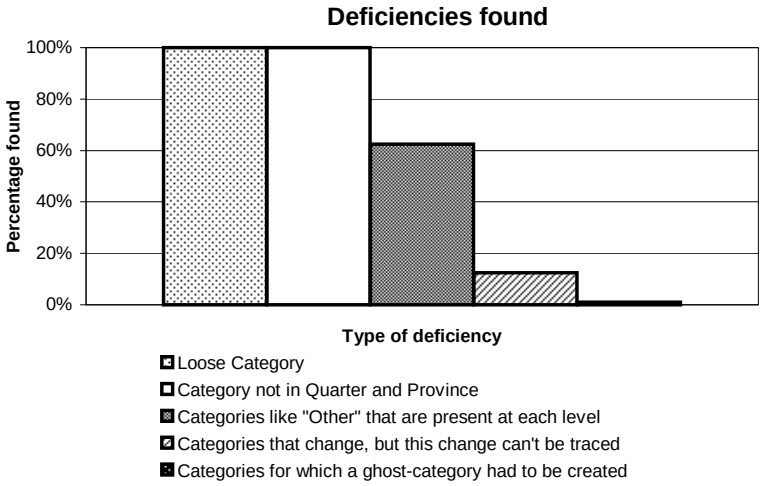


Figure 7.3: *Percentage of participants that pinpointed a deficiency.*

There were several deficiencies to be found in the classification structure in regions of The Netherlands. For example, there was a ”Loose Category”, i.e., a category that was not contained in the root category ”Total” and not in any level. Furthermore, there was a category that was only contained in the root category ”Total”, but not in any lower level such as ”Province” or ”Quarter”. Also there were categories like ”Other” that were present at each level in the classification structure. A lot of municipalities were at some point in time joined, or split. These joining and splitting operations were not traceable, so could be marked strange. Finally, there were categories for which the set of subcategories is a part of the set of subcategories of another category. For these categories a ghost-category had to be created (see Section 5.2.1). This can also be marked as strange.

Figure 7.3 shows the results of the second part of the experiment. Every participant noticed the strange ”Loose Category” that did not belong to any root category, as well as the category that did not belong to any province or quarter. More than half the participants noticed the categories like ”Other” that were present at each and every level. The municipalities that were split up, or joined together were only noticed by ten percent of the participants, and the categories that have exactly the same set of supercategories were not noticed at all.

Part three - Survey

As stated in Section 7.2, the third part of the user experiment consisted of a small survey in which the participants could express their feelings about the tool. The general opinion was that CristalView is very much a welcome contribution to the Cristal package. Every participant was willing to use the tool to gain insight in a Cristal.

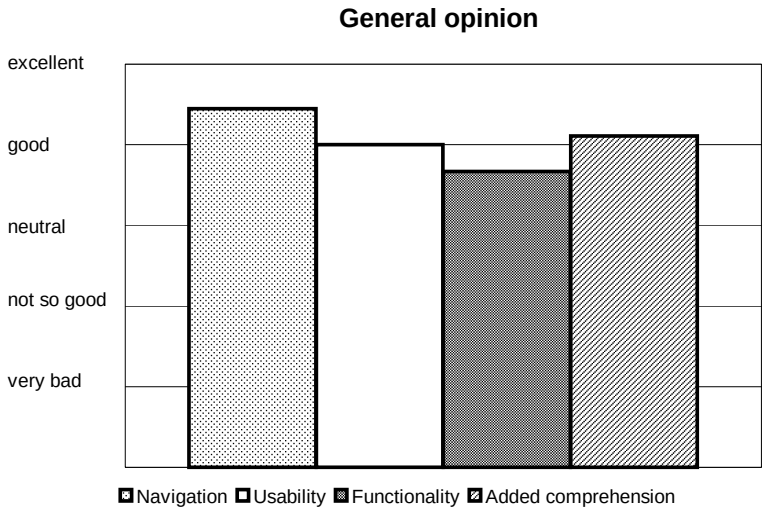


Figure 7.4: *General opinion about CristalView.*

Figure 7.4 expresses the general opinion of the participants, which was very positive at the different points. Navigating through the dataset was judged to be very good, and also the usability of the program was quite good. The participants thought the tool contained enough functionality, but since they all had ideas of functionality to add, they generally judged the functionality to be good. Finally, the participants all found their comprehension of the Cristal model very much increased.

Participant remarks

Of course, when a number of participants test a program which is still in development, they all have an opinion about it and they all miss certain functionality which they would like to be added to the program. We discuss a number of remarks that were expressed by several of the participants. In Section 5.7 we have described how these remarks were dealt with in the final application.

- Search functionality. The Variable Editor contains search functionality, and almost all of the participants were looking for a search option in CristalView to get a quick start in finding their way in the information set. Nearly all participants mentioned this search functionality as a function that is very much needed in an application that provides the user insight in an information set.

- Folding and unfolding of categories. The tested prototype contained a category-fold function, which folded a category (and all its subcategory) to a small circle when the user right-clicked the category. The idea behind this was to avoid visual clutter and enable the user to focus on aspects of interest. During the user experiment, however, a few participants accidentally folded a number of categories and they were very surprised. The participants did not know what they had done, and were unable to trace the folded category. The described effect of surprise is enhanced when a root category is accidentally folded, because in that case a large portion of the image on the screen will vanish into a small circle and users panic and ask themselves what they have done.

- Select multiple levels at the same time. The prototype only allowed to select one level at a time, which colored the categories in the selected level with the specified color. A few questions during part one of the user experiment could be answered by taking the intersection of two levels, which was not possible, because only one level could be selected at a time. Therefore, participants remarked that they would like to be able to select multiple levels at a time. When implementing the selection of multiple levels at the same time, it is possible that categories reside in two different levels that are both selected. The overlapping level-region would have to be colored with the two colors of the two levels at the same time. This would result in a blended color or some sort of pattern with multiple colors, which would both be very distracting. Therefore, to avoid these problems, we decided not to implement the selection of multiple levels at the same time.

- Degree of abstraction (see also: van Ham, van Wijk [27]). During the user experiment, participants remarked that it would be more convenient to have some degree of abstraction. Of course, the folding functionality addresses this problem, but since this functionality proved to be very confusing, some users requested a way to only show the high level categories (such as the root categories). In combination with zooming functionality, the lower level categories (such as the atoms) could be shown when the user zooms in on a specific area of the dataset to see a more detailed picture. This would decrease visual clutter and, therefore, increase the comprehension of the

information set. We decided not to implement a degree of abstraction, since the folding functionality provides enough possibilities to our opinion.

7.4 Conclusion

In this section we draw conclusions on the results of the different parts of the user experiment. We address the test methods defined in Section 7.1 separately.

- E1: *Improvement over Variable Editor*
  From the results of the first part of the user experiment (see Figures 7.1 and 7.2) follows that CristalView is more effective regarding the category structure, but that the Variable Editor is more effective when it comes to the level structure. Regarding the time structure, both tools are equally effective. An explanation for these results can be found when we look at the design principles of both tools. CristalView was designed to visualize the category structure. Visualization of levels was added later by coloring the categories that form a certain level. The Variable Editor was designed the other way around, because it focuses mainly on the hierarchy- and level structure and the categories were added to complete the lot. It is, therefore, not very surprising that CristalView does a better job at visualizing categories, while the Variable Editor is better off visualizing the level structure. With regard to the time structure, both programs have an appropriate way of dealing with categories that are predecessors or followers of each other, that works for both programs equally well. Furthermore, from the first part of the test in combination with the concluding survey of the experiment, we can conclude that the participants gained more insight in the full capabilities of the Cristal model, as well as a broader understanding of the Cristal model.

- E2: *Pinpoint deficiencies*
  Also, in Chapter 3 we expressed the need for users to create data with some degree of symmetry and without unnecessary objects. The results of part two of the user experiment show that every participant remarked the outlying categories and that more than half of the participants noticed the strange categories like ”Other” that existed at every level in the information set. From this we can conclude that users will create better information sets when they can really see what they are doing. They can really pinpoint unnecessary objects and remove them from the information set. A few participants even wondered why some parts of the image ’looked better’ to them, before discovering that these good looking parts were in fact tree structures

instead of graph structures. Therefore, it seems likely that users will create information sets with an increased degree of symmetry which only adds to the comprehension of the information.

- E3: *Ease of use*
  Finally, Chapter 3 states that the program should be easy to use, which means that the user interface of the program should be clear to the user. From the results of the first part of the experiment already follows that participants find their way into the program quite fast, according to the measured answer times. Often the user finds an answer in under ten seconds, which can be regarded as fast. Furthermore, from the concluding survey of the experiment follows that navigating, usability and functionality are all rated good to excellent. All participants stated that they would like to use CristalView to get insight in a particular Cristal, so we state that the program is easy to use.

In general, people who have experience with computer programs are used to textual representation of information. Of course, this does not necessary mean that a textual representation is more effective. While a visual representation of information can be visually appealing, people often want to know what the underlying textual information is. It takes, therefore, some time for most people to get used to a visualization tool that reduces the amount of textual representations with a theoretically more comprehensive visual representation of the information set. In this case, CristalView is clearly a visual representation of information, while the Variable Editor uses a textual representation of information. It is, therefore, good to test this in practice with a user experiment, and according to the above, the overall conclusion is very positive regarding the visual representation of CristalView. However, a problem still remains with the visualization of the level structure. Perhaps it is possible to combine the best features of CristalView and the Variable Editor which gives the best of both worlds, i.e., the category visualization from CristalView combined with the level visualization of the Variable Editor.

Chapter 8

Conclusions

This chapter presents conclusions based on working with the visualization method explained in this thesis, subdivided into possibilities and limitations. The final section gives a number of recommendations for further research and possible improvements.

8.1 Possibilities

- *Inclusion and overlap*
  The visualization of parent rectangles surrounding their child rectangles to represent part-whole relations was already known to present a visually clear and comprehensive image. The combination with overlapping rectangles to represent child objects with multiple parent objects also proved to be very useful. An additional advantage is the intrinsic visualization of transitive parent-child relations, in other words, parent-grandchild relations.

- *Interaction*
  CristalView provides a clear picture of the entire graph without overloading the user with information. Using this overview image, the user can quite easily gain insight in the overall structure of the graph. Because of the many ways to interact with the information set, the user can navigate or zoom in to a region of interest to be able to see more detail. Moreover, the user can drag the rectangles to new positions to provide an even more appealing image. Since the user can also select which categories should be labeled, and even drag the labels to a new location, well known problems regarding labels overlapping regions of interest are avoided.

- *Positive evaluation*
  Following from the user experiment we have conducted to evaluate CristalView (see Chapter 7), we have found that CristalView can help users to pinpoint deficiencies in the classification structure and is easy to use. Since CristalView does not support editing of the structure of the classification, an edit-tool has to be used to clear out the possible deficiencies, for example the Variable Editor (see Appendix B). However, since CristalView does support editing of the attributes of a category, level or hierarchy, changing these attributes can be done very easily while viewing a classification structure.

8.2 Limitations

- *Unwanted overlap*
  Since overlapping rectangles really denote someting, namely a child category with multiple parent categories, we do not want rectangles overlapping when this relation is not present. The force-directed layout algorithm should take care of that, however, we have not succeeded in completely avoiding unwanted overlap. On the other hand, the user can move categories to another position by dragging the categories with the mouse, such that there is no unwanted overlap.

- *Graph shape*
  We found that narrow graph structures tend to cause less unwanted overlap. Fortunately, in practice most classification structures used at SN are narrow structures, i.e., graph structures in which the categories near the root of the graph have little subcategories (about 5). The structures at SN often are narrow at the top, but very wide at the bottom, i.e., the supercategories of the atoms each have a large number of subcategories.

- *Graph size*
  A graph structure with up to 1,000 categories can be shown with CristalView in around 2 minutes on a high-end desktop computer (Intel Pentium 4 2.8 with 512MB RAM). Due to the algorithmic complexity of the force model (see Section 6.10), the calculation of category positions tend to take too much time when trying to visualize more categories. Moreover, the drawing of the categories on the screen becomes more time consuming with larger classification structures, which we were not able to speed up. Due to the

many interaction possibilities, a refresh of the image on the screen needs to occur very often, which requires all the categories to be redrawn.

- *Hierarchical DAG structures*
  The visualization method works with hierarchical DAG structures. Other graph structures usually do not have an hierarchical structure and, thus, the system of rectangles including each other to denote part-whole relations does not provide a useful image. However, since a tree structure can be regarded as a special case of a hierarchical DAG, namely when each node has at most one supernode, tree structures can be visualized quite well, resulting in an image with only inclusive rectangles and no overlapping rectangles.

- *Arbitrary size of rectangles*
  Since the size of a rectangular category is determined by the number and positions of its subcategories, the size of the rectangles does not reflect an actual measure used in the classification structure. For example, when a classification is made on departments and subdepartments of an enterprise, usually the largest department is the production department and the smallest departments are the support and facilitating departments. The large production department could be denoted by a small atom, because there is no further subdivision made, but the support and facilitating departments usually contain more subdepartments, resulting in larger rectangles with subrectangles and subsequently atoms in them. This could be counter-intuitive for the users, because CristalView only shows the structure of the classification, not the contents of the classification.

8.3 Further research

- *Avoid unwanted overlap*
  We have tried to avoid unwanted overlap, but as stated previously, we could not prevent it completely. We have, however, a few recommendations for future research on this topic.

  1. *Placement of atoms*
     Since the atoms at the lowest level only have supercategories, it should be possible to identify groups of atoms that have the same supercategories. This can be one supercategory, or a set of supercategories.

These atoms can subsequently be placed using a linear placement algorithm that places all these atoms in a square. Using this method, there is no need for springs at the lowest level of the atoms, which would greatly reduce calculation complexity, since in practice, that is the widest part of the graph with the largest amount of springs. Furthermore, since the groups of atoms will be connected with springs to other groups of atoms, instead of the atoms themselves, the spring system will allow for more flexibility, because there is less tension on the springs. This should result in a better layout with less unwanted overlap.

2. *Longer springs in larger fully connect graphs of springs*
   Since the springs are defined between the sibling categories of the graph, i.e., the categories that share at least one supercategory, this results in a fully connected graph of springs between the siblings under one supercategory. When there are many siblings in this fully connect graph, the result after a number of iterations of the force algorithm will be that the sibling categories will remain overlapping each other in a circular shape. It should be possible to increase the lengths of the springs when the number of siblings connected by such a fully connected graph of springs is large to avoid this sort of unwanted overlap.

- *Extend to general graphs*
  As stated previously, the visualization only works with hierarchical DAG structures. To allow for a general graph, this graph has to be transformed into a hierarchical DAG, according to the following. To transform an undirected graph, a startnode has to be chosen and a ranking has to be applied. Each node receives a rank, for instance the number of edges to the startnode. Subsequently the undirected edges are transformed into directed edges, while keeping the path from startnode to node equal to the rank of the node. The resulting graph is a hierarchical DAG.

- *Editing of structure*
  Currently, only editing of the attributes of the objects in the graph is allowed. This could be extended by allowing the editing of the structure of the graph. This means that it should be possible to create and remove categories, as well adding supercategory-subcategory relations. Springs have to be added and removed on the fly while editing the structure, which would require a more flexible force model.

- *Hardware acceleration*
  We have experimented a little bit with using hardware accelerated drawing of rectangles to improve performance. Currently, we use the Microsoft Windows GDI+ system to draw the rectangles on the screen. Our system relies on transparency to denote overlap, and the GDI+ system uses, for example, software transparency calculations. We have implemented OpenGL hardware accelerated drawing, but unfortunately there was no noticeable difference in drawing performance. The reason for this could be the fact that the coordinate system with its zooming matrices was originally based on the GDI+ system and was used again in the OpenGL implementation. A speed-up can probably be expected when these zooming operations are performed using OpenGL as well, since OpenGL has built-in functionality for that.

Bibliography

[1] Asahi, T., Turo, D., Shneiderman, B. (1995), *Using treemaps to visualize the Analytic Hierarchy Process*, Information Systems Research, 6 (4), pp. 357-375.

[2] Bartram, L. (1997), *Perceptual and interpretative properties of motion for information visualization*, Proc. of NPIV '97, New Paradigms in Information Visualization and Manipulation, ACM Press, 3-7.

[3] Bartram, L. (1998), *Enhancing Visualizations with Motion*, Proc. of IEEE InfoVis '98: Late Breaking Hot Topics, IEEE Computer Society Press, pp. 13-16.

[4] Bartram, L., Uhl, A., Calvert, T. (2000), *Navigating Complex Information with the ZTree*, Proc. of Graphics Interface 2000, Morgan Kaufmann, pp. 11-18.

[5] Battista, G., Eades, P., Tamassia, R., Tollis, I.G. (1999), *Graph drawing: Algorithms for the visualization of graphs*, Upper Saddle River: Prentice Hall.

[6] Bederson, B. B. (2001), *PhotoMesa: A Zoomable Image Browser using Quantum Treemaps and Bubblemaps*, Proc. of UIST 2001, ACM Press, pp. 71-80.

[7] Bederson, B. B., Boltman, A. (1999), *Does Animation Help Users Build Mental Maps of Spatial Information?*, Proc. of InfoVis '99, IEEE Computer Society Press, pp. 28-35.

[8] Bederson, B. B., Hollan, J. D. (1994), *Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics*, Proc. of UIST '94, ACM Press, pp. 17-26.

[9] van Bracht, E. (2001), *Cristal, a Model for the Description of Statistics*, Seminar on the Exchange of Technology and Know-how and New Techniques and Technologies for Statistics (ETK-NTTS), Crete June 2001 .

[10] van Bracht, E. (2004), *Cristal, a Model for Data and Metadata*, Joint UN-ECE/Eurostat/OECD work session on statistical metadata (METIS), Geneva February 2004 .

[11] Bruls, M., Huizing, K., van Wijk, J. J. (2000), *Squarified Treemaps*, Proc. of VisSym '00, SpringerWienNewYork, pp. 33-42.

[12] Card, S. K., Robertson, G. G., Mackinlay, J. D. (1991), *The Information Visualizer, an Information Workspace*, Proc. of ACM CHI '91: Human Factors in Computing Systems, ACM Press, pp. 181-188.

[13] Carrière, J., Kazman, R. (1995), *Interacting with Huge Hierarchies: Beyond Cone Trees*, Proc. of IEEE InfoVis '95, Computer Society Press, pp. 74-81.

[14] De Chiara, R., Erra, U., Scarano, V. (2003), *VennFS: A Venn-Diagram File Manager*, Proc. of the Seventh International Conference on Information Visualization (IV '03), IEEE Computer Society, pp. 120-125.

[15] Davidson, R., Harel, D. (1996), *Drawing graphs nicely using simulated annealing*, ACM Transactions on Graphics, 15 (4), pp. 301-331.

[16] Eades, P. (1984), *A heuristic for graph drawing*, Congressus Nutnerantiunt, 42, pp. 149160.

[17] Eick, S. G. (1994), *Data Visualization Sliders*, Proc. of UIST '94, ACM Press, pp. 119-120.

[18] Engdahl, B., Köksal, M., Marsden, G. (2005), *Using Treemaps to Visualize Threaded Discussion Forums on PDAs*, CHI '05 extended abstracts on Human factors in computing systems, pp. 1355-1358.

[19] Feiner, S., Beshers, C. (1990), *Worlds within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds*, Proc. of ACM UIST '90, ACM Press, pp. 76-83.

[20] Fruchterman, T. M. J., Reingold, E. M. (1991), *Graph Drawing by Force-directed Placement*, Software - Practice and Experience, 21 (11), pp. 1129-1164.

[21] Furnas, G. W. (1986), *Generalized Fisheye Views*, Proc. of ACM CHI '86, ACM Press, pp. 16-23.

[22] Gansner, E. R., North, S. C. (1998), *Improved Force-directed Layouts*, Proc. of Graph Drawing '98, Springer-Verlag, pp. 364-373.

[23] Gershon, N., Card, S. K., Eick, S. G. (1998), *Information Visualization Tutorial*, Proc. of ACM CHI '98: Human Factors in Computing Systems, ACM Press, pp. 109-110.

[24] Graham, M., Kennedy, J.B., Hand, C. (2000), *A comparison of set-based and graph-based visualisations of overlapping classification hierarchies*, Proc. of the working conference on Advanced visual interfaces, Palermo Italy, ACM Press, pp. 41-50.

[25] *Graphviz - Graph Visualization Software*, `http://www.graphviz.org`.

[26] Graves, A., Hand, C., Hugill, A. (1997), *Interactive Visualisation of Musical Form using VRML*, Proc. of Fourth UK VR-SIG Conference, pp. 98-109.

[27] van Ham, F., van Wijk, J.J. (2004), *Interactive visualization of small world graphs*, IEEE Symposium on Information Visualization (InfoVis '04), pp. 199-206.

[28] Hao, M. C., Hsu, M., Dayal, U., Krug, A. (2000), *Navigating Large Hierarchical Space Using Invisible Links*, Proc. of Visual Data Exploration and Analysis VII, SPIE Press, pp. 73-83.

[29] Harel, D., Koren, Y. (2000), *A Fast Multi-Scale Method for Drawing Large Graphs*, Proc. of Graph Drawing 2000, Springer Verlag, pp. 183-196.

[30] Healey, C. G., Booth, K. S., Enns, J. T. (1995), *Visualizing real-time multivariate data using preattentive processing*, ACM Transactions on Modelling and Computer Simulation, 5 (3), pp. 190-221.

[31] Healey, C. G., Booth, K. S., Enns, J. T. (1996), *High-speed Visual Estimation using Preattentive processing*, ACM Transactions on Human-Computer Interaction, 3 (2), pp. 107-135.

[32] Hendley, R. J., Drew, N. S., Wood, A. M., Beale, R. (1995), *Narcissus: Visualising Information*, Proc. of IEEE Information Visualisation Symposium 95, Computer Society Press, pp. 90-97.

[33] Herman, I., Melançon, G., Marshall, M. S. (2000), *Graph Visualisation and Navigation in Information Visualisation: a Survey*, IEEE Transactions on Visualization and Computer Graphics, 6 (1), pp. 24-43.

[34] Hubona, G. S., Shirah, G. W., Fout, D. G. (1997), *The effects of motion and stereopsis on three-dimensional visualization*, International Journal of Human-Computer Studies, 47 (5), pp. 609-627.

[35] Inselberg, A., Dimsdale, B. (1990), *Parallel Coordinates: A Tool for Visualizing Multidimensional Geometry*, Proc. of IEEE Visualization 1990, IEEE Computer Society Press, pp. 361-378.

[36] Jeong, C.-S., Pang, A. (1998), *Reconfigurable Disc Trees for Visualizing Large Hierarchical Information Space*, Proc. of IEEE InfoVis '98, Computer Society Press, pp. 19-25.

[37] Johnson, B., Shneiderman, B. (1991), *Treemaps: A Space-Filling approach to the visualization of hierarchical information structures*, Proc. of IEEE Visualization '91, IEEE Computer Society Press, pp. 284-291.

[38] Kumar, A., Fowler, R. H. (1994), *A Spring Modeling Algorithm to Position Nodes of an Undirected Graph in Three Dimensions*, University of Texas - Pan American, Technical CS-94-7.

[39] Levkowitz, H., Herman, G. T. (1992), *Color Scales for Image Data*, IEEE Computer Graphics & Applications, 12 (1), pp. 72-80.

[40] MacDonald, L. W. (1999), *Using Color Effectively in Computer Graphics*, IEEE Computer Graphics & Applications, 19 (4), pp. 20-35.

[41] Mackinlay, J. D. (2000), *Opportunities for Information Visualization*, IEEE Computer Graphics & Applications, 20 (1), pp. 22-23.

[42] Munzner, T. (1998), *Exploring Large Graphs in 3D Hyperbolic Space*, IEEE Computer Graphics & Applications, 18 (4), pp. 18-23.

[43] North, C., Shneiderman, B. (1997), *A Taxonomy of Multiple Window Coordinations*, Technical Report CS-TR-3854, University of Maryland, December 1997.

[44] Robertson, G. G., Mackinlay, J. D., Card, S. K. (1991), *Cone Trees: Animated 3D Visualizations of Hierarchical Information*, Proc. of CHI '91: Human Factors in Computing Systems, ACM Press, pp. 189-194.

[45] Shneiderman, B., Wattenberg, M. (2001), *Ordered Treemap Layouts*, Proc. of InfoVis 2001, Human-Computer Interaction Lab, University of Maryland, pp. 73-78.

[46] Siirtola, H. (2000), *Direct Manipulation of Parallel Coordinates*, Proc. of IEEE InfoVis '00, IEEE Computer Society Press, pp. 373-378. (Accompanying "Parallel Coordinate Explorer", Java 1.1 Applet available at: `http://www.cs.uta.fi/~hs/pce/`).

[47] Simons, P. (2000), *Parts: A Study in Ontology*, Oxford University Press, 2000, ISBN: 0199241465.

[48] Tversky, O. J., Snibbe, S. S., Zeleznik, R. (1993), *Cone Trees in the UGA Graphics System: Suggestions for a More Robust Visualization Tool*, Technical Report CS-93-07, Brown University, February 1993.

[49] Tweedie, L., Spence, R., Dawkes, H., Hua, S. (1996), *Externalising abstract mathematical models*, Proc. of ACM CHI '96, ACM Press, pp. 406-412.

[50] Young, F.W. (1985), *Multidimensional scaling*, Kotz-Johnson (ed.) Encyclopedia of Statistical Sciences, Volume 5, Lindberg Conditions to Multitrait-Multimethod Matrices, John Wiley & Sons, Inc. HTML-version available at: `http://forrest.psych.unc.edu/teaching/p208a/mds/mds.html`.

[51] Ware, C. (2000), *Information Visualization: Perception for Design*, Morgan Kaufmann Publishers; 1st edition (January 1, 2000), ISBN: 1558605118.

[52] Ware, C., Franck, G. (1996), *Evaluating Stereo and Motion Cues for Visualizing Information Nets in Three Dimensions*, ACM Transactions on Graphics, 15 (2), pp. 121-140.

[53] van Wijk, J. J., van de Wetering, H. (1999), *Cushion Treemaps: Visualization of Hierarchical Information*, Proc. of IEEE InfoVis '99, IEEE Computer Society Press, pp. 73-78.

[54] Wood, A., Drew, N., Beale, R., Hendley, B. (1995), *HyperSpace: Web Browing with Visualization*, Proc. of Third International World-Wide Web Conference, pp. 21-25.

Appendix A

Our force directed algorithm

Pseudo-code for the force directed algorithm proposed in Chapter 6 is shown in Figures A.1 through A.4.

```
1: const A { Strength of attraction }
2: const R { Radius of atom }
3: const C { Rate at which to diminish Δt }

4: procedure LAYOUT(Graph G)
5:     { Add ghost categories to the graph }
6:     G' ← DEFINEGHOSTS(G)
7:     Place the nodes of G' at random positions
8:     { Define springs between the categories }
9:     S ← DEFINESPRINGS(G')
10:    repeat
11:        for all v ∈ V(G) do
12:            F_v ← 0
13:            for all (v, u) ∈ springs(v) do
14:                { Calculate force vector resulting from spring }
15:                F_v ← F_v + A_uv · (|p_u − p_v| − x_0^uv)
16:            end for
17:        end for
18:        { Move each node v according to its force vector F_v }
19:        for all v ∈ V(G) do
20:            p'_v ← p_v + F_v · Δt
21:        end for
22:        { Change each spring's zero energy length }
23:        for all (u, v) ∈ S do
24:            x_0^uv ← CHANGESPRINGLENGTH((u, v))
25:        end for
26:        { Diminish time step }
27:        Δt' ← C · Δt
28:    until Δt below threshold
29: end procedure
```

Figure A.1: *The force directed algorithm proposed in Chapter 6.*

30: **function** DEFINEGHOSTS($Graph\ G$)
31: { Iterate over all pairs of categories }
32: **for all** $u \in V(G)$ **do**
33: **for all** $v \in V(G)$ **do**
34: **if** $sub(u) \subseteq sub(v)$ **then**
35: { u is contained in v, so add ghost to u }
36: $sub(u) \leftarrow (sub)u \cup ghost$
37: **end if**
38: **end for**
39: **end for**
40: **return** G
41: **end function**

Figure A.2: *The force directed algorithm proposed in Chapter 6 continued.*

42: **function** DEFINESPRINGS($Graph\ G$)
43: { Define springs between siblings }
44: $S \leftarrow \{(u,v)|u,v \in V(G) \wedge super(u) \cap super(v) \neq \emptyset\}$
45: **for all** $(u,v) \in S$ **do**
46: { Set spring stiffness to resemble multiple springs }
47: $A_{uv} \leftarrow \{\#s|s \in super(v) \cap super(u)\} \cdot A$
48: **if** $\#super(u) = 1 \wedge \#super(v) > 1$ **then**
49: { u is lonely category, so increase spring stiffness }
50: $A_{uv} \leftarrow 2 \cdot A_{uv}$
51: **end if**
52: { Set initial springlength }
53: $x_0^{uv} \leftarrow R \cdot (\sqrt{w_u} + \sqrt{w_v}) + m$
54: **end for**
55: **return** S
56: **end function**

Figure A.3: *The force directed algorithm proposed in Chapter 6 continued.*

```
57: function CHANGESPRINGLENGTH(Spring (u, v))
58:     if sub(u) ∩ sub(v) = ø then
59:         { u and v share no subcategories }
60:         return r_u + r_v + m
61:     else
62:         { u and v share subcategories }
63:         return r_u + r_v − 2 · r_o − m
64:     end if
65: end function
```

Figure A.4: *The force directed algorithm proposed in Chapter 6 continued.*

Appendix B

Variable Editor

Currently, there exist two ways to create and/or modify a classification structure in a Cristal. One way is to program an application that communicates with the Cristal library (.dll) to load a Cristal in memory, modify it programmatically and save it to disk. The other way is to use the Variable Editor, which is an editing tool, especially designed to create and/or modify Cristals. In this appendix we give an overview of the Variable Editor.

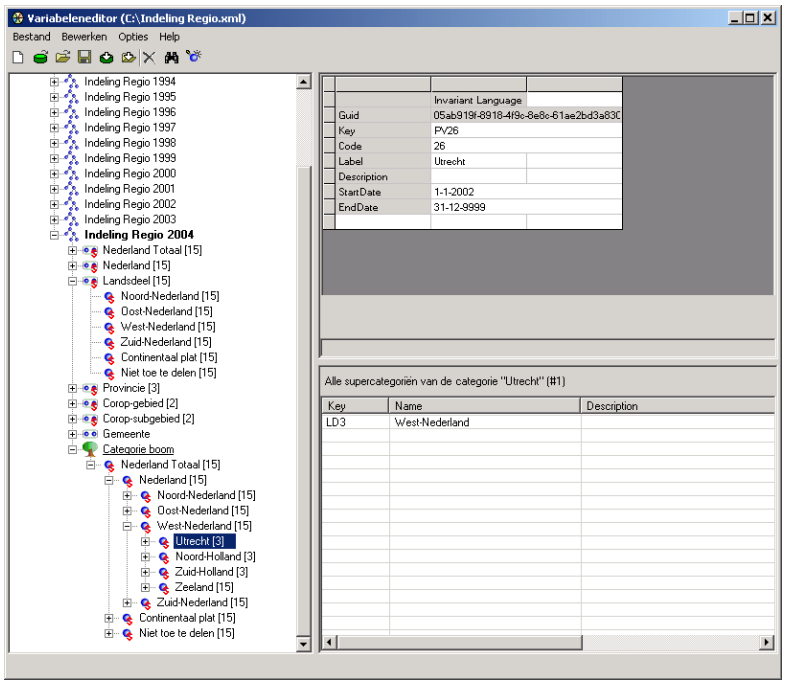
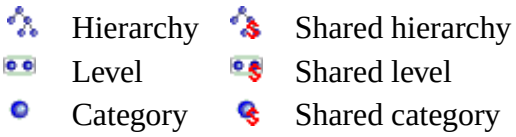


Figure B.1: *Screenshot of the Variable Editor.*

A screenshot of the Variable Editor is shown in Figure B.1. We continue by giving an explanation of each part of the program.

Toolbar
The top of the window displays the toolbar with easy-access buttons. Besides the standard actions such as loading, saving and searching, the toolbar supports creating new hierarchies, levels and categories depending on the selection in the listview on the left.

Hierarchy-tree
The left of the window displays the hierarchy-tree. Each object in the tree has an icon associated with it, which can be one of the following:



The tree displays a list of hierarchies. When a hierarchy is expanded, a list of levels is displayed, followed by an icon of a tree. A list of levels can be expanded to show the categories in that level, while below the tree-icon the tree of categories is displayed.

As explained in Chapter 2, a hierarchy contains a sequence of levels and a tree of categories, but multiple hierarchies can share levels and categories which results in complicated graph structures. The hierarchy-tree in the Variable Editor, therefore, only shows the hierarchies, but denotes a shared level or shared category with an additional ’s’ in the icon. A shared level is, thus, a level that resides in multiple hierarchies and a shared category is a category that resides in multiple levels (which implies it resides in multiple hierarchies). The Variable Editor, therefore, displays multiple instances of the same objects in the hierarchy-tree when they are shared.

The user can add or remove objects using the right-mouse button. Adding an object pop-ups a new window in which the user can enter the values of the attributes for the new object. Also, the user can select an object that is already present from the list, to share this object in multiple hierarchies. Removing an object is also done using the right-mouse button.

Attributes
The top-right of the window displays the attributes of the selected object. The attributes are listed vertically while the different languages are listed

horizontally. Each object has a number of standard attributes that are displayed using a gray box in the attribute-window. The user can add or remove attributes from objects in this window.

Category-list
The bottom-right of the window displays a list of categories. When a hierarchy is selected on the left, this is a complete listing of all the categories present in the selected hierarchy. When a level is selected, this is a listing of the levels in the selected hierarchy. When a category is selected, this is a listing of the supercategories of the selected category.

Appendix C

User experiment test set

This appendix lists the test sets used in the user experiment explained in Chapter 7.

C.1 Part one - Comparison

During the first part of the user experiment, the users got to see a particular classification structure of a Cristal. The graph structure of version A of the classification is shown in Figure C.1 and Figure C.2 shows version B. Note that the participants never saw these graph structures. As can be seen in the figures, we used two classifications with exactly the same structure, but with different subjects. In fact, a number of category-names were shuffled.

The visualization of version A of the test classification in CristalView is shown in Figure C.3 and Figure C.4 shows version B. Of course, the participants were free to navigate, zoom, click and drag the information set.

We asked the same type of questions for the two test sets (A and B), but the questions were regarding different subjects, since the subjects were mixed. The questions the participants had to answer are listed below. Note that the user experiment was in Dutch, so the questions were in Dutch also:

Test set A

Category visualization:

1. Zoek de categorie ”Marianne” op. Tot welke supercategorie behoort ”Marianne”?

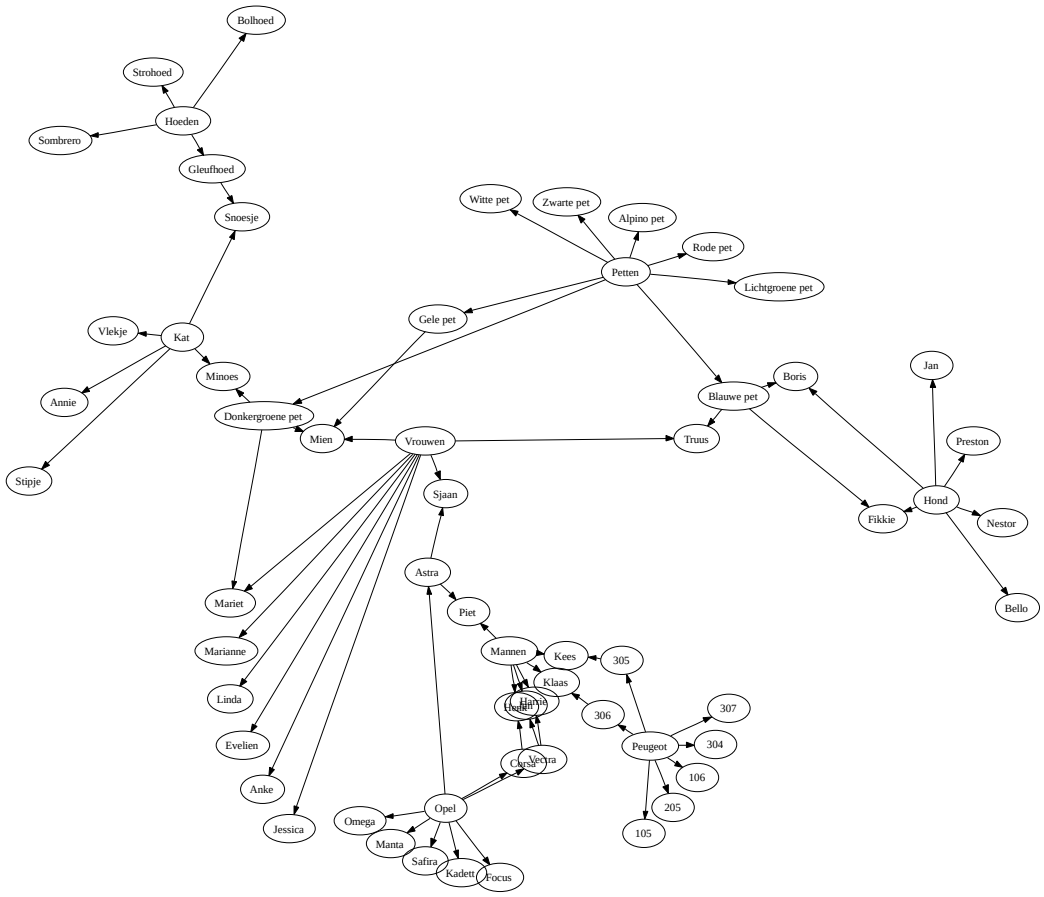


Figure C.1: *Graph structure of version A of the test set used in part one of the user experiment. (Created with the twopi layout algorithm in Graphviz [25])*

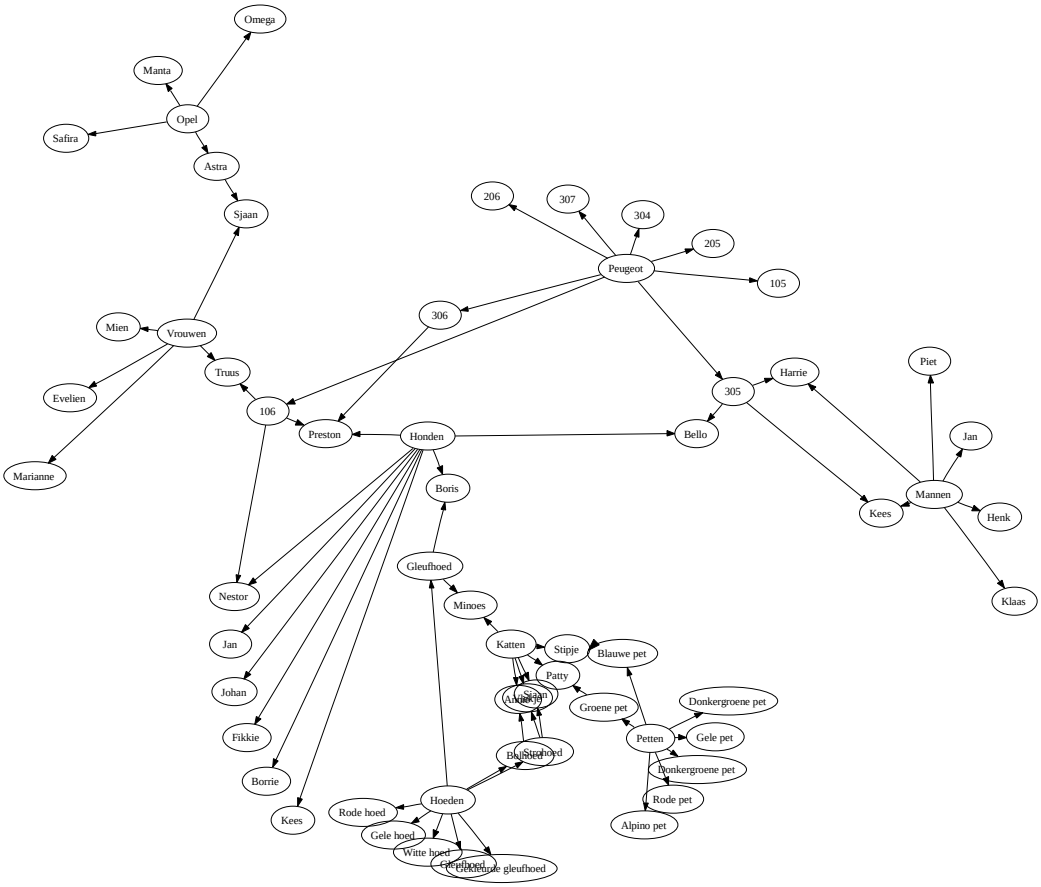


Figure C.2: *Graph structure of version B of the test set used in part one of the user experiment. (Created with the twopi layout algorithm in Graphviz [25])*

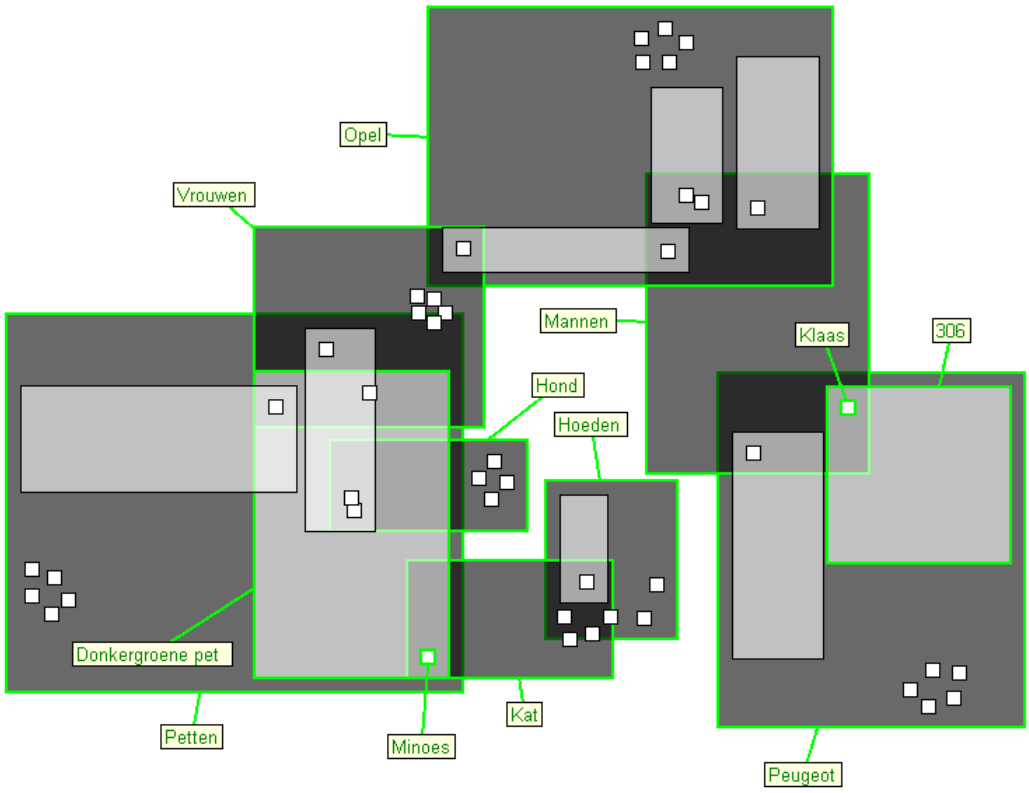


Figure C.3: *Visualization of version A of the test set used in part one of the user experiment.*

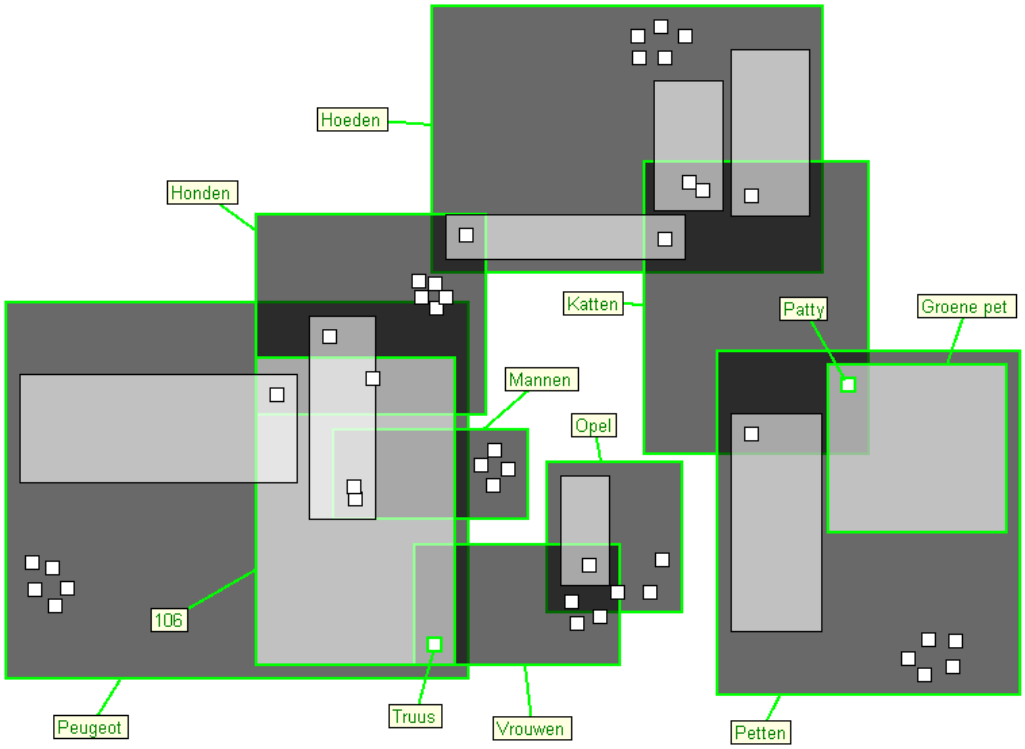


Figure C.4: *Visualization of version B of the test set used in part one of the user experiment.*

2. Hoeveel vrouwen bevat de categorie ”Vrouwen”?
3. Zoek ”Truus” op. Wat voor pet heeft zij op haar hoofd?
4. Wie heeft/hebben er naast ”Truus” nog meer een blauwe pet op het hoofd?
5. Zoek de categorie ”Opel” op. Wie hebben er een Opel ”Vectra”?
6. Zijn er nog meer mensen naast Jan en Harrie die een Opel Vectra hebben? Zo ja, wie?
7. Hoeveel Opels staan er in dit Cristal?
8. Hoeveel Mannen hebben er een Peugeot?
9. Hoeveel Mannen zijn er in totaal?
10. Gemengde vragen: Heeft Mien twee petten tegelijk op?
11. Hoeveel Hoeden staan er in dit Cristal?
12. Wie hebben er allemaal een Donkergroene pet op?

Level visualization:

13. Welke soorten Huisdieren zijn er?
14. Welke hoofdcategorien Hoofddeskels zijn er in 2004?
15. Hoeveel Huisdieren hebben een Hoofddeksel op hun Kop?
16. Hoeveel Personen hadden een Auto in hun garage staan in 2004?
17. Welke Auto’s bestonden alleen in 2005?

Time visualization:

18. Zoek de Peugeot ”304” op. Heeft de ”304” een opvolger? Zo ja, welke?
19. Wat zijn alle opvolgers van de ”305”?
20. Wat is de voorganger van de ”Focus”?
21. Wat is er in de nieuwjaarsnacht van 2004 op 2005 met de ”Lichtgroene pet” gebeurd?
22. Welke pet had Mien het eerst op?

Test set B

Category visualization:

1. Zoek de categorie ”Fikkie” op. Tot welke supercategorie behoort ”Fikkie”?
2. Hoeveel honden bevat de categorie ”Honden”?
3. Zoek ”Bello” op. In wat voor Auto zit ”Bello”?
4. Wie zitten er naast ”Bello” nog meer in een 305?
5. Zoek de categorie ”Hoeden” op. Wie dragen er een ”Strohoed”?
6. Zijn er naast Vlekje en Sjaan nog meer huisdieren die een Strohoed dragen? Zo ja, welke?
7. Hoeveel Hoeden staan er in dit Cristal?
8. Hoeveel Katten hebben er een Pet?
9. Hoeveel Katten zijn er in totaal?
10. Gemengde vragen: Zit ”Preston” in twee Auto’s tegelijk te spelen?
11. Hoeveel Opels staan er in dit Cristal?
12. Wie zitten er allemaal in een ”106”?

Level visualization:

13. Welke soorten Geslacht zijn er?
14. Welke merken Auto’s zijn er in 2004?
15. Hoeveel Personen hebben een Auto in hun Garage?
16. Hoeveel Huisdieren hebben een Hoofddeksel op hun kop in 2004?
17. Welke Hoofddeksels bestonden alleen in 2005?

Time visualization:

18. Zoek de ”Gele pet” op. Heeft de ”Gele pet” een opvolger? Zo ja, welke?
19. Wat zijn alle opvolgers van de ”Blauwe pet”?
20. Wat is de voorganger van de ”Gekleurde gleufhoed”?
21. Wat is er in de nieuwjaarsnacht van 2004 op 2005 met de Peugeot ”105” gebeurd?
22. In welke Auto zat ”Preston” het eerst?

C.2 Part two - Deficiencies

Figure C.5 shows the visualization of a classification of the Netherlands we used during the second part of the user experiment. The participants were asked to write down all strange things they could see and find in the loaded classification structure. Subsequently they were asked to write down the changes they would have made when they would have created the classification structure themselves.

C.3 Part three - Survey

The third part of the user experiment consisted of more open and multiple choice questions. English translations of the original Dutch questions are added. A *Q* denotes a question, and an *A* denotes an answer, for both question-sets.

The original questions:

- Q: Wat vond u van het programma? Denkt u dat u het gaat gebruiken wanneer u inzicht wil krijgen in een bepaald Cristal?
- A: .....
- Q: Wat vindt u van het navigeren door de dataset? (omcirkel keuze)
- A: goed / voldoende / matig / onvoldoende / slecht
- Q: Vindt u het programma gemakkelijk te gebruiken? (omcirkel keuze)
- A: zeer gemakkelijk / gemakkelijk / neutraal / ongemakkelijk / zeer ongemakkelijk
- Q: Vindt u dat het programma voldoende functionaliteit bevat? (omcirkel keuze)
- A: ruimschoots voldoende / voldoende / neutraal / onvoldoende / zeer onvoldoende
- Q: Heeft u het idee dat u door het programma een beter begrip heeft van de mogelijkheden van het Cristal model (voor zover dit eerst niet het geval was)?
- A: veel beter / beter / hetzelfde / minder / veel minder
- Q: Wat had u graag verbeterd/veranderd/toegevoegd willen zien aan het programma?
- A: .....

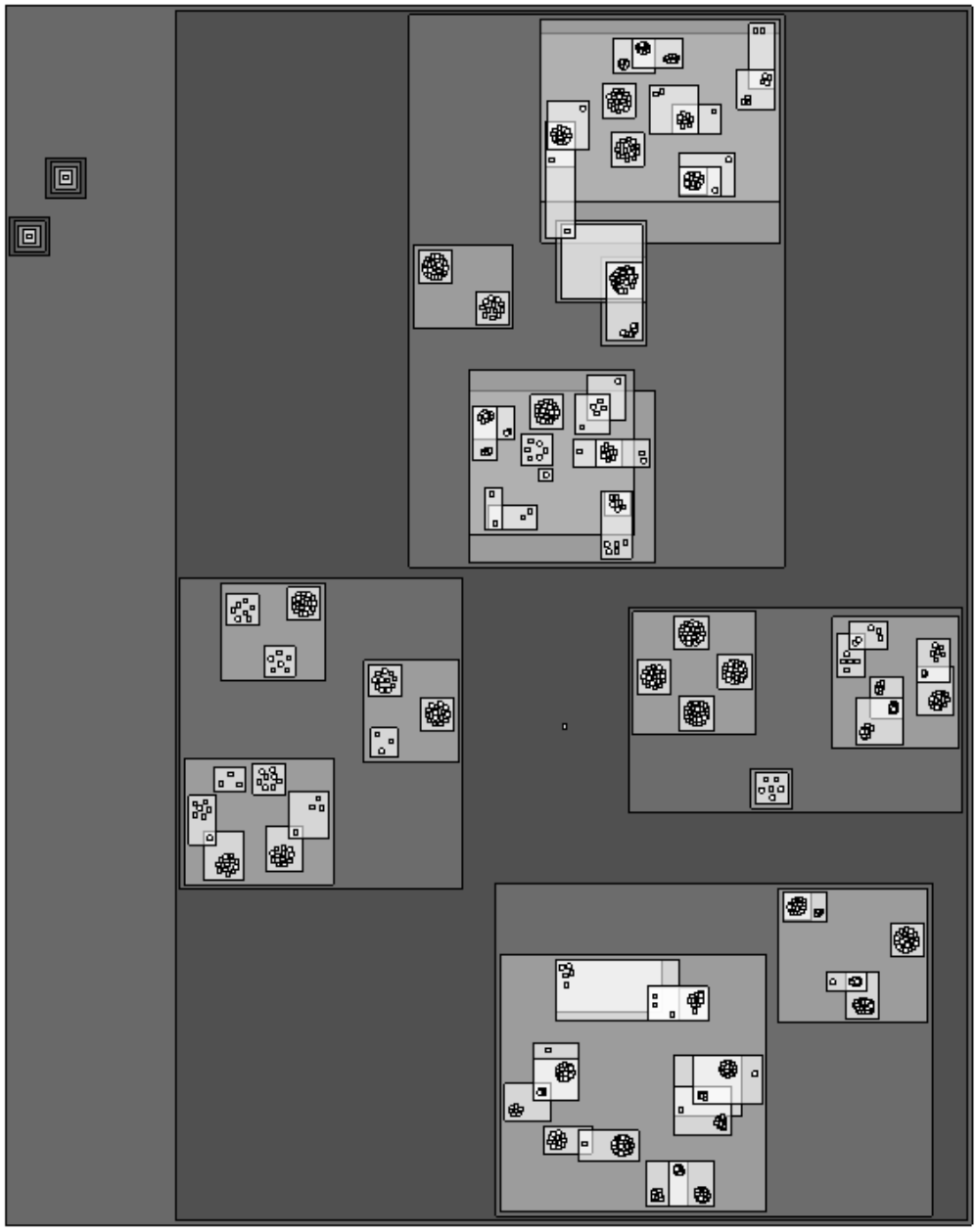


Figure C.5: *Visualization of classification of the Netherlands used in part two of the user experiment.*

- Q: Extra opmerkingen kunt u hieronder kwijt.
- A: .....

English translation of the questions:

- Q: What did you think of the program? Do you think you will use the program in case you want to get insight in a certain Cristal?
- A: .....
- Q: What is your opinion on navigating through the dataset? (circle choice)
- A: good / sufficient / average / insufficient / bad
- Q: Do you think the program is easy to use? (circle choice)
- A: very easy / easy / neutral / not easy / very uneasy
- Q: Do you think that the program contains sufficient functionality? (circle choice)
- A: very much sufficient / sufficient / neutral / insufficient / very insufficient
- Q: Do you think you have gained a better understanding of the possibilities of the Cristal model because of the program (when this wasn't already the case)?
- A: much better / better / the same / less / much less
- Q: What do you like to see improved/changed/added to the program?
- A: .....
- Q: Below there is room for additional remarks.
- A: .....