Classification and Visualization of Contents using Real Valued Attributes

ICS 227: Advanced User Interface Architectures
PROJECT REPORT
Spring 2004

Sushil K Bajracharya, Orion Auld

{sbajrach,oauld}@ics.uci.edu

Department of Informatics
School Of Information and Computer Science
University of California, Irvine

Abstract

This report describes the background, motivations, usage (operation) and architecture of *nD Visualizer*, the application we designed and implemented during Spring 2004 as our *ICS 227: Advanced User Interface Architectures* course project. We also discuss enhancements we intend to make to the application in the future.

1 INTRODUCTION

Classification and *filtering* are two basic operations all computer users deal with regularly. A simple search for an important file on the hard disk, browsing an online music catalog, performing complex database queries in an ERP data warehouse or understanding patterns of information in huge scientific databases - all these forms of information seeking encompass some form of filtering operations. On the other hand the structures that are used to represent information are themselves the embodiment of some classification scheme, such as the hierarchy of files inside folders, the navigational structure of a web site, the schema for a database, design of multidimensional data cubes in data warehouses and so on. A tremendous amount of computing infrastructure and innovation lies behind all those systems that support the kinds of operations we just listed, starting from the structures for information representation, storage and retrieval techniques, visualization of results and modalities to interact with such systems. For different problems there are different solutions and often times we get used to a particular solution strategy that has dominated a particular domain. In this paper we will describe an application we built, *nD Visualizer* that is our attempt to experiment with an alternative strategy for *filtering and classifying content*.

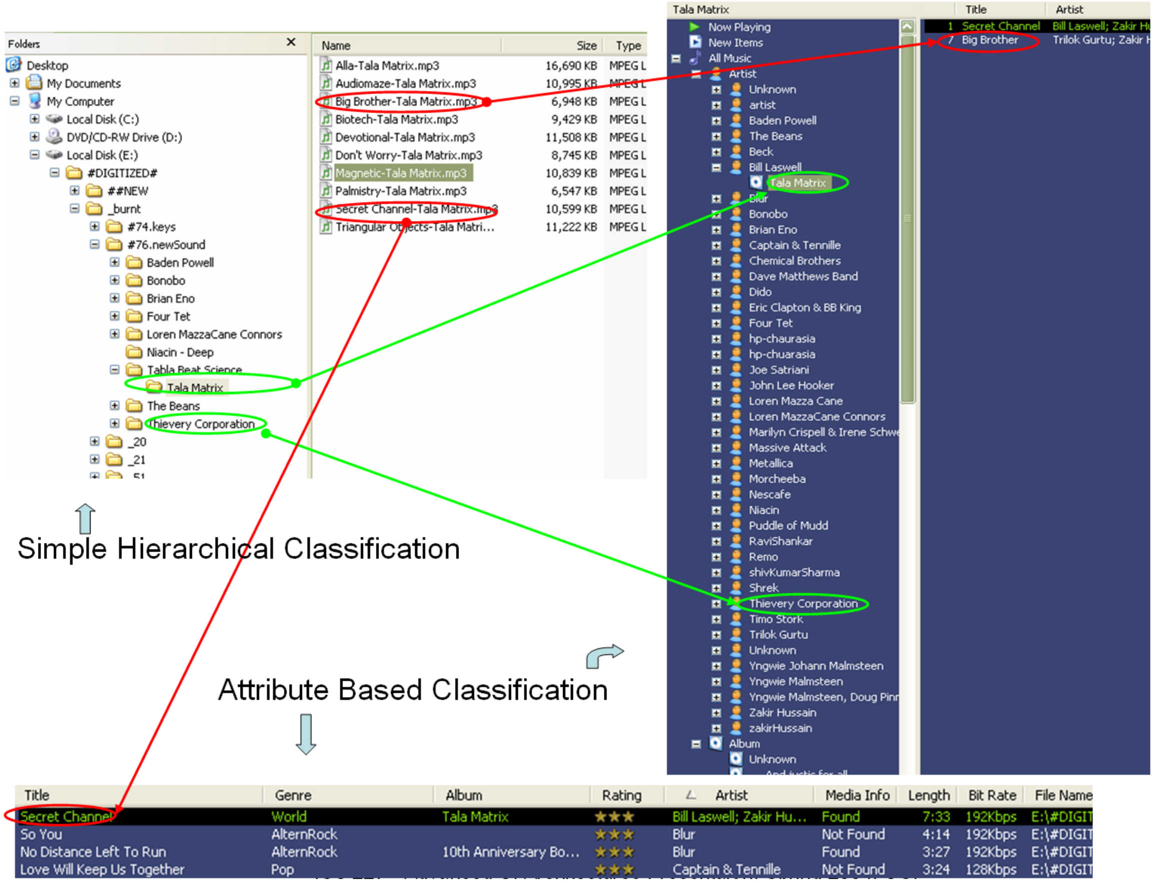
By *content*, we mean any binary representation of usable information. For our purpose a content is equivalent to a file in a hard disk. To start out simply we assume there already exists an implicit classification or categorization, for example 'songs', 'papers' etc. So this reduces our problem domain into a finite space as we are in a way dealing with content classification in the small, but with different strokes. We also are not looking into issues like versioning and updates of the content under consideration, although these may be issues worth considering in some domains.

2 CONTENT CLASSIFICATION IN SMALL

A universal example of managing content in computers is the hierarchal filesystem. Creating folders and placing files into them is a daily practice. Hierarchical organization is in essence a way of clasifying content, and navigation of those folders is a kind of filtering. Despite its simplicity and ease, this method has some distinct limitations. First, the organization and classification is static. There is only one explicit structure in the organization whose semantics tend to be subjective. Second, there will be only one way to navigate the contents through the pre-defined tree or hierarchy. With a structure like this one ends up with an obvious dilemma when one figures out that a file might fit under two different folders.

2.1 Classification using Meta-Data

Meta-Data or *attributes* are properties that we assign to contents. These attributes can be used in many different ways. One obvious way is to uniquely identify some content; another use might be to group contents based on same values for some attributes. Associating meta-data with content and classifying them according to the values given can be used as an alternative to static folder-style hierarchical classification. Indeed there are many ways to implement this. We can define a schema with relationships between meta-data elements and filter/classify contents by querying on meta-data. This is nothing new, as every database application works that way. While this approach is very useful in some circumstances – hence the ubiquity of the relational database – it is inappropriate for others. One would have second thoughts before replacing



Name	Size	Type
Alla-Tala Matrix.mp3	16,690 KB	MPEG L
Audiomaze-Tala Matrix.mp3	10,995 KB	MPEG L
Big Brother-Tala Matrix.mp3	6,948 KB	MPEG L
Biotech-Tala Matrix.mp3	9,429 KB	MPEG L
Devotional-Tala Matrix.mp3	11,508 KB	MPEG L
Don't Worry-Tala Matrix.mp3	8,745 KB	MPEG L
Magnetic-Tala Matrix.mp3	10,839 KB	MPEG L
Palmistry-Tala Matrix.mp3	6,547 KB	MPEG L
Secret Channel-Tala Matrix.mp3	10,599 KB	MPEG L
Triangular Objects-Tala Matri...	11,222 KB	MPEG L

	Title	Artist
1	Secret Channel	Bill Laswell; Zakir Hu
7	Big Brother	Trilok Gurtu; Zakir H

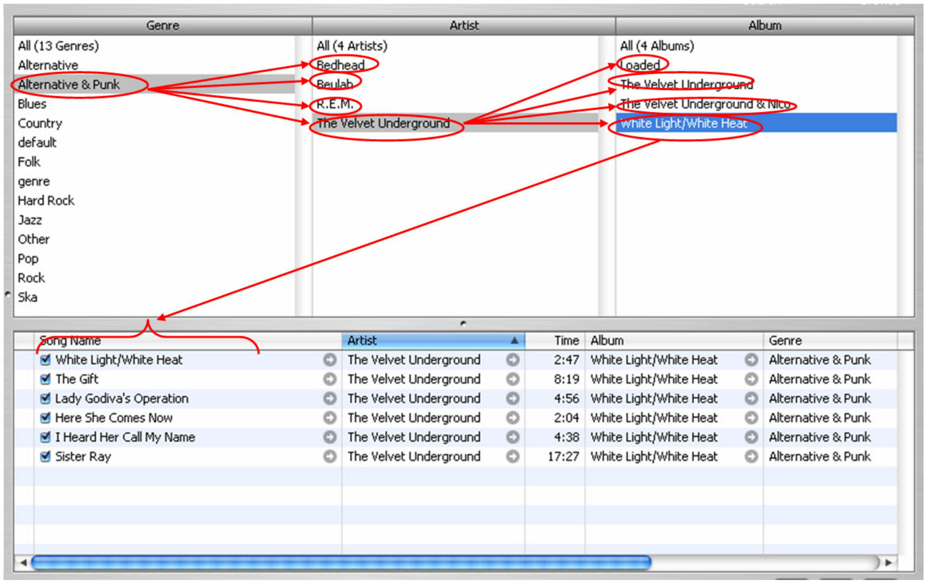
Title	Genre	Album	Rating	Artist	Media Info	Length	Bit Rate	File Name
Secret Channel	World	Tala Matrix	★★★	Bill Laswell; Zakir Hu...	Found	7:33	192Kbps	E:\#DIGIT
So You	AlternRock		★★★	Blur	Not Found	4:14	192Kbps	E:\#DIGIT
No Distance Left To Run	AlternRock	10th Anniversary Bo...	★★★	Blur	Found	3:27	192Kbps	E:\#DIGIT
Love Will Keep Us Together	Pop		★★★	Captain & Tennille	Not Found	3:24	128Kbps	E:\#DIGIT

Figure 1: Differentiating attribute based classification

one's file system with a relational database. Furthermore , there will always be an open problem to create an ontology that describes the relationship between all kinds of attributes we might need tag our contents with.

A simpler, less formal solution is to embed meta-data in the content itself and not explicitly define the schema. If we can facilitate filtering and classification of such contents it is possible that the users can see the relationship between contents without defining all possible relationships among the attribute themselves. To clarify these issues we move to a specific domain and deal with a particular kind of content - music files. Even though we have used music files (mp3) as the illustrative domain for *nD Visualizer* it should be noted that our application is generic enough to work with any type of content and any set of attributes, though it may not be appropriate for all domains.

Before going into the details of content classification we take a look at Figure 1 that differentiates conventional file system like hierarchal classification with attribute based classification. It should be noted that how a content that exists inside a folder can be reached through different hierarchies when the classification is based on meta-



Song Name	Artist	Time	Album	Genre
☑ White Light/White Heat	The Velvet Underground	2:47	White Light/White Heat	Alternative & Punk
☑ The Gift	The Velvet Underground	8:19	White Light/White Heat	Alternative & Punk
☑ Lady Godiva's Operation	The Velvet Underground	4:56	White Light/White Heat	Alternative & Punk
☑ Here She Comes Now	The Velvet Underground	2:04	White Light/White Heat	Alternative & Punk
☑ I Heard Her Call My Name	The Velvet Underground	4:38	White Light/White Heat	Alternative & Punk
☑ Sister Ray	The Velvet Underground	17:27	White Light/White Heat	Alternative & Punk

Figure 2: Classifying contents in a Hierarchical Structure based on Attributes

data. In particular, the image at the bottom of Figure 1 shows how contents (music files) are listed according to meta-data like ’genre’, ’artist’,’rating’ etc. The way these attributes are stacked on top of each other creates a new oraganization and a new filtering criteria. Figure 2 shows more explicitly how one can get a hierarchy using attributes or meta-data. Applications like these (media players) truly demonstrate how,

- Organizing into hierarchies is one way of classifying data
- Navigating a hierarchy is one way of filtering data

2.2 Nominals and Reals - Two categories of attributes

Broadly attributes can be categorized into two - *Nominals* and *Reals*. *Nominal* attributes (like ’genre’, ’artist’, ’albums’ for music files) take values which do not have any defined realtionship among them as such. In order words, we cannot comeup with a standard or consistent metric to arrange these values in a scale. *Real* attributes on the other hand take values that we can order in some way. For example, ’rating’, ’grooving’, ’danceability’, ’contemporary-exotic’ are few of the Real-valued attributes – of varying degrees of objectivity – for the music domain.

3 USING REAL-VALUED ATTRIBUTES FOR CLASSIFICATION

Consider a group of real-valued attributes over a set of mp3 files. Presume these attributes are chosen to represent aspects of the music that are relevant to the sensibility of the collector. They can be quantitative attributes such as Tempo, Rating, and Bitrate, or qualitative attributes such as Familiar-Exotic, Sad-Happy, or Classic-Contemporary.

These compound names indicate that the user means for the property to represent the continuum between two extremes. In the case of FamiliarExotic, a 0 value would indicate that the item being rated has a very conventional sound to the user, where a 100 would indicate that the song sounds exotic.

What is the utility of attaching a large number of subjective numeric attributes to mp3 files? Consider the following scenarios:

- *Scenario 1:* A large media company has a subdivision that specializes in small, anonymous musical samples of the sort that are used every day in radio commercials, promotional videos, and all sorts of throwaway media that impinge upon our lives every day. The company's catalog offers a bewildering array of musics which do a decent job of approximating all styles and genres of music. Too many, in fact. In order to find an upbeat, happy jazz number with a bit of exotic flavor that would be just perfect for one's Suntan Lotion Commercial (whose deadline is Friday), one must browse three hundred songs in the Jazz > Combo subcategory. This is especially irritating when one isn't particularly wedded to using Jazz as a genre; something calypso or reggae or otherwise caribbean would do just as well, if it had the same essential characteristics, but that would mean another two or three hundred songs to browse through.[1]

- *Scenario 2:* Some poor soul is throwing a brunch for a few of his friends. He wants the music to blend into the background, but he also wants the material to be interesting and cohesive should people decide to listen to it more carefully. The material may come from a great many genres, the way he normally organizes his music, but should all have some essential characteristics: midtempo, smooth, and slightly exotic. Unfortunately, he has 5000 mp3s he must wade through in order to find the right songs to put in his playlist. Meanwhile, the hollandaise is separating.

  These scenarios, particularly the second, are the ones that have driven the design of our application.

[1] One of the authors has, in fact, written software for just such a large media company. The author's program was not intended to solve this this particular problem, but he is convinced that, if it had solved this issue, the company the author worked for could have bilked at least another $50,000 out of the large media company, though he is certain they would have passed none of that on to him.

3.1 Belling the cat

One question that springs immediately to mind is this: who will do all of this classification? A user's music collection may range from a few hundreds to tens of thousands of songs, with new ones being integrated all of the time. Even with a small number (say, nine or ten) of properties associated with each song, it is unreasonable to imagine that a user would take the time to rate their entire collection. And what if they wanted to add a property? This would mean revisiting every song they owned.

The possibility exists that this content could be provided by a company on a subscription basis. Several companies are, in fact, in the business of providing metacontent for music and video. But we think that with sufficient interest in the metadata, a groupware solution would suffice.

In this vision, each user would have an interface that would allow them to rate their songs according to any existing properties and to add (private) properties. This software would be in communication with a server which aggregates this information for all users. The client software intermittently updates the ratings of the songs based on the values on the server. The client can display all of the songs in a user's collection that have unrated properties. When the user rates one of these songs, this information is transmitted to the server and propogates to other users.

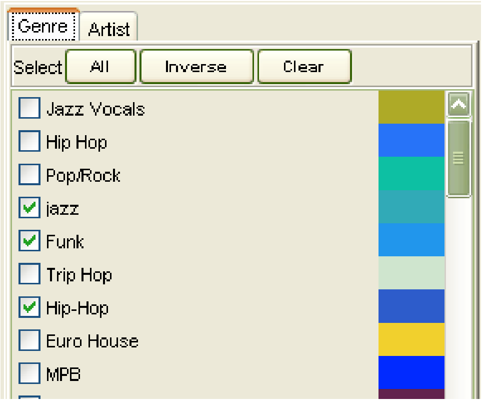
If a user disagrees with the rating that a song has been given, she can adjust the ratings. These new ratings are stored locally, permanently overriding the ratings transmitted from the server. The new ratings are also transmitted to the server, where they are aggregated with the existing ratings. In this way, the ratings converge to a consensus opinion.

Music fans are an opinionated bunch. When a user selects a group of songs according to a certain criteria, and songs are included that do not conform to the user's expectations, they are likely to examine the properties responsible. When they find that they substantially disagree with the ratings, the temptation to protest by expressing their own opinion will be too great to resist for many.

The ontology of properties (i.e., the list of properties used to rank songs) should probably be under control of a few very interested parties. When users create new (local) properties, these should be transmitted to the server as well, with the results taken as advice to those that maintain the ontology.

This kind of oligarchical control would allow deep consideration for decisions that have far-reaching effects, such as adding or removing from the list of properties. [2]

[2] A wise group, for instance, probably would not allow a property such as ”FamiliarExotic”, which was used in an earlier example. After all, what is exotic to one group of people is quite likely to be familiar to another group, based on culture and geography. A well-chosen set of attributes, while still qualitative, would minimizes issues like this.



☐ Jazz Vocals 🟨
☐ Hip Hop 🟦
☐ Pop/Rock 🟩
☑ jazz 🟦
☑ Funk 🟦
☐ Trip Hop ⬜
☑ Hip-Hop 🟦
☐ Euro House 🟨
☐ MPB 🟦

Figure 3: Nominal Selection Panel (Controls)

4 OPERATING THE APPLICATION - *nD Visualizer*

A screenshot of our running applcation is shown in Figure 5. It shows all the elements of the application, controls for filtering and clustering, for setting the visualization options, the three dimensional view of clustered contents and a table or list view showing the list of the contents that have been filtered and then clustered.

4.1 Filtering

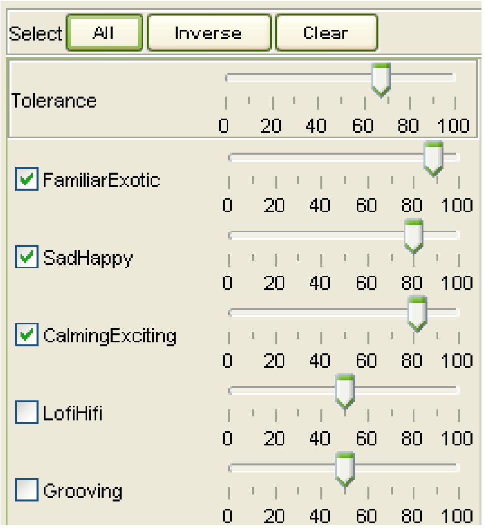
The nominal selection panel (Figure 3) offers the user the opportunity to filter the data that is considered for clustering. A tab is presented for every nominal property that the data has. In Figure 5, the application has been loaded with a datafile that has properties for Genre and Artist. When a particular nominal is selected (in this case, Genre), checkboxes are presented that represent all possible values that the nominal can take. The user may toggle the value for one, or, using the buttons at the top of the pane, select or clear all of them, or toggle the current values for all of the nominal values.

Each of the values for a nominal is associated with a color. This color is used in certain visualization modes to display the nominal values that a selected points or bins belong to. More information on this can be found in the section on visualization.

Filtering is useful for crude selection of a large sphere of data from which the user will make a more fine-grained selection using the real-valued clustering controls.

4.2 Clustering

The real-value selection panel (Figure 4) is the heart of the application. It is here that the main vision of the application is achieved - the ability to specify a set of qualitative criteria and recieve a group of songs that fits that criteria.



☑ FamiliarExotic

☑ SadHappy

☑ CalmingExciting

□ LofiHifi

□ Grooving

Figure 4: Real Values Selection Panel (Controls)

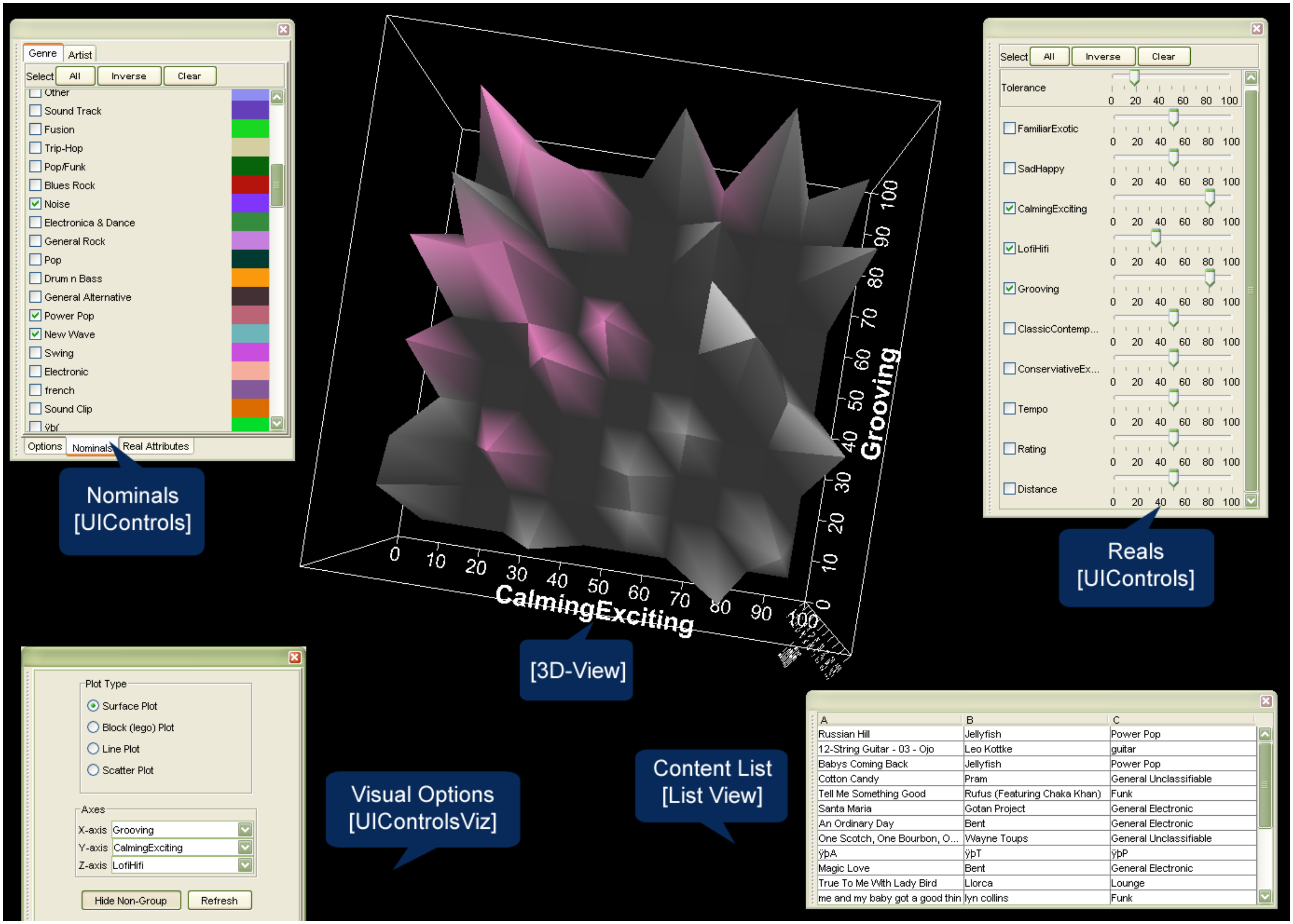
The tolerance slider indicates how large of a pool of items that the user is trying to select in terms of a percentage of the total number of items (though in the current implementation this is interpereted as the literal number of items).

Each real valued property contained in the datafile corresponds to a slider value on the selection panel. In the case of Figure 4, the application has been loaded with a datafile that includes properties named ”LofiHifi”, ”FamiliarExotic”, etc. These compound names indicate that the user means for the property to represent the continuum between two extremes. In the case of ”FamiliarExotic”, a 0 value would indicate that the item being rated has a very conventional sound to the user, where a 100 would indicate that the song sounds exotic. The nature of this semantic association, of course, is completely transparent to the application, which just sees the name and value of a real-valued property.

The checkboxes near the property names indicate whether or not the particular property is actively being used to select a cluster or not. In the case of our example, the user wishes to select songs which are happy, exciting, and very exotic. The tolerance bar indicates that the user wants the 68 (or so) songs that best meet this criteria. The user can use the buttons at the top of the panel to change the selection of real-valued attributes en-masse. The ”all” button causes all real-valued properties to be selected; the ”clear” button removes any clustering based on these properties. ”Invert” toggles which properties are selected.

4.3 Visualization Modes

The view of the data that is determined by the current settings of the (nominal and real) properties is always reflected in the table or list view (Figure 5) and the visualization



□ Other
□ Sound Track
□ Fusion
□ Trip-Hop
□ Pop/Funk
□ Blues Rock
☑ Noise
□ Electronica & Dance
□ General Rock
□ Pop
□ Drum n Bass
□ General Alternative
☑ Power Pop
☑ New Wave
□ Swing
□ Electronic
□ french
□ Sound Clip
□ ÿþ

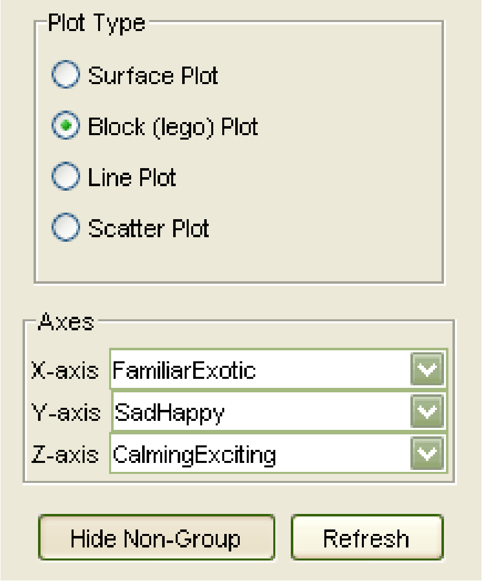
Tolerance 0 20 40 60 80 100
□ FamiliarExotic 0 20 40 60 80 100
□ SadHappy 0 20 40 60 80 100
☑ CalmingExciting 0 20 40 60 80 100
☑ LofiHifi 0 20 40 60 80 100
☑ Grooving 0 20 40 60 80 100
□ ClassicContemp... 0 20 40 60 80 100
□ ConserviativeEx... 0 20 40 60 80 100
□ Tempo 0 20 40 60 80 100
□ Rating 0 20 40 60 80 100
□ Distance 0 20 40 60 80 100

Plot Type
● Surface Plot
○ Block (lego) Plot
○ Line Plot
○ Scatter Plot

Axes
X-axis Grooving
Y-axis CalmingExciting
Z-axis LofiHifi

A	B	C
Russian Hill	Jellyfish	Power Pop
12-String Guitar - 03 - Ojo	Leo Kottke	guitar
Babys Coming Back	Jellyfish	Power Pop
Cotton Candy	Pram	General Unclassifiable
Tell Me Something Good	Rufus (Featuring Chaka Khan)	Funk
Santa Maria	Gotan Project	General Electronic
An Ordinary Day	Bent	General Electronic
One Scotch, One Bourbon, O...	Wayne Toups	General Unclassifiable
ÿþA	ÿþT	ÿþP
Magic Love	Bent	General Electronic
True To Me With Lady Bird	Llorca	Lounge
me and my baby got a good thin	lyn collins	Funk

Figure 5: Screenshot showing all application components



Plot Type

○ Surface Plot

● Block (lego) Plot

○ Line Plot

○ Scatter Plot

Axes

X-axis FamiliarExotic

Y-axis SadHappy

Z-axis CalmingExciting

Figure 6: Visual Controls

pane (Figure 6). We will discuss each of these in turn.

4.3.1 The Table View

The table view (in Figure 5, labelled as ListView) is a straightforward list of all of the data items that are chosen by the application given the current selection criteria. It is updated any time any of the criteria are updated. The table can be sorted by any column, and columns can be reordered, but is not otherwise malleable.

4.3.2 The Visualization Pane

The visualization (visual control) pane (Figure 6) displays a graphical view of the selected data. Using the visual control pane, the user can control the nature of the visualization. All of the visualizations are 3D in nature, with the axes each representing one of the real-valued attributes. The mapping of attributes to axes is controlled by the Axes pulldown menus.

Below this is the ”Show/Hide Non-Group Data” button. This changes how the display reacts to data that is not part of the group that is selected by the predicate supplied by the user. If the button is set to ”show”, then all data is displayed in the visualization. Data points that are not selected by the user’s predicate are shown in dark grey. Data points that do match the predicate are shown in a vivid purple.

If the button is set to ”hide”, only the data that matches the user’s predicate is displayed in the visualization. If this is the case, colors are used to indicate how the

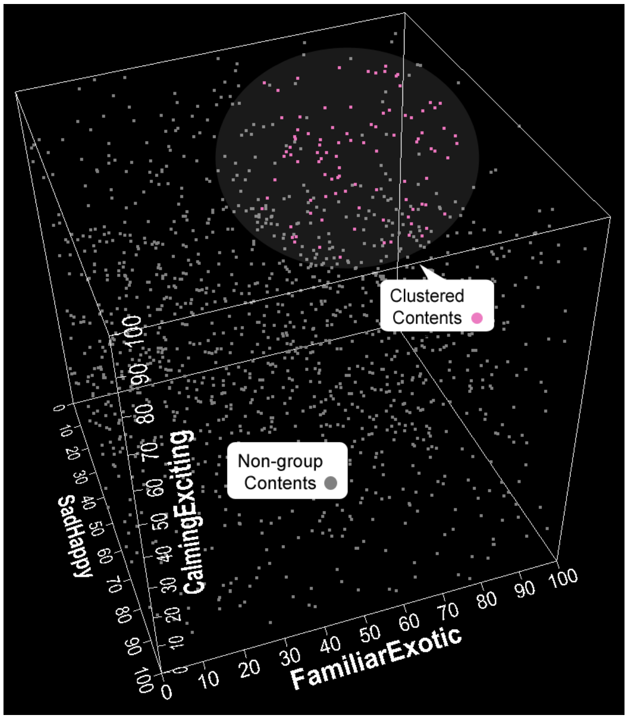


Figure 7: Clustering in Scatter View showing 'non-group' contents as well

data match the nominal aspects of the user's predicate, e.g., what genre each song belongs to.

Four visualizations are supplied by the software. The primary visualization is the scatter view. This maps each data point to a point in 3D space, forming an explicit representation of the selected data. Figures 7 and 8 depict the scatter view.

Other visualizations are aggregated representations. They take two of the dimensions, bin the data into groups, and then display a view of the third dimension that shows aggregate information about those groups. This can most easily be seen with the blocks view in Figures 9, 10, 11 and 12.

The statistics for each block represents an aggregation of the data for all of the points that fall within the square on the xy plane that intersects the box. This allows the user to see a more global view of the data that might elude him or her when confronted with the complexities of the scatterplot view.

Other aggregate data views include the Line view (Figures 13 and 14) and the Surface Plot view (Figures 15 and 16), which essentially show the same information as the Blocks view, but with a different representation.

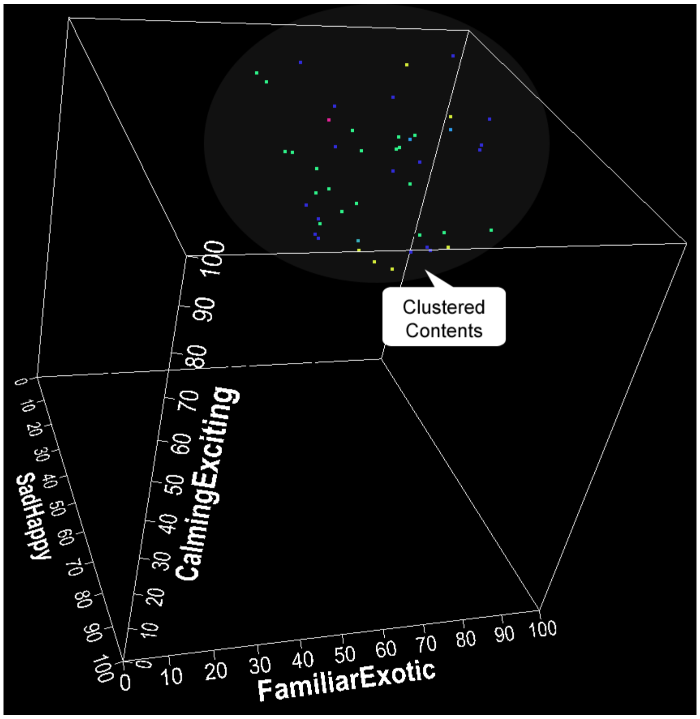
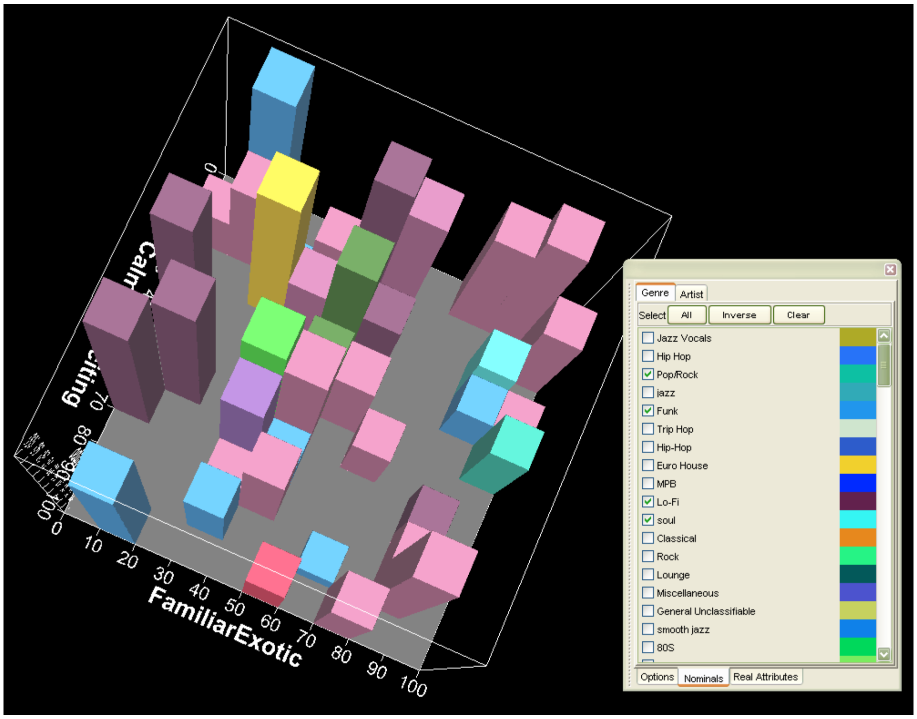


Figure 8: Clustering in Scatter View with nominals shown as colored points



- ☐ Jazz Vocals
- ☐ Hip Hop
- ☑ Pop/Rock
- ☐ jazz
- ☑ Funk
- ☐ Trip Hop
- ☐ Hip-Hop
- ☐ Euro House
- ☐ MPB
- ☑ Lo-Fi
- ☑ soul
- ☐ Classical
- ☐ Rock
- ☐ Lounge
- ☐ Miscellaneous
- ☐ General Unclassifiable
- ☐ smooth jazz
- ☐ 80S

Figure 9: Filtering in Block View

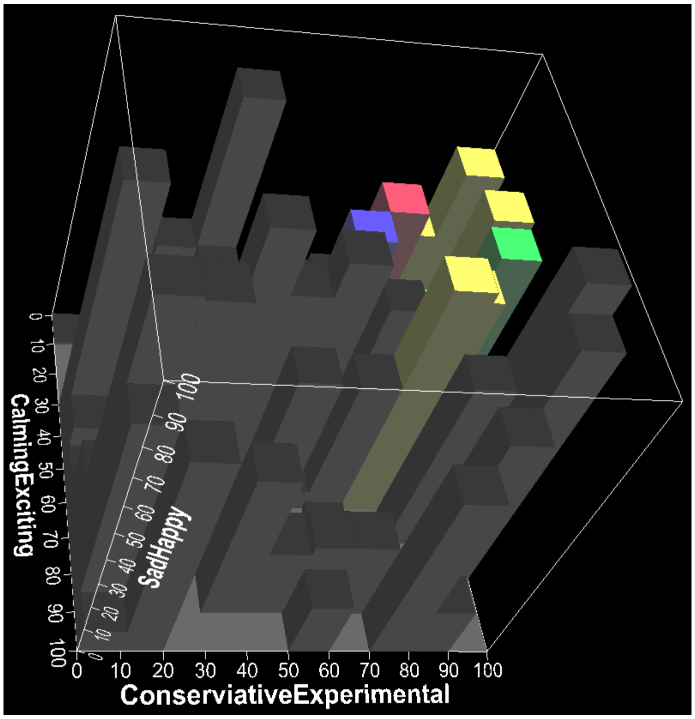


Figure 10: Clustering in Lego (Block) Plot View I

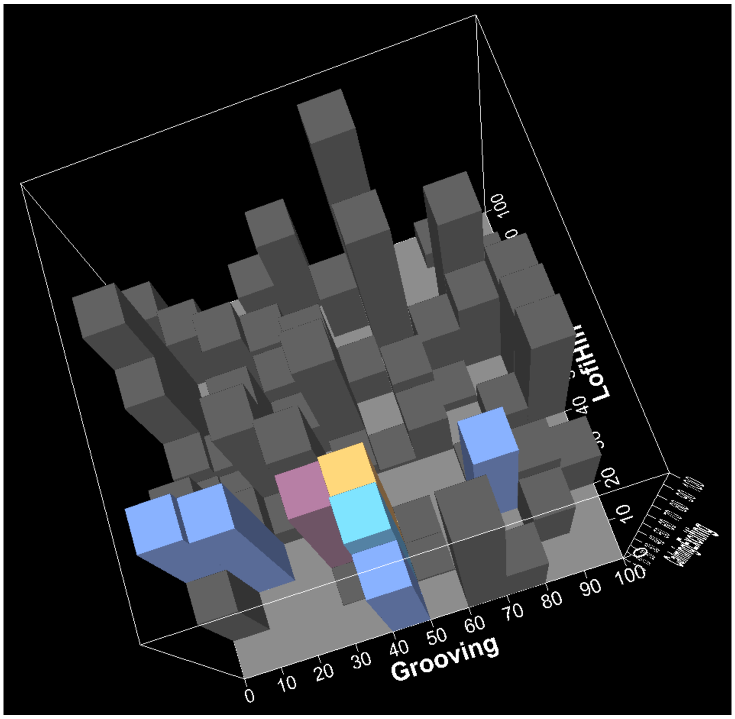


Figure 11: Clustering in Lego (Block) Plot View II

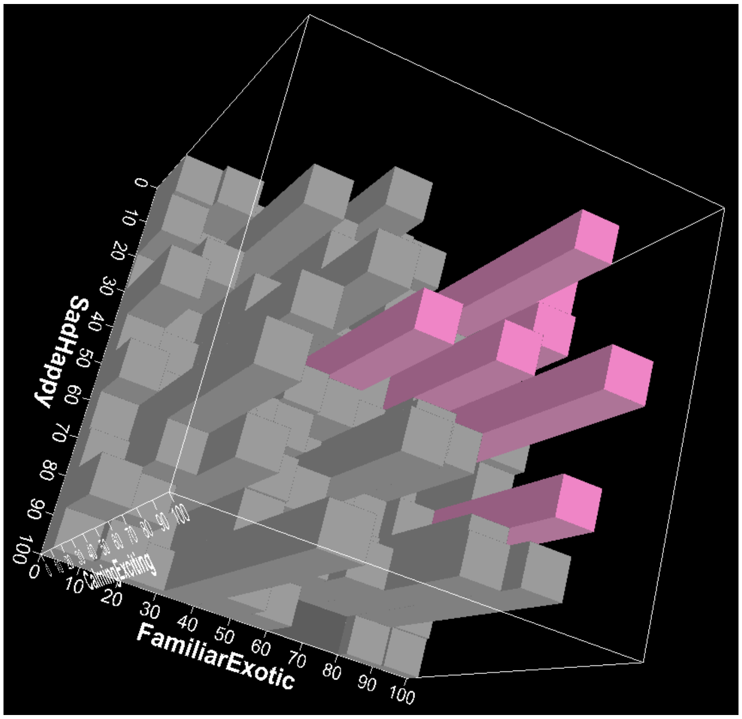


Figure 12: Clustering in Block View with Group and Non-Group contents shown

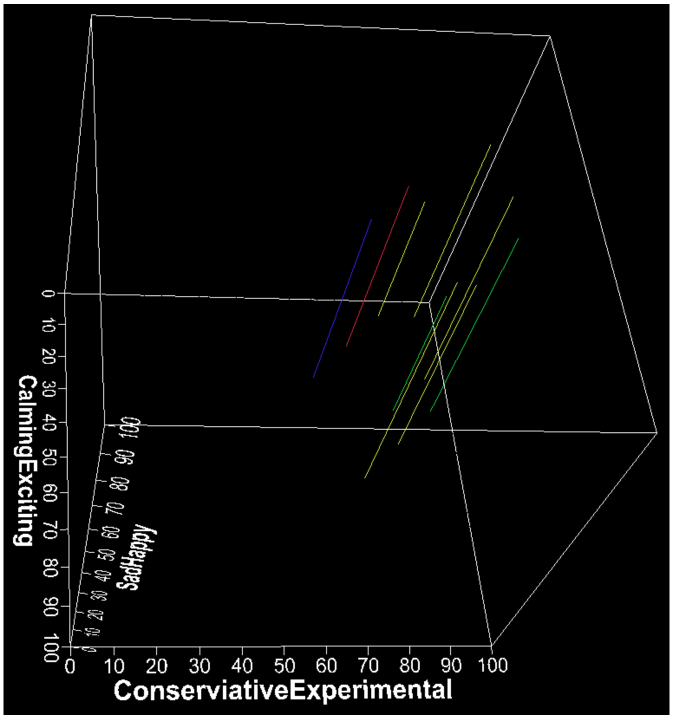


Figure 13: Clustering in Line Plot View

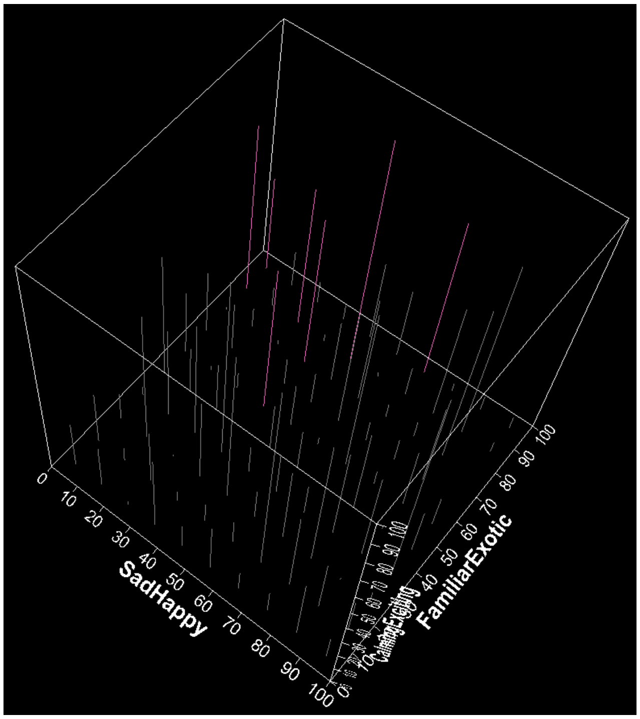


Figure 14: Clustering in Line View with Group and Non-Group contents shown

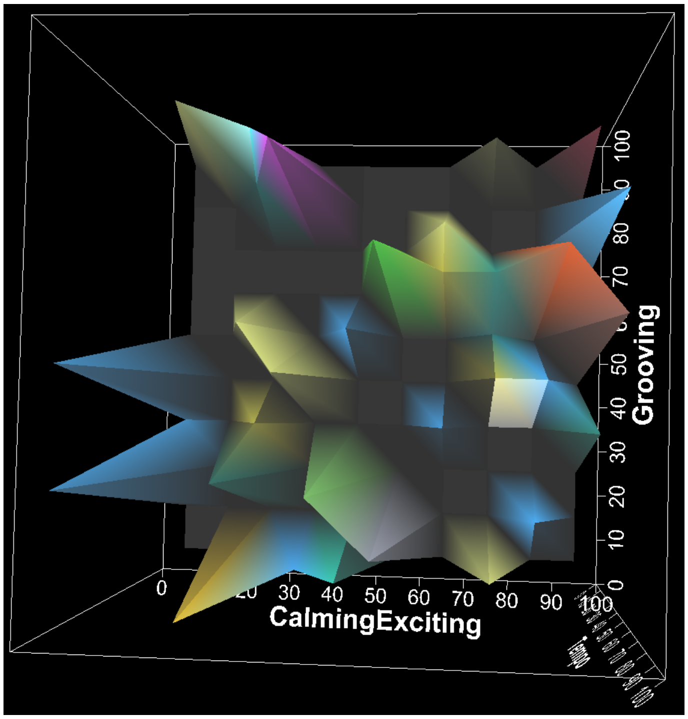


Figure 15: Filtering in Surface Plot View

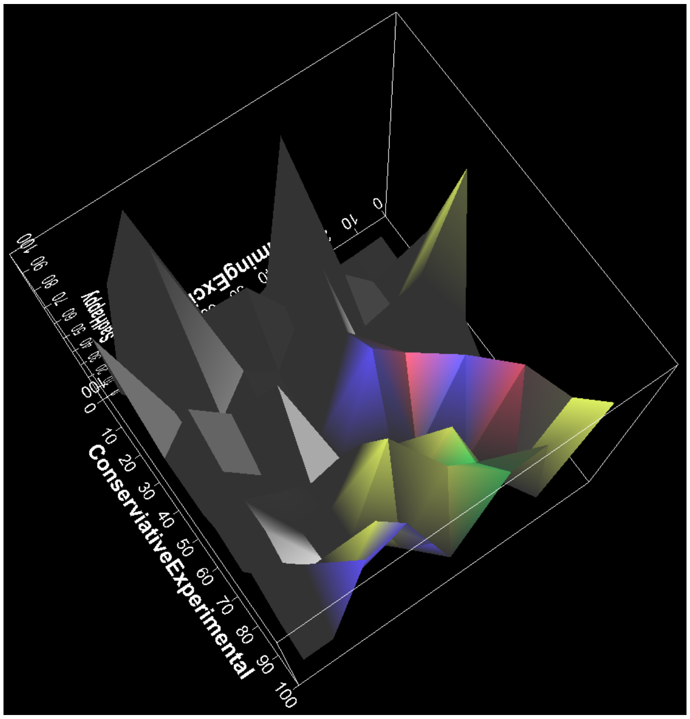


Figure 16: Clustering in Surface Plot View

5 PLATFORM AND LIBRARIES

For *nD Visualizer* we needed, (i) a library for classifying dataset using Machine Learning (ML) algorithms, (ii) a library for visualizing data points in 3D space and (iii) a platform to put these all together. As a ML toolkit we have used WEKA, for visualization we have used a 3D Plotting Pakage `org.freehep.j3d.plot` *(j3d)*, a part of FreeHEP visualization library and we have implemented our application in the Java platform using GUI components from AWT and Swing. Furthermore, we have used AspectJ to localize/encapsulate event triggering message calls in our application. Detail discussions on these follow [3].

5.1 Clustering with WEKA

We have used the WEKA [wek] data representation to store the data elements for this project, but the clustering software was implemented ourselves. Leveraging the WEKA infrastructure not only saved time, but it also means that we are open in the future to the possiblity of using actual WEKA clustering algorithms or utility classes if they prove to be advantageous.

[3] we skip discussion of Swing/AWT as it will be too general and not significant for the purpose of this report

The application clusters songs using a simple Euclidean distance measure. The criteria given by the user forms a centroid in an n-dimensional space (where n is equal to the number of properties specified by the user in the request). The algorithm then finds the k nearest neighbors of the centroid in that n-dimensional space.

This is not the only way of solving this problem. It is, in fact, a rather naive method which was implemented for efficiency reasons. Efficiency is an important concern because, given the interactive nature of the application, the criteria is likely to be constantly adjusted by the user, resulted in many executions of the clustering algorithm. Even getting the simple algorithm to be sufficiently speedy required quite a bit of tweaking.

More sophisticated algorithms might offer compelling advantages, however, and therefore we plan to investigate them further. Consider a distribution composed of two adjacent spherical clusters in an n-dimensional space. If the user requests a point that is halfway between the center and the edge of one distribution along the line segment that connects the centroids of both distributions, then a request from the user, if for a sufficient number of songs, would include many songs from the other cluster. Arguably, however, songs that are a little farther out but in the same cluster ”belong” in the list of request songs more than songs that are closer in terms of Euclidean distance, but live in another cluster.

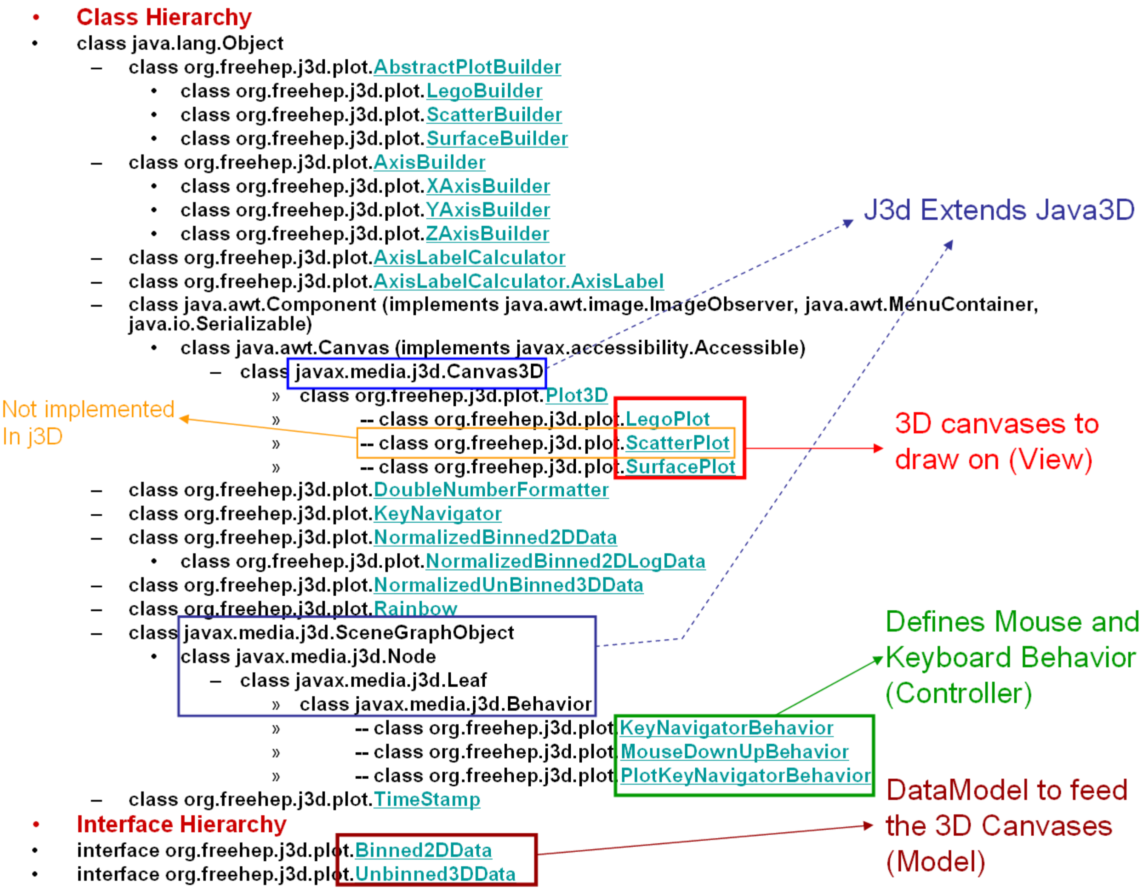
The answer to this problem is to allow the cluster to move in the direction of the center of gravity of the points as they are sampled. This is a characteristic of most clustering algorithms. We plan to investigate if any of these are fast enough for our interactive application, or if approximations can be written that are suitable.

The package structure showing our extension to WEKA is shown in Figure 20 that is included as an Appendix.

5.2 3D Plotting Package

FreeHEP is a huge library for scientific visualization avaialble from `www.freehep.org`. We used Java 3D plotting package (*j3d*) from FreeHEP, `org.freehep.j3d.plot` (available as a single .jar). *j3d* uses Java3D underneath and for basic functionality we dont need to know intricacies of Java 3D. But *j3d* had some of the core functionalities not implemented that we absolutely needed for our project so we had to implement those ourselves that made us tinker with Java 3D a bit.

Figure 17 shows the package structure (API) of *j3d*. As with all standard Java UI application/libraries, *j3d* too follows the MVC (Model View Controller) design. *j3d* extends `Canvas3D` from Java3D to give an abstract class `Plot3D` that already incorporates standard mouse and keyboard behaviors using the classes `KeyNavigatorBehavior`, `MouseDownUpBehavior` and `PlotKeyNavigatorBehavior`. Thus standard graphic operations in the 3D space like panning, zooming and rotating have already been implemented. *j3d* provide two types of *viewers* for plotting three dimensional data with one missing viewer



- **Class Hierarchy**
- class java.lang.Object
  - class org.freehep.j3d.plot.AbstractPlotBuilder
    - class org.freehep.j3d.plot.LegoBuilder
    - class org.freehep.j3d.plot.ScatterBuilder
    - class org.freehep.j3d.plot.SurfaceBuilder
  - class org.freehep.j3d.plot.AxisBuilder
    - class org.freehep.j3d.plot.XAxisBuilder
    - class org.freehep.j3d.plot.YAxisBuilder
    - class org.freehep.j3d.plot.ZAxisBuilder
  - class org.freehep.j3d.plot.AxisLabelCalculator
  - class org.freehep.j3d.plot.AxisLabelCalculator.AxisLabel
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Canvas (implements javax.accessibility.Accessible)
      - class javax.media.j3d.Canvas3D
        - » class org.freehep.j3d.plot.Plot3D
        - » -- class org.freehep.j3d.plot.LegoPlot
        - » -- class org.freehep.j3d.plot.ScatterPlot
        - » -- class org.freehep.j3d.plot.SurfacePlot
  - class org.freehep.j3d.plot.DoubleNumberFormatter
  - class org.freehep.j3d.plot.KeyNavigator
  - class org.freehep.j3d.plot.NormalizedBinned2DData
    - class org.freehep.j3d.plot.NormalizedBinned2DLogData
  - class org.freehep.j3d.plot.NormalizedUnBinned3DData
  - class org.freehep.j3d.plot.Rainbow
  - class javax.media.j3d.SceneGraphObject
    - class javax.media.j3d.Node
      - class javax.media.j3d.Leaf
        - » class javax.media.j3d.Behavior
        - » -- class org.freehep.j3d.plot.KeyNavigatorBehavior
        - » -- class org.freehep.j3d.plot.MouseDownUpBehavior
        - » -- class org.freehep.j3d.plot.PlotKeyNavigatorBehavior
  - class org.freehep.j3d.plot.TimeStamp
- **Interface Hierarchy**
- interface org.freehep.j3d.plot.Binned2DData
- interface org.freehep.j3d.plot.Unbinned3DData

J3d Extends Java3D
Not implemented In j3D
3D canvases to draw on (View)
Defines Mouse and Keyboard Behavior (Controller)
DataModel to feed the 3D Canvases (Model)

Figure 17: Package Structure of j3D

- `ScatterPlot` that we had to implement ourselves. The data model to be fed to these viewers are implementation of two interfaces `Binned2DData` and `Unbinned3DData`.

Finally three kinds of viewers were available for us to plot the points in the three dimensional space -

1. `LegoPlot`: This viewer gives a visualization as shown in Figures 9, 10, 11 and 12. Setting some attributes this viewer can also produce visualizations as shown in Figures 13 and 14.

2. `ScatterPlot`: This viewer gives visualization as in Figures 15 and 16.

3. `SurfacePlot`: This viewer gives visualization as in Figures 5, 15 and 16.

`LegoPlot` and `SurfacePlot` both take `Binned2DData` as the data model whereas `ScatterPlot` takes `Unbinned3DData` as the data model. Using *j3d* involves providing appropriate instances of Data-Models by implementing these interfaces and hooking-up custom controllers to synchronize these MVC elements with rest of the Application.

There is a very fundamental difference between the two interfaces for data model in *j3d*. One important one is Binned2DData treats the value at z-axis as a function of value at x and y axis. This makes only one point being shown in the z axis for a given set of (x,y) pair in the viewers that use this interface as their data model. These are clearly visible in the figures we have listed above. ScatterPlot, on the other hand uses Unbinned3DData that does not impose that constraint thus we can plot all the points that can possible exist in a 3D space using this viewer. Unfortunately this viewer is not implemented in *j3d*, thus we had to make some extra effort in implementing it.

6 APPLICATION ARCHITECTURE

Figure 5 is an architectural diagram that depicts the basic design of *nD Visualizer*. It shows the message/event flow between the core components, some of the components are tagged as ≪*xxx*≫, where *xxx* would imply the special role a component might have. They can be regarded as stereotypes in UML. Our application works on an event-based model and the components are designed based on well defined interfaces. That adds a great deal of design flexibility and good separtion of concerns.

6.1 Design

We have imlemented a decent setup of Publisher-Subscriber design pattern [BS96] (also known as the Observer pattern [GHJV95]) following the standard Observer-Observable realtionship between classes available in Java [Coo]. This set up gives rise to a complex setup of MVC architectural pattern [BS96] [KP88] in our application. We will walk through Figure 18 and Figure 5 to explain this setup.

The application is entirely generic in terms of the contents and attributes it can handle. All the necessary controls for nominals and reals (Figure 5, UIControls) are generated dynamically while the application loads by reading a data file (Figure 18, ≪*Data*≫). This data file contains two kinds of information, (i) Content_Header: a list of all the nominal and real attributes that can be used to filter/cluster the contents and (ii)Content_Data: the list with values given for each of the nominal and real attributes for each of the content in a repository.

Reading the Content_Header, AppModel generates all the required UIControls - this step actually creates the controls shown in Figure 3 and 4. AppModel imlements an interface that extends two other interfaces, RealChanges and NominalChanges. These interfaces model any changes made to the real and nominals and are ”Publishers” or are capable of generating event in case of any changes. AppModel hooks up all the UIControls it genreates with this ”Publisher” such that actions such as changing slider positions and clicking checkboxes generate events

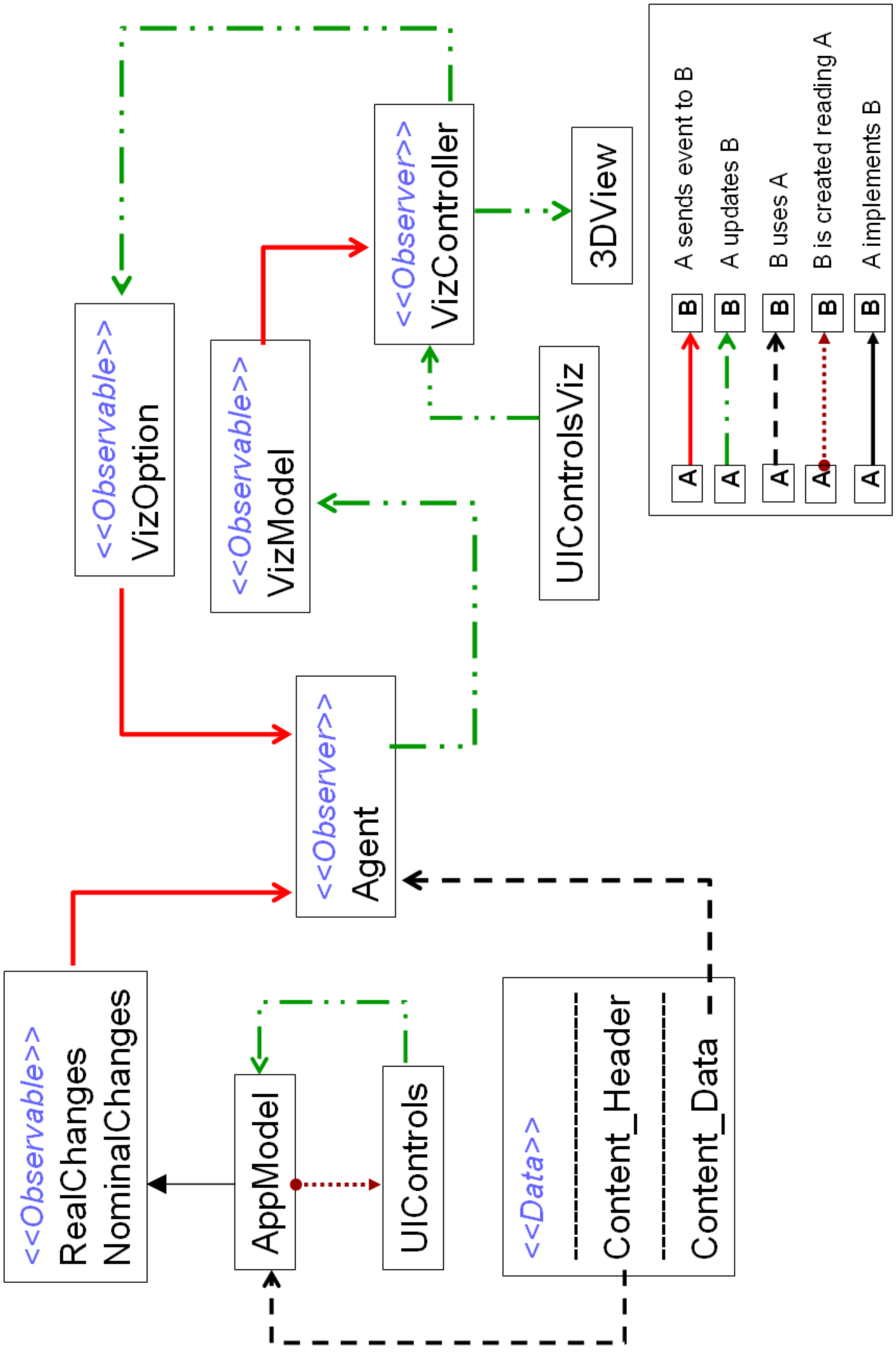


Figure 18: Application Architecture

corresponding to real and nominal changes. With each change the application generates a ”filtering predicate” that gets propagated with the event.

The `Agent` acts as a ”Subscriber” to these real and nominal changes and receives a new ”filtering predicate” on each change. It also reads the `Content_Data` from ≪*Data*≫ and performs filtering and clustering on the dataset upon receiving each new ”filtering predicate”.

`3DView` represents all the possible views we have discussed earlier, it is controlled by `VizController`. `UIControlsViz` represent all the controls related to visualization options (Figure 5, Figure 6) these controls notify `VizController` of any change. Any new change in this case sends message to `VizOption` that is another ”Publisher” to which the `Agent` is again subscribed to receive event notification. Each new visual configuration change will thus generate a ”visual predicate” if necessary that reaches the `Agent`.

With every new ”filtering predicate” or ”visual predicate” `Agent` receives, it updates the `VizModel` that is the data model that `VizController` uses to feed the `3DView`. `VizModel` again is a ”publisher” that generates an event whenever it is changed. `VizController` is a ”subscriber” to `VizModel` so that every new update to the `VizModel` gets reflected in the `3DView`.

Not shown in the diagram is the ”List View” that too gets updated in the same manner as `3DView`.

6.1.1 MVC in *nd Visualizer*

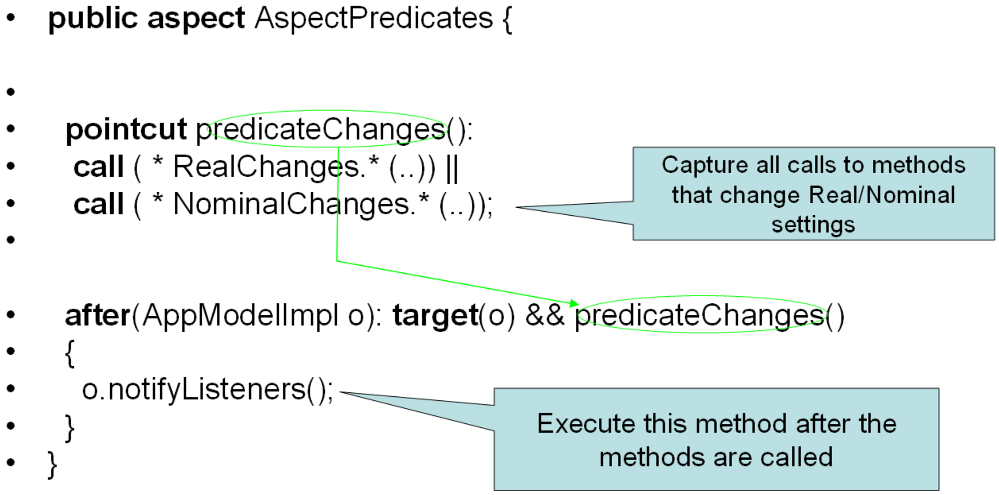
From above discussion it should be clear that there are two different MVC setups in the design,

- *MVC 1:* Model = `VizModel` ,View = `3DView` and Controller = `VizController`.
- *MVC 2:* Model = `VizOption` ,View = `VizModel` and Controller = `Agent`.

6.2 Localizing event triggering method calls with AspectJ

We have used AspectJ [KHH$^+$01], [WEB], to encapsulate all the relevant method calls that trigger the event generation. AspectJ is an extension to Java that supports Aspect Oriented Programming (AOP) [KLM$^+$97]. A detail discussion of AOP and AspectJ is beyond the scope of this paper so we briefly describe an annotated Aspect code (Figure 19) to illustrate the basic concept.

Figure 19 shows `AspectPredicates` that captures all the method calls that represent changes in real or nominal attributes with a *pointcut*, `predicateChanges`. It also has an *after advice* that says if any of these mehtods are called the ”subscribers” to those events should be notified. An advantage of using Apsect is it does not require



```
•  public aspect AspectPredicates {
•
•    pointcut predicateChanges():
•     call ( * RealChanges.* (..)) ||
•     call ( * NominalChanges.* (..));
•
•    after(AppModelImpl o): target(o) && predicateChanges()
•    {
•      o.notifyListeners();
•    }
•  }
```

Capture all calls to methods that change Real/Nominal settings

Execute this method after the methods are called

Figure 19: Using AspectJ to localize event triggering method calls

us to inject the code that notifies the ”subscribers” inside all those method definition that cause the event generation. Another usage might be to use Apsect to queue and schedule events in case performance goes down when lot of events need to be generated. [4]

7 CONCLUSIONS AND FUTURE WORK

7.1 Using actual ratings data

During the course of this project, we did not get to actually rate a collection of songs and observe the effectiveness of the visualizations and the clusterings in a real-world context. This is obviously of the highest priority, and is prerequisite for the other enhancements. It may make sense to enhance the table view to provide a way to change the ratings inside the program's interface itself.

7.2 Visualization

The visualizations, while visually appealing, do not offer enough to the user to justify their existence (with the possible exception of the scatterplot view, which at least gives a good global view of the user's data in three dimensions). This may lead the user to pay attention primarily to the table view. But there *must* be a way to provide some useful synthesis of the data in dimensions higher than 3. Hence,

[4] For instance, generation of hundreds of events within seconds in our application might cause problems like memory leaks. However this issue can be resolved in may different ways.

we will continue researching visualization methods in the hope of finding something worth incorporating.

7.3 Clustering

As mentioned in the ”How Clustering Works” section, other clustering methods may be worth investigating, including traditional clustering algorithms or gaussian mixture models, if we can find implementations (or write them) that can run fast enough for interactive use.

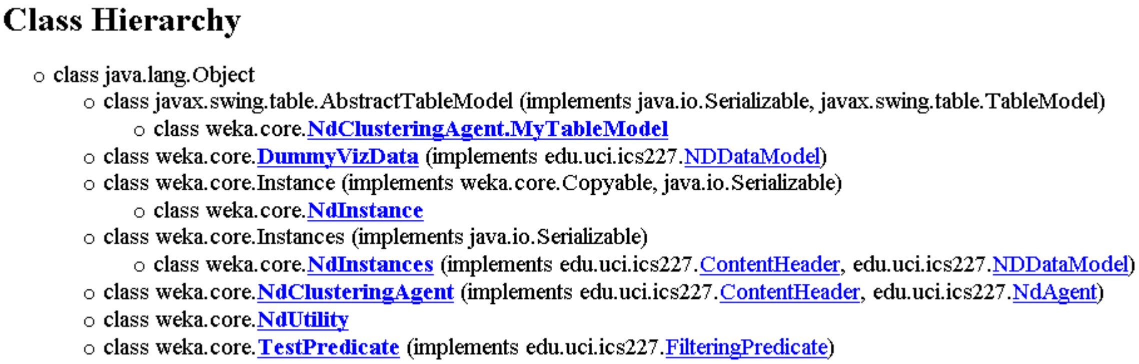
7.4 Other functionality

The only thing that’s necessary for creating a cluster is to pick a centroid. In the current implementation of the application, the centroid is explicitly picked by the user. Another useful way for the user to select a centroid would be for them to select a song and tell the application to find ”more songs like this”. The selection process could be guided additionally by the user specifying what attributes are to be used for selecting nearby points.

Finally, the application should be able to export playlists for use with external media players, and potentially interface directly with a media player itself, so that one can fully enjoy the fruits of the computer’s labors.

8 APPENDIX: API/Package Structure

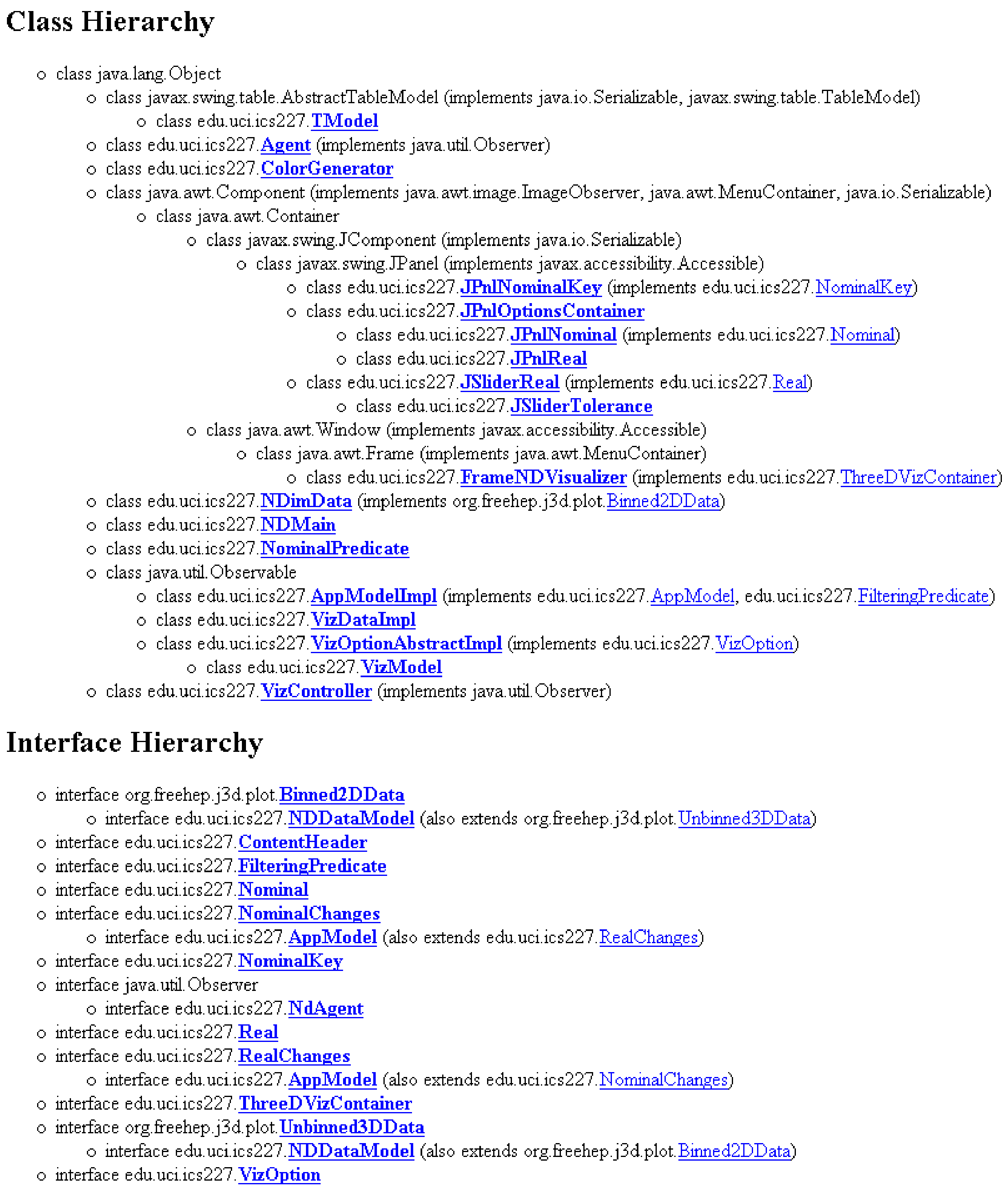
We have included the API/Package structure of the WEKA extension we implemented (Figure 20) and of the core Application (Figure 21) itself. These two packages along with the *j3d* package (Figure 17) constitute our entire application - *nD Visualizer*.



Class Hierarchy

- class java.lang.Object
    - class javax.swing.table.AbstractTableModel (implements java.io.Serializable, javax.swing.table.TableModel)
        - class weka.core.**NdClusteringAgent.MyTableModel**
    - class weka.core.**DummyVizData** (implements edu.uci.ics227.NDDataModel)
    - class weka.core.Instance (implements weka.core.Copyable, java.io.Serializable)
        - class weka.core.**NdInstance**
    - class weka.core.Instances (implements java.io.Serializable)
        - class weka.core.**NdInstances** (implements edu.uci.ics227.ContentHeader, edu.uci.ics227.NDDataModel)
    - class weka.core.**NdClusteringAgent** (implements edu.uci.ics227.ContentHeader, edu.uci.ics227.NdAgent)
    - class weka.core.**NdUtility**
    - class weka.core.**TestPredicate** (implements edu.uci.ics227.FilteringPredicate)

Figure 20: API `weka.core` (Agent)



Class Hierarchy

- class java.lang.Object
    - class javax.swing.table.AbstractTableModel (implements java.io.Serializable, javax.swing.table.TableModel)
        - class edu.uci.ics227.**TModel**
    - class edu.uci.ics227.**Agent** (implements java.util.Observer)
    - class edu.uci.ics227.**ColorGenerator**
    - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
        - class java.awt.Container
            - class javax.swing.JComponent (implements java.io.Serializable)
                - class javax.swing.JPanel (implements javax.accessibility.Accessible)
                    - class edu.uci.ics227.**JPnlNominalKey** (implements edu.uci.ics227.NominalKey)
                    - class edu.uci.ics227.**JPnlOptionsContainer**
                        - class edu.uci.ics227.**JPnlNominal** (implements edu.uci.ics227.Nominal)
                        - class edu.uci.ics227.**JPnlReal**
                    - class edu.uci.ics227.**JSliderReal** (implements edu.uci.ics227.Real)
                        - class edu.uci.ics227.**JSliderTolerance**
            - class java.awt.Window (implements javax.accessibility.Accessible)
                - class java.awt.Frame (implements java.awt.MenuContainer)
                    - class edu.uci.ics227.**FrameNDVisualizer** (implements edu.uci.ics227.ThreeDVizContainer)
    - class edu.uci.ics227.**NDimData** (implements org.freehep.j3d.plot.Binned2DData)
    - class edu.uci.ics227.**NDMain**
    - class edu.uci.ics227.**NominalPredicate**
    - class java.util.Observable
        - class edu.uci.ics227.**AppModelImpl** (implements edu.uci.ics227.AppModel, edu.uci.ics227.FilteringPredicate)
        - class edu.uci.ics227.**VizDataImpl**
        - class edu.uci.ics227.**VizOptionAbstractImpl** (implements edu.uci.ics227.VizOption)
            - class edu.uci.ics227.**VizModel**
    - class edu.uci.ics227.**VizController** (implements java.util.Observer)

Interface Hierarchy

- interface org.freehep.j3d.plot.**Binned2DData**
    - interface edu.uci.ics227.**NDDataModel** (also extends org.freehep.j3d.plot.Unbinned3DData)
- interface edu.uci.ics227.**ContentHeader**
- interface edu.uci.ics227.**FilteringPredicate**
- interface edu.uci.ics227.**Nominal**
- interface edu.uci.ics227.**NominalChanges**
    - interface edu.uci.ics227.**AppModel** (also extends edu.uci.ics227.RealChanges)
- interface edu.uci.ics227.**NominalKey**
- interface java.util.Observer
    - interface edu.uci.ics227.**NdAgent**
- interface edu.uci.ics227.**Real**
- interface edu.uci.ics227.**RealChanges**
    - interface edu.uci.ics227.**AppModel** (also extends edu.uci.ics227.NominalChanges)
- interface edu.uci.ics227.**ThreeDVizContainer**
- interface org.freehep.j3d.plot.**Unbinned3DData**
    - interface edu.uci.ics227.**NDDataModel** (also extends org.freehep.j3d.plot.Binned2DData)
- interface edu.uci.ics227.**VizOption**

Figure 21: API `edu.uci.ics227` (Application Core)

References

[BS96] Meunier R. Rohnert H. Sommerlad P. Buschmann, F. and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons, 1996.

[Coo] James W. Cooper. *The Design Patterns Java Companion*. URL: http://www.patterndepot.com/put/8/JavaPatterns.htm.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[KHH$^+$01] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In J. L. Knudsen, editor, *Proc. ECOOP 2001, LNCS 2072*, pages 327–353, Berlin, June 2001. Springer-Verlag.

[KLM$^+$97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *11th Europeen Conf. Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242. Springer Verlag, 1997.

[KP88] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model view controller user interface paradigm in smalltalk-80. In *Journal of Object-Orientated Programming*, volume 1, pages 26–49, August/September 1988.

[WEB] AspectJ official site URL: (http://www.aspectj.org).

[wek] WEKA official site URL: (http://www.cs.waikato.ac.nz/ ml/weka/index.html).