TagMatrix – a data visualization for TraceTrack. Improvements on the Lama framework

GIRARD Thomas

June 2009

Abstract

Recommendation systems are an important part of a lot of web applications today. Web retailers need them to increase sales. They're used in most media-related application, be it movie rental, music stores, news aggregators. What makes a good recommendation system ?

In the particular field of music recommendation, what data is available and how can we use it properly ? What's the importance of implicit user feedback versus explicit user feedback ? How do users interact with recommendation systems, how do they understand them, and how do the systems understand the users. These are all important questions as web applications get richer, more complex and try to use more and more data to give users what they need.

When data gets more complicated and harder to understand, the need for good data visualizations appears. Data visualizations can be used to understand recommendation systems, both from a developer and a user point of view. By understanding the visualization, we may understand the data better and use it more wisely.

For this project, we extended TraceTrack, a music recommendation system and Lama, the framework it's built on. We added a strong data visualization based on the user feedback on this system. We also tried to understand how the data that's used both for the recommendation and the visualization works.

This semester project was done at the Human Computer Interaction Group from EPFL under supervision of Dr. Pearl Pu and Nicolas Jones.

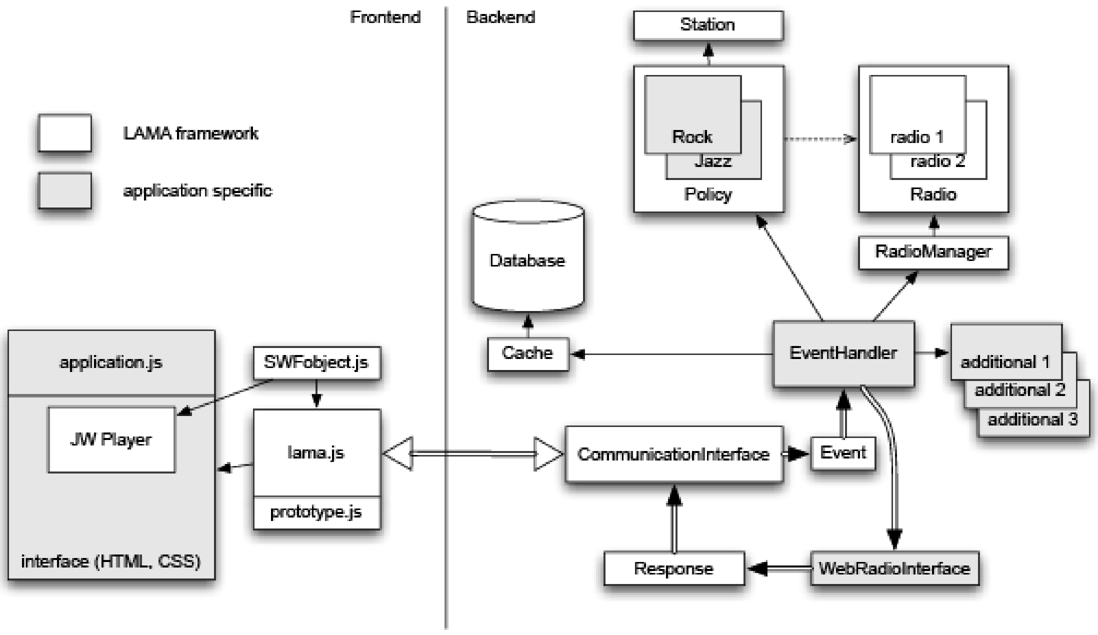
Contents

Introduction

This section introduces the framework on which we worked. It also provides a global overview of the basis on which our system is based as well as our motivations and goals in this project.

What is Lama ?

Lama (short for Last.fm Musical Amplifier) is a PHP-based web framework for quickly building applications using the last.fm radio service. Its design makes it easy to create new music recommendation systems. Lama is being developed at the HCIG[1] at EPFL.



This diagram shows how Lama is designed. The main thing to consider is that it uses a thin-client architecture. The frontend, or client-side, only has to execute some javascript and plays music through a flash player. All requests ar handled using AJAX and processed in the backend (server-side) using PHP.

Notice where the darker parts of the diagram are. Those parts are the ones that any application based on the framework will have to implement. This shows that the framework (white parts) handles most of the tedious work and that an application developer can mostly focus on interface design and recommendation logic.

More information on Lama including full documentation and source code is available online[2].

1 Website of the Human Computer Interaction Group: http://hci.epfl.ch
2 Lama website: http://whizz.ch/

What is TraceTrack ?

TraceTrack is a web application built using Lama. It uses a recommendation system based on different alternatives and produces a time-line (the so-called tracetrack) that represents the way the user advances through different musical recommendations. Tracetrack is accessible through the Lama website.

Goals

Our goal was to create a new interface and/or data visualization based on Lama. Last.fm provides a lot of data to its user, both regarding music and their listening habits. TraceTrack adds a new layer of data that last.fm doesn't provide. Using those two data sources offered us many possibilities in order to create an innovative visualization

A secondary goal was to improve and extend Lama and TraceTrack. Some bug-fixes were made, some new functionality was added and modifications were made so that the new visualization was well integrated with the old TraceTrack player.

Modifications on Lama

The Lama framework was built from the start with genericity in mind. One of the goals at conception was that the framework could work with another music provider than last.fm. This is done using many layers of abstraction in the application[3]. One of the downsides of this is that the framework is sometimes a bit convoluted and complex. For example, the simple processing of a user-action from the UI passes through a dozen of classes and layers, making it quite hard to understand what's going on.

My objective was not to change the framework completely and in particular I didn't want to alter its core architecture. However some simple things could be done to improve the framework and make it easier to use.

Some bug fixes were also made. And we had to write additional documentation since the original one was lacking on some points (mainly the database architecture).

A major improvement was also made so that the framework is now capable of storing meta-data on the track the users have previously listened to. This was a required feature in order to implement our data visualization later on.

Environment modifications

One of the problems we encountered at the start of the project was that all applications used a different database, a different last.fm account, different callback urls and so on. Setting up new instances of the tracetrack application took us a very long time because nothing worked correctly because of this.

3 The full documentation for the framework is available on request, consult the Lama website for more information.

These are small changes that affect the way some calls are made in the application, they consist mostly in a relocation of some of the configuration variables, and the addition of functions for some common tasks. These changes didn't add much functionality but will probably make the framework easier to use and extend in the future. They also solve most of the difficulties that we encountered while installing copies of the application.

One of the longer-term goal is to change parts of the implementation of the framework in a way that would make it work so that a single instance of Lama could host many different applications. These changes are a first step towards this goal.

New settings location

The static class Settings was added. It simply contains application-wide settings like database authentication information, cookie directory and so on. It replaces the old way the settings were accessed (through the database, or in global arrays) for more security and consistency.

The Setting class is also used for initializing things in the application that need to be done at each page call, for example setting some environment variables or the page-encoding. Such actions were previously done on a page-by-page basis and creating new pages or modifying old ones was an error-prone process due to this.

New global functions

Some global functions were added to centralize and encapsulate tasks that occurred often in the code. For example, before this was modified, the connection to the database was made the same way in about 10 different files. It's now done via a call to the connectDatabase() function.

Other functions that were implemented are redirect() that replaces the old header() calls. A sqlQuery() that encapsulates mysql_query() (although most of the database calls are made using specific classes in the database packages).

A debug() mechanism was added that logs messages to a text file. Such mechanism was lacking and is really useful given the fact that the backend of the framework may hardly display errors to the frontend.

Authentication callback change

A site-wide script was added to the /utilities folder, it's called callback.php. Its function is to receive all last.fm authentication callbacks from all applications and dispatch them correctly to the proper folder. In the future this file could also be used to maintain a shared user database for all Lama applications.

The old way of doing this was that every application had its own callback url. This url is still used but this new script adds a centralized point for all applications that may be usedful later.

The use of the script is optional (the old way of doing the callback still works). An application may use the script by specifying its url in the lama API configuration panel.

In order to identify the application from which the call is being made, you may either specify an application name (it's then necessary to modify the script so that the name is recognized) or specify the full path to the callback file in the Lama framework. For example our modified version of TraceTrack specifies the following callback url:

http://whizz.ch/utiliites/callback.php?name=tracetrack2**&**

(note the final & that is necessary because of the way last.fm sends its token.

Saving Tracks

This is the most important modification to the framework and one that was essential to the later parts of our work. Until now, the application didn't have any mechanism to save what the user did in previous sessions. Each time a user connected to the application, it was as if he had never used it before.

While this was not really problematic because of the way tracetrack worked (it didn't provide information that's really meaningful from one session to another other), it was essential to implement such a mechanism for the creation of a complex visualization

This was done by storing tracks in the database. There are many ways to do this and here are the options that we considered.

One possible option was to store only minimal information on the tracks, and to recover the full information if needed in the last.fm database. This implementation has some advantages: it is easy to implement, doesn't consume much space in the database. However it would probably have been slow to use, because we'd have to make many last.fm API calls each time we wanted to access information on a track.

The second possibility was to build a full database with complete meta-data on each track (tags, artist name, duration and so on). This is harder to implement and has one major downside: if at any time we become interested in a new kind of information on the tracks, the database would've to be modified with the new information. Worse: we'd have to build a script each time so that older tracks also get updated with new meta-data.

In the end a solution that's in-between those two was chosen. We actually store in the database (in the *tracks* table) a small subset of meta-data, the kind of meta-data we'd be interested in making SQL-queries on. We also created a new PHP class, named BetterTrack (because the class Track already exists in Lama, more on that below). The standard meta-data is stored alongside the serialized BetterTrack object so that we may retrieve it from the database and have full metadata access later on.

Additionally, it was important for us to store the tags associated with each track in the database directly. Given the nature of the tags, it was necessary to insert them in a new table in the database. This is the *tracktags* table. Each record in this table associates a tag and its number of uses to a previously stored track. Note that the implementation of tags in this way is not memory-efficient and a solution would be to eliminate redundancy in tag names by creating a uniqueTag table so that tags that are often used wouldn't have their name stored each time. This was not implemented to simplify the use of the database since memory usage is yet far from critical.

The BetterTrack class

The class was conceived to be easy to use, while at the same time providing access to all the meta-data we may need, be it from last.fm or from what we already stored. All data is accessed from an associative array and data is easy to add and update, be it from a local source or from last.fm.

A BetterTrack is built from a track title and artist, it fetches all known data from last.fm at creation time (making a "track.getInfo" API call). The most commonly used meta-data, is automatically stored in an associative array. Then it's possible to read any additional data we may want in the XML that last.fm provided us.

As previously mentioned, Lama already uses a class named Track. The Track class is a really basic container for track meta-data, limited to artist name, title, album name and a few other fields. It could be interesting to replace all uses of Track by corresponding calls to a BetterTrack. The BetterTrack class may need to be changed a little bit (to conform to the interface of the old Track). But this would make later modification and use of track meta-data easier for possible improvements of the framework in the future.

A visualization: TagMatrix

Music visualizations

There seems to be an increase in the number of people that use recommendation systems to find and listen to new music that they like. Maybe people are trying to move away from more passive means to discover music (like radio or TV) and become attracted to more active ways to find new music. Last.fm is a web-service that allows people to register what music they're listening to, and provides recommendations of new bands that users may like based on their current taste. The system also provides statistics to users from their listening habits. However this functionality is quite limited and no advanced visualization of the data is provided. Last.fm also offers an API that, among other things, allows developers to create visualizations from user data.

Last.fm and visualization services that rely on its data seem to answer to two main different questions: "What should I listen to ?" and also, "How do I listen to music ?". Tracetrack kind of answers the former, but gives little insight on the latter. A visualization is what is needed to answer this second question and that's what we were interested in providing.

Another aspect of the problem is that, while most recommendation systems and visualizations try to give information on the user about themselves, few are giving feedback to users as to how they are perceived by the system. This is an important point, because if users understand how the system work, what the system sees from them, and how their behavior affects it, then they may understand better what the system tells them and why. They may even adapt to the system in order to beneficiate more from it, and it could also be a factor that gives users more trust and appreciation to the system.

Next we'll reflect on what data TraceTrack, Lama and last.fm provide us that may lead to an interesting visualization We'll also consider a few examples of other music visualizations and what's interesting with them.

What data do we have ?

Our basic goal was to provide a visualization that relied on TraceTrack use and not on last.fm statistics directly. Basically TraceTrack gives us one simple thing: a list of songs the user listens to, and for each song, something that (possibly) links the two song (maybe a common tag, an artist link, a song-link, or sometimes nothing).

From there we can access any meta-data on any of those songs from last.fm and try to find patterns that may give an insight to how the system and the user behave.

We also have less explicit information that we can get from implicit user action (this was one of the key-point in the conception of lama and TraceTrack: to use implicit user feedback). For example the fact that a user skips a track quickly may be an indicator of something that we could work on. Another thing we can exploit is temporal data, that is: how does the tracks a user listens to change over time or between two different sessions.

As was just mentioned, what links two tracks is usually a simple bit of meta-data, either an artist, a "similar song" link or a tag. In all three cases we can suppose that there's some similarity in tags, even when the link is not explicitly a tag link. Artists usually perform music that stays in the same or similar genres over different tracks and albums. Songs that are marked as similar are often in the same genres too.

This led us to decide that working with tags was probably the most interesting thing we could do as TraceTrack works with this data implicitly at different levels.

Tags are not always easy to manipulate. From one song to another, the number of tags may vary a lot (from zero to hundreds of tags) and for any tag the number of people that applied it varies greatly too (from a few people to millions). Tags are user-driven data, which means that they're not always reliable. Tags may be redundant or uninteresting. For all these reasons it's not always easy to work with them.

It's also worth thinking about what tags means to users and how they are (or aren't) useful for music recommendations. Some tags are simply synonyms for genres or trends (e.g: rock, pop, 90s) and can obviously be used for recommending similar music. Some describe music well based on more complex appreciations (e.g: happy, party, erotic) which describe very well what users think and may want, but is hardly translatable in musical terms. Some tags are nearly useless (examples include "vvvvv" or "in prison").

Before seeing how we used tags in our system, let's see some other visualizations that are based on last.fm. Some use tags, some don't, we tried to take a small selection of visualizations that share some similarities with what we've done.

Other visualizations

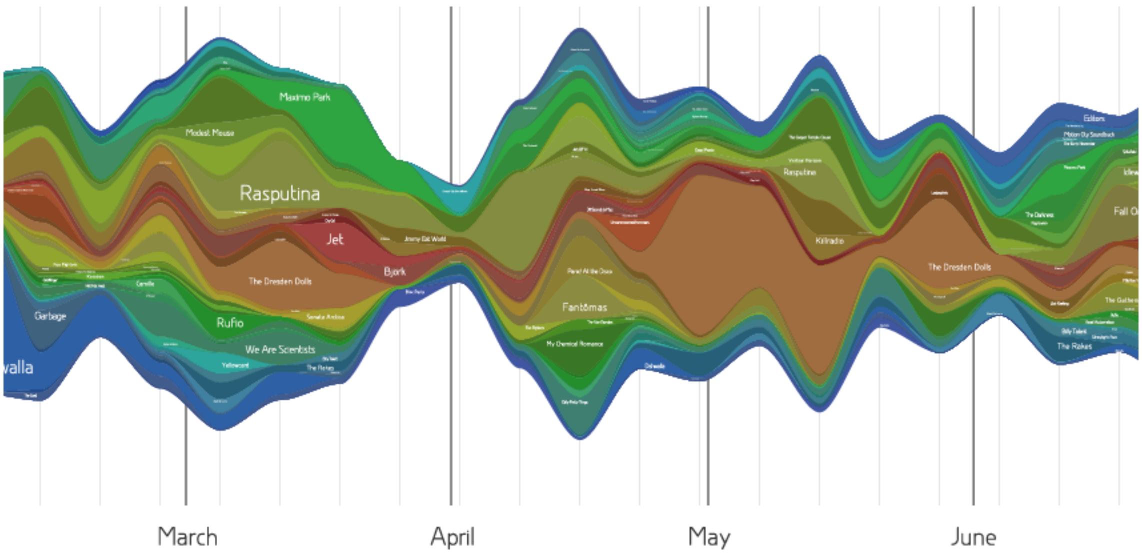
Lastgraph[4]

Lastgraph is a very attractive visualization that produces a time-line showing what bands the user listened to over time and in which proportions. Additionally it uses colors and position in the graph to try and differentiate what artists a user listens to often or for long periods of time, and those that the user only discovered recently.

The general outline of the graph shows the total number of tracks the user listened to (week by week) and each artist has it's delimited part in the general shape that's created. Frequent and/or ancient artists are closer to the center, while newer ones are in the borders. The visualization uses a complex algorithm to produce curves as smooth as possible so that shapes representing a single artist have the smallest possible distortion.

This visualization gives lots of information on the volume of music one listens to, on the importance of some artists, the duration during which the user may have peaks of interests on some artists and so on. It's visually rich and complex while remaining simple to read.

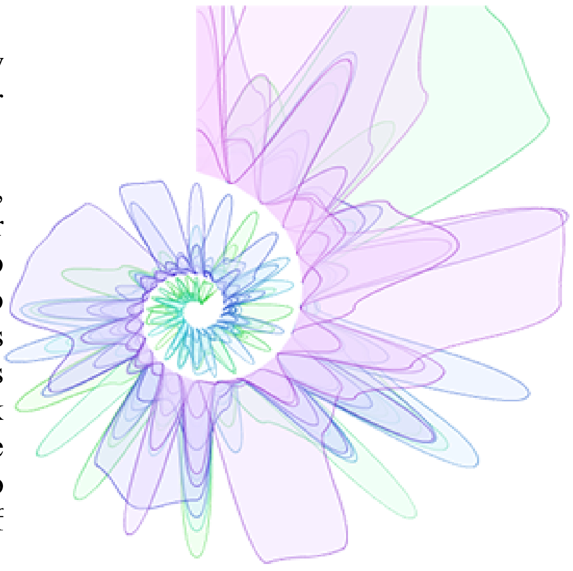
One of the downsides is that it's creation is very complex: lastgraph uses an asynchronous rendering system, which may take several minutes to render a graph. It has little configuration options and offers no interactivity.



4 Lastgraph is available here: http://lastgraph3.aeracode.org/

Last.fm Spiral[5]

Last.fm Spiral shares some similarities with Lastgraph. It's also a time-line (time-spiral actually) that shows the number of tracks the user has listened to over time, but it separates curves for different artists instead of stacking them like Lastgraph does. This gives a visualization that's a bit harder to read, but makes pattern emerge that Lastgraph doesn't show.



The application makes it possible to show only one or a subset of artists at a time for better readability.

For example when tried on a test account, the Spiral allowed us to see how that user usually had short periods (from five to fifteen days) during which he tends to listen to a single artist a lot. These peaks of interest rarely overlap and there seems to be little repeat of artists from one peak to another. Additionally it becomes quite easy to see what artist one listens to regularly but without specific pikes of interest.

The choice of making the time-line in a spiral form is probably what makes this visualization stand out from others, but it may not be so good of a choice because the center of the spiral becomes harder to read. In particular, the closer to the center the line gets, the smaller the amplitude of the curves become. Consequently it becomes difficult to compare the intensity of different peaks that occurred at distant times.

Why add a visualization to TraceTrack

As previously seen, TraceTrack, as it was before this project answered to a user's need to find new music. The trace itself gives some information to the user, but it's quite hard to read (the user has to understand the color-codes, the symbol letters) and doesn't give a long-term insight on how the user or the system works.

What was needed was a visualization that interacted with the player, but was meaningful in the long term and more expressive. We already mentioned why we considered the tags as interesting data to manipulate, the next step was to take an approach that used tags on all tracks listened over time to produce its results.

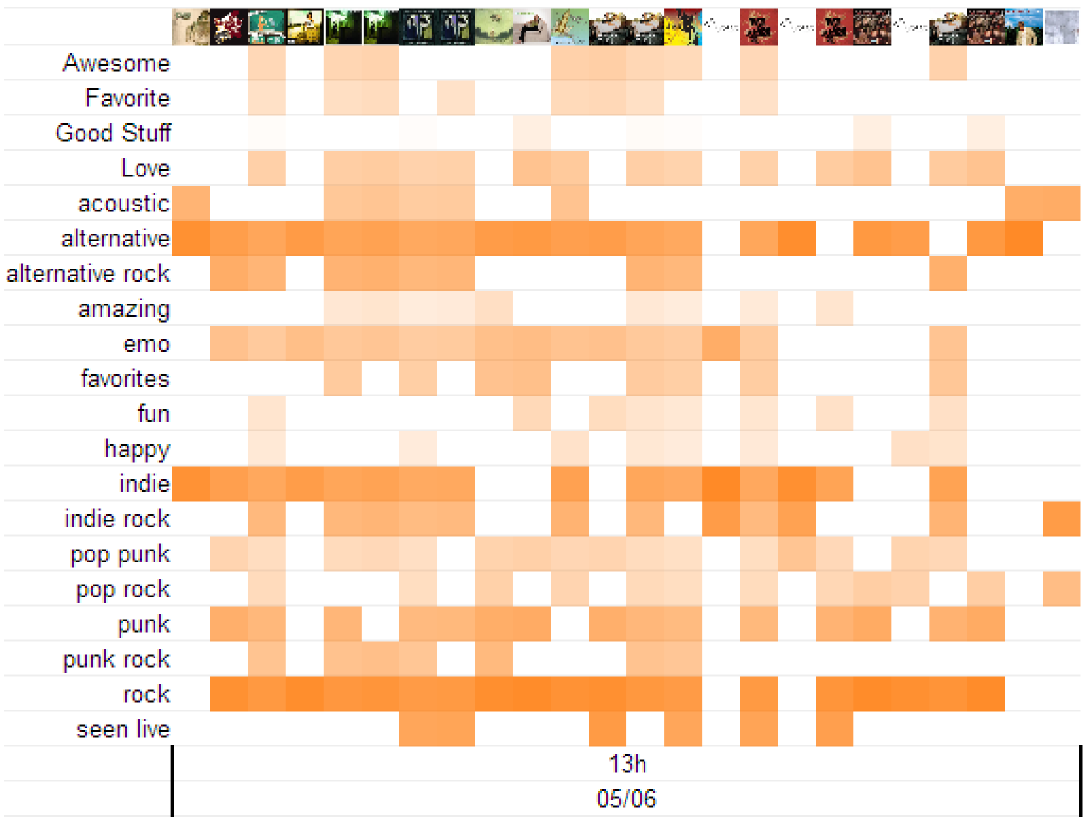
5 Last.fm Spiral is available here: http://www.diametunim.com/muse/

Layout

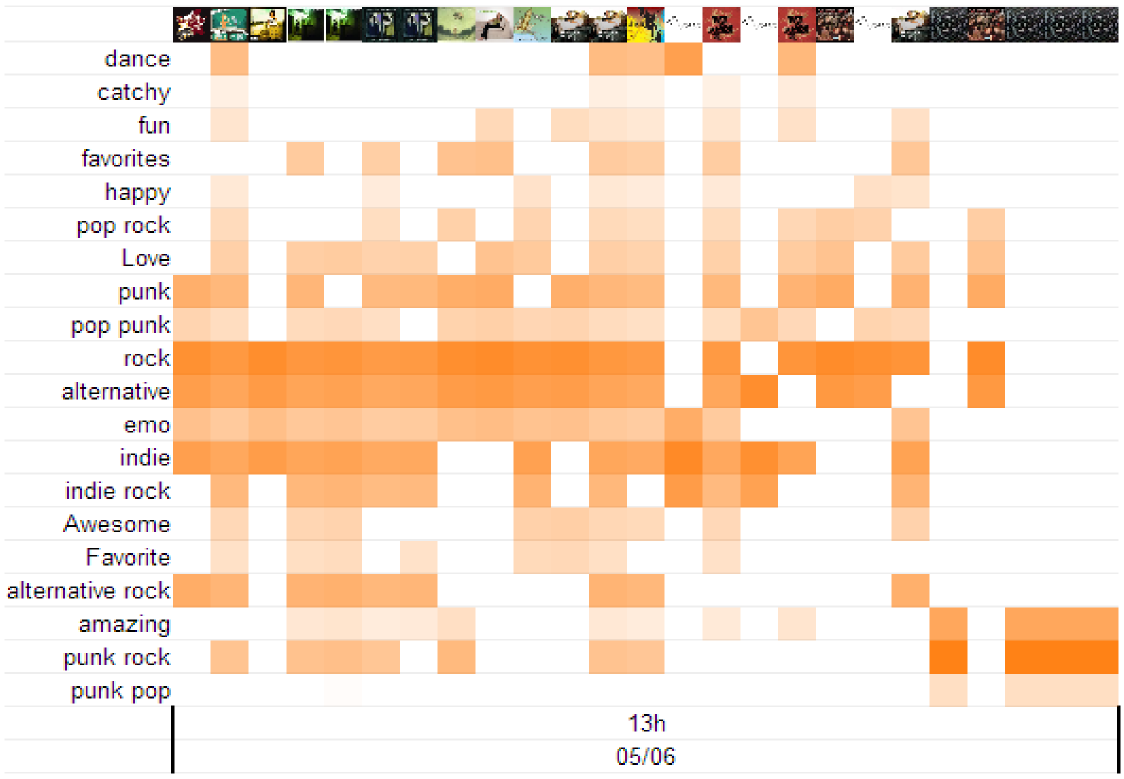
The TagMatrix is not very complex in structure but allows us to show users different things because it's easy to alter the contents of the matrix while the layout stays the same.

Basically it's a simple table with, on top, album covers that show the last tracks a user has listened. Then, on the leftmost column are tags that correspond to those tracks. In the simplest disposition, the tags are ordered by alphabetical order. To the bottom are separations that indicate when tracks were listened and allows the user to distinguish between different listening sessions.

In the next example, the tags are ordered in alphabetical order and there's only one session. The user reads the matrix and sees that he mostly listened to alternative, indie and rock music. However this is not really expressive and a sorting algorithm was implemented so that tags make a more meaningful cluster, this can be seen in the second image.



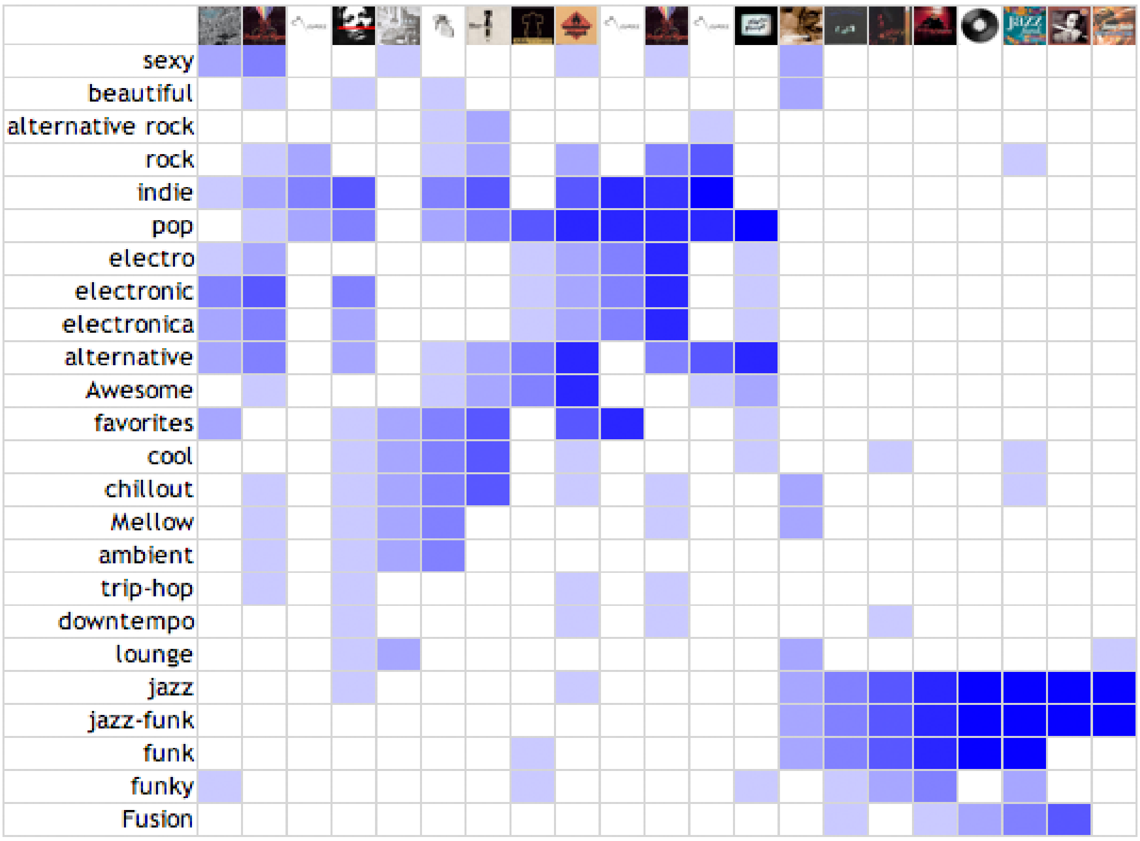
Example 1: tags ordered by alphabetical order



Example 2: tags ordered using a clustering algorithm

With this second example, the user can see what genres the music he listened to fits in. We also notice that rock, alternative and emo can be considered like similar genres. At the end of the track the user switched radio completely and started to listen to (amazing) punk rock.

This previous example is a bit too homogeneous to see the best the visualization can provide. We provide a third example that shows how the algorithm may produce different separated clusters. Here we see that the user started listening to electronic music with some indie and rock influencesm then switched completely to jazz and funk. Note that the appearance of this example differs because it's a screenshot from an old version of the application. The algorithms are the same.



Example 3: clustering algorithm produces different clusters

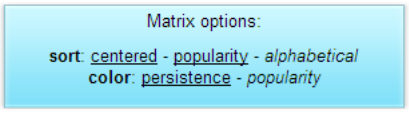
We also observed that, with this algorithm, similar tags have a tendency to get automatically grouped together. This can be seen here with the funk/funky tags, electro/electronic/electronica is another good example.

This algorithm works by creating a tag-matrix in which each tag gets assigned a similarity value, based on the number of songs this exact pair of tags are used at the same time. The algorithm then selects the tag with the higher similarity index, places it at the center. It then takes the tag most similar to the first one and places it to one side, and continues in decreasing order of similarity until the matrix is filled.

The coloration (from dark to light blue) indicates the popularity of the tag for this particular song. We took the percentage of use of this tag over all tags used, and applied a logarithmic function to try to normalize it to a 0-255 color range.

Interactivity

The next step was to give the user a possibility to alter the way the information was shown in order to have a better understanding of what was shown, and because it made the application more attractive overall.


Matrix options:

sort: centered - popularity - *alphabetical*
color: persistence - *popularity*

One may choose the way the tags are sorted, or the way the squares are colored. More options could be made available (choice of the timespan, more sorting algorithms, number of tags to consider, filtering of some tags based on criteria) but we lacked time.

The matrix is a source of information of two kinds. First it gives the user a general feedback on what he listened to and how. The matrix makes pattern emerge from users behavior. The grouping algorithm for tags will often create different clusters of tags for different listening sessions, the user will easily see how his actions (choosing a tag and sticking to it for example) reflect on the matrix.

For some users the patterns are clear, some have a long line in the center, very dense and with little variation. Some have a smaller cluster that indicate their general interest, but that's surrounded by a lot of "tag-noise", which shows that they tend to listen to very various things at the same time. Some users have clusters that vary a lot from session to session, some don't. It's at the same time easy and amusing to view our own music patterns.

On the other side, the user may use the matrix to comprehend how the system works and tries to understands them. Since tracetrack relies on implicit user interaction to collect data and make recommendations, users may not actually know how their actions will affect the system. By looking at the tag matrix they may realize how, for example, choosing a particular tag will reflect on what information the system will gather. A user that uses its music library a lot will probably have an noisier pattern, and may then understand that this kind of pattern doesn't give a coherent information to the system.

Still, it should be noted that for now, the matrix has only been tested by a few people so it's still not easy for us to know how well our ideas and suppositions will concretize. The way the users perceive and use the matrix may also evolve as more functionality is added to it.

Integration with TraceTrack

With the TagMatrix built and functional, more things remained to do. As it was, the TagMatrix was completely independent (visually) from the TraceTrack player. It seemed a good idea to modify TraceTrack so that the user could easily use it with the TagMatrix seamlessly.

Another thing that we decided to work on was to give the user a clearer feedback of what his interaction with the player produced as result on the TagMatrix.

In order to make the experience coherent from one session to another, we also changed the policy that the application used to choose the first song the user would listen to at login. The previous policy was to choose a random song with the acoustic tag (arbitrary choice), instead the application now chooses a song that's similar to the song the user listened to the last time he was connected.

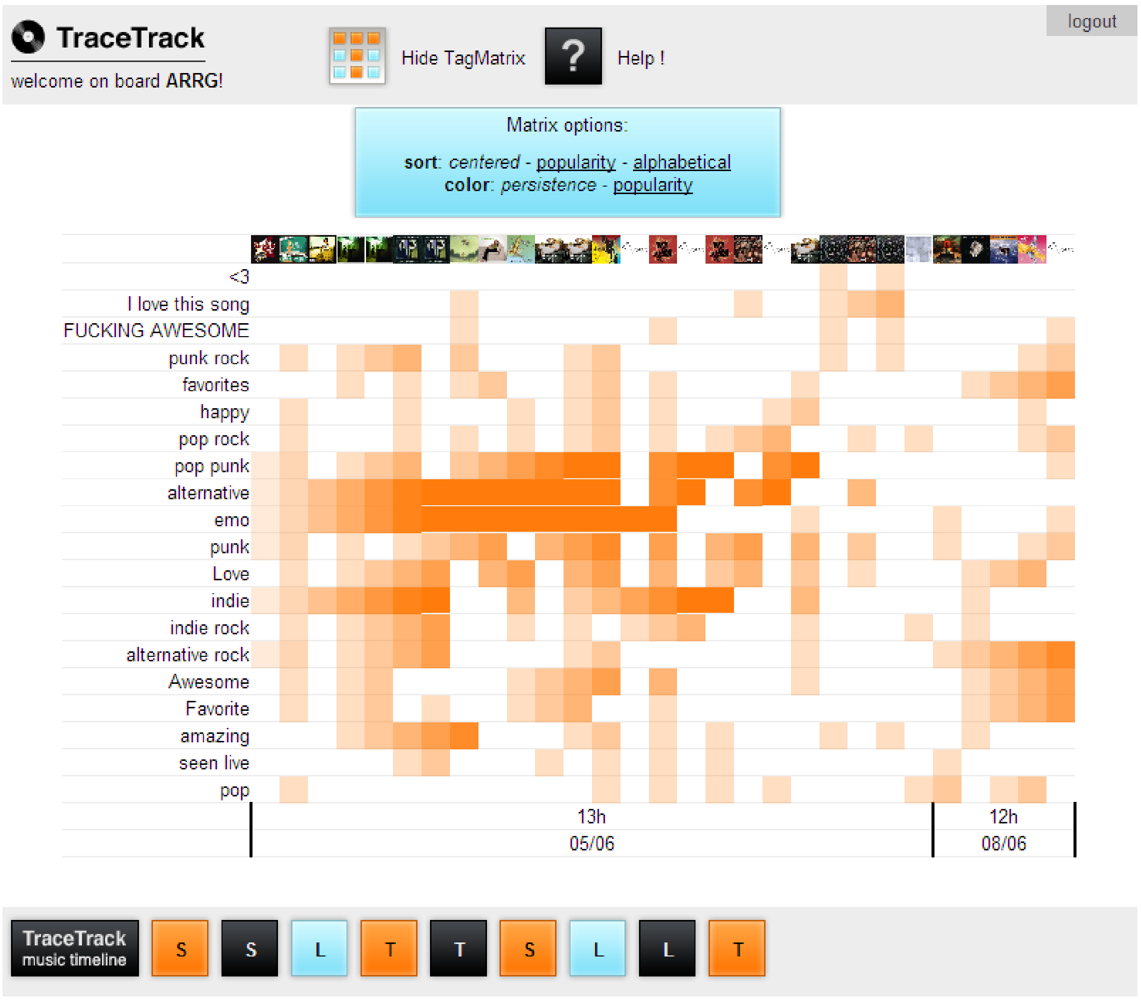
Integrating the player: modifying the interface

Given that the TraceTrack already had a visually strong design, some work was made to give the tagmatrix a similar look. This was achieved by trying to use similar colors and reusing the box design of the player.

This was not enough, because the user still had to quit the player in order to view the matrix. To solve this, the original layout of TraceTrack was slightly modified. The header is now used as a menu the user may use to show different parts of the application. A simple click on the TagMatrix icon will pop-up the matrix over the player, while the music still plays in the background. Another click and the matrix is gone with the player accessible again.

Given the increasing complexity of the application, some time was also spent making it more user-friendly. A small explanation was added to the front page so that the user know what the application is about, and how the inscription process works.

Inside the application a help box was added that briefly explains how the player works, what is the TagMatrix and the TraceTrack.



Matrix options:

sort: *centered* - popularity - alphabetical
color: *persistence* - popularity

Screenshot of the TraceTrack interface as it is now.

Development of the project

Here are a few notes on different problems we met during the project and on some challenges that weren't addressed in the previous pages. We also take a look at some of the possible improvements for the continuation of the development of Lama, TraceTrack and the TagMatrix.

Difficulties and improvements

No debugging means

Lama doesn't provide any good mechanism for debugging. When an error happens in the backend, most of the time the user gets absolutely no feedback on what's going on. No logging is done for errors. The only way to try to understand what happens is to inspect the AJAX communication messages. This gets impossible if for any reason the frontend cannot parse the messages from the backend.

This is really problematic and we lost about two weeks of development time because of a stupid error that gave no message and couldn't be debugged.

Implementing more informative error messages, debugging mechanisms and even reliability mechanisms should be a priority for the next cycle of development.

No database documentation

The database wasn't documented at all when we started the project. This was quite unfortunate especially given the fact that some table were in the database but never used, and that some other tables followed a non standard and unusual scheme.

This is now dealt with because the documentation was written during the project.

However it should be noted that database interaction is currently done in a weird way through custom made classes and that it could be interesting to rewrite the whole database driver (for example using PDO).

Redundancy in the new features

Because the code for Lama and Tracetrack is well documented in its structure, but nearly not commented at all, it is quite hard do modify. For this reason, some of the new features implemented during the project do not use some parts of the framework and re-implement some mechanisms that could have been reused instead.

For example, the AJAX communication used to display the TagMatrix or the Help window in TraceTrack doesn't rely on the Javascript code that was initially there. It could be interesting instead to use the old code and show the matrix by creating a new command in the Lama communication protocol and by implementing the proper PHP code in the main EventHandler.

Bug Fixes

There are still some undocumented bugs in the framework. Most of those bugs result from cases where an object was expected by some part of the application (either coming from an API call, or a database read, or a client-side argument) and a null value is returned instead.

In most cases there's nothing done to check that the object is null and exceptions are thrown. Since there's no error mechanisms, this often results in an unresponsive interface for the user (no alternatives showing, or no music playing).

Improvement in TagMatrix options

A lot of new things can be done with the TagMatrix. One of the facets of it we haven't had time to develop is the time aspect. By merging listening sessions, or tracks, or larger timespans together, it would be possible to create a matrix that represents a user's listening habits over a longer period. We could imagine things such as a matrix that give tag informations for different hours of the day. There's a lot to do with those ideas.

New sorting algorithms can be developed too. We talked a lot about a clustering algorithm that would generate two big but opposed tag groups (this would be kind of the opposite of the current centralized view).

Another thing to work on is the filtering of tags. For reasons exposed earlier in this document, tags are a very heterogeneous data source. If we could automatically find ways to organize it better, by removing unwanted tags, and merging tags that are actually synonyms, we could then use them better.

On a simpler UI-related note, it would be good if the album covers on top of the tracetrack displayed informations on the track when hovered with the mouse.

There could also be a way to integrate transition types (that is, recommendation, neighbors or artist similarity, as given by the player) into the matrix.

Improving the player for the TagMatrix

As is, the player allows users to use 5 different radio alternatives. Three of them are based on users profiles and only 2 are based on the tracks themselves.

It would probably be more interesting to have a larger number of track-based alternatives. We even thought about making the radio tag-based only. The user could choose one or more tags at the time and the radio would suggest similar tags and artists. This way, the progression in recommendation would make more sense from a meta-data point of view and we could generate better visualizations

The trace itself should be improved. There's probably a lot of things that can be done to make it clearer. For example we could display album covers on the trace (that would also make a nice visual link between this and the tag matrix) so the users could keep track of the transitions between two tracks without having to remember everything in his head.