Browsing Online Music Catalogs in a Vehicle.

Connecting Automotive User Interfaces with the World Wide Web.

Stéphane Turlier
BMW Group Research and Technology
Munich, Germany
Eurécom, Sophia-Antipolis, France
stephane.turlier@bmw.de

Clemens Hahn
BMW Group Research and Technology
Munich, Germany
Ulm University, Germany
clemens.hahn@bmw.de

Sascha Gebhardt
BMW Group Research and Technology
Munich, Germany
University of Munich, Germany
sascha.gebhardt@bmw.de

ABSTRACT

The increasing amount of information available on the internet has raised a lot of challenges in terms of organization of knowledge. In the domain of music indexing, the multimedia research has produced valuable techniques to sort content and compute similarity measures based on different criteria from low-level features like acoustic properties to high-dimensional data like folksonomies. On the other hand, most of the recommender systems need a lot of user interactions to produce suitable results. Those are major drawbacks that prevent from integrating them as such in vehicle multimedia systems.

We present in this article a prototyped study which combines a new interface to browse online music catalogs and to create playlists with a hardware and software architecture designed to overcome known limitations of vehicle connectivity like limited datarates, high network latency and limited computation performance.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: User interfaces; D.2.11 [**Software Architecture**]: Data Abstraction, Domain-specific architectures

General Terms

Design, Human Factors, Performance

Keywords

Embedded vehicle software, Entertainment ergonomic, Online music, Cloud metadata

1. INTRODUCTION

The display of online music information in a vehicle encounters mainly two sorts of constraints:

Firstly it is necessary to cope with the specific modalities that are available in a vehicle (video, haptic and audio), and to overcome their limitations. Secondly it is necessary to deal with the limitations of computing and networking resources of embedded systems.

We tried to address both aspects in our prototyped study and to achieve the following goals:

- Provide the user with a graphical interface and a controller knob which reduce the complexity of the task of browsing online music catalogs.
- Improve the latency of the display of this information by caching information on a multi-layered architecture.
- Use the cloud information that is proposed by web services to improve content preselection and the quality of information that is delivered to the user.

2. RELATED WORK

This prototype is developed to deliver music into a vehicle, which best matches the drivers' music taste. The driver can tell the system his music wish and the system will automatically generate a playlist based on this input. The literature proposes several methods to automatically generate playlists [1]. They can be grouped into three different categories: (1) seed-song(s), (2) visual selection, (3) user constraints.

The first method expects one or more seed-songs from the user. In the playlist generation process similar songs to these selected songs are ordered in a playlist [19, 12, 18]. The quality of the resulting playlist strongly depends on the computation of similarity and accordingly to the available dataset of songs. Another drawback is that the user has to remember songs or artists he likes for the seed selection.

A lot of research effort has focused on the development of graphical user interfaces to display a music catalog. All visualizations aim at providing the user with intuitive understanding of the underlying music catalog. Most of them offer the ability to browse the music collection. [17] provides an annotated overview of a music collection. Similar songs are arranged close together. These clusters are presented on a two dimensional map [5]. Data from external sources and self computed information from the raw audio data are used for the similarity computation. [9] also clusters similar music on a two dimensional map. It combines contextual and content-based features of each of the songs in the available music collection. In both approaches the user can generate playlists. He can draw lines, circles or other geometric

MCMC'10, October 29, 2010, Firenze, Italy.

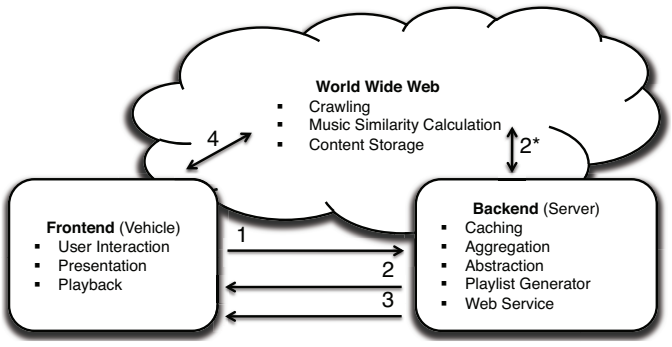

Figure 1: Conceptual model and interaction between the components frontend, backend and cloud services

shapes onto the map. The resulting playlist gets built from the songs residing closest to these shapes. We believe that drawing shapes is too distracting to be used in the vehicle while driving; therefore we propose in our prototype other methods for selecting songs.

[13, 4, 10] present a playlist generation method close to the presented approach. The user has the ability to tell the system what kind of songs he wants to listen to. Via constraint satisfaction songs are ordered into the playlist. These systems, for example, provide duration, genre, artist, album, tempo or mood as constraint.

Our approach combines the former methods. The user can tell his song constraints to the system via an easy-to-understand user interface. Based on these constraints, a few seed songs are looked up in our own database. Over similarity methods the playlist is generated automatically and then be presented to the driver.

3. PROTOTYPE ARCHITECTURE OVERVIEW

The prototype is split in two main applications: firstly the graphical user interface (the so called frontend), with whom the user can interact in the car and secondly a web service and a database (backend). A conceptual model of this architecture is presented in figure 1. The roles and tasks of the backend are to cache, to aggregate, to generate personalized playlists and to handle user administration. An advantage of this architecture is that the development of the frontend can be focused on the ergonomics of the user interactions regardless of the integration of the cloud services. This approach reduces the complexity of the frontend components in charge of integrating the music cloud services. Interfacing the frontend with a single set of methods avoids reimplementing them for every new single music service and reduces the need of software updates. Moreover, the use of a cloud similarity service is justified by the fact, that vehicle head-units have limited computation resources. This prevents intensive signal processing to compute song similarity. The role of the backend is to abstract the cloud services and to aggregate them.

A typical interaction between frontend and backend is a request of a new playlist. As described, the frontend collects the user inputs, more precisely his music wish. This information is sent to the backend (1). The backend which has already cached some music metadata from the World Wide Web, can instantly send back a preview with a few songs. Simultaneously the automatic playlist generation process starts. Cloud music similarity services are needed for this. Querying them takes a few seconds and consumes computation time, because they have to be called several times (2*). (2) and (2*) are separated from each other to offer on the one hand a fast answer to the user input and on the other hand to generate interesting playlists. Once the playlist generation process is finished, the playlist is ready to be delivered (3). The vehicle gets a download URL for each track of the playlist. With this URL the vehicle can autonomously get the audio content from the web (4). In the next sections we will explain the main components more precisely.

4. OVERCOMING THE CONSTRAINTS OF A VEHICLE UI TO CREATE PLAYLISTS

The frontend is an embedded system in the vehicle. It receives user input, communicates with the server and visually presents the results to the driver.

4.1 Vehicle Interface Constraints

Creating an entertainment system for a vehicle is a challenging task. In contrast to desktop entertainment systems, where the user can spend his full cognitive load on those, the primary task in an automotive context is to drive the vehicle. Operating additional systems like setting turning signals or switching on and off the windshield wiper are secondary tasks. Consequently operating an entertainment system is a tertiary task [16]. Therefore, a user interface for such a system needs to be designed in a simplified way as. It must not distract the user from its main task. Accordingly gaze duration needed in order to control the system has to be as short as possible and the manual load should be minimized. The increasing resolution (1280x480 pixels in our testing car) allows more details to be displayed, while the driver's distance (approximately 85cm) to the display limits the full use of this potential. The prototype proposes a tradeoff between the information density and the font and icon size. In our application we do not use font sizes smaller than 30 pixels, resulting in very few text lines that we can use.

4.2 Radio-Like Listening Behavior

Car drivers are used to radio broadcasting as a standard entertainment system in their car. It has several advantages compared to systems like CD or MP3 Players: the user has no need to care about creating playlists or selecting particular media assets, which he wants to listen to. He just switches on the radio and starts to listen. Though, radio has some disadvantages. The number of channels in a traditional radio system is quite limited resulting in little choice of auditory possibilities. Among other media the problem is quite the opposite. When dealing with CDs or MP3, the user can select the audio content of his choice, while being required to interact with the media assets directly. That implies both cognitive as well as manual load which is distracting the user from his primary driving task.

4.2.1 *Fuzzy Multicriteria Search*

In order to compensate these drawbacks, we created a system that offers a radio-like listening style by providing a never ending playlist, combined with a fuzzy multicriteria search. That means that a user defines a set of filters and that the system provides him with a playlist, containing only songs, matching his filter criteria. He can choose from five filter categories: genre (hierarchical list of 542 genres, organized below 19 top level categories), year (decades, years or ranges of time), popularity (three stages: hot, mainstream, underground), mood (choice of 25 moods) and source (virtual sources like the own listening history, the own music collection, system recommendations or loved songs). The user can add as many filters as he wants, while filters from different categories are linked by a logical AND and filters from the same category are linked by a logical OR. We expect the user to naturally assume that kind of linkage. This fuzzy multicriteria search gives the user more control over the playlists he is provided compared to traditional radio broadcasting. It also produces little manual and cognitive load due to the fact that the process of selection and handling of media assets is being avoided: when the user has specified his choice of filters, the system provides a never-ending playlist.

4.2.2 *Multi-Playlist Presentation of Recommendation*

According to the principle previously demonstrated, the user is likely to get music he appreciates. Furthermore he is only required to interact with the system when he wants to listen to something quite different or when he wants to rate or skip songs. In addition to the playlist, the user gets offered alternate choices of music in two ways. He can choose to insert one or more songs from the following alternative playlists into the running playlist: the album of the currently playing song or a selection of music that is similar to it.

4.3 UI-Prototype

The UI-Prototype we implemented for the frontend is written in Flash. The Input/Output-devices used in the prototype are the controller knob (a push-shift-rotate controller) for input and the central information display (CID) for visual output. The music playback performs over the car audio system.

4.3.1 *User Interface*

Conceptually, the frontend is split into two major modes: the filter specification mode (figure 2) and the music listening mode (figure 3). In the filter specification mode (figure 2 (1)), the user can add, modify or remove different filter criteria. We apply a fish-eye filter to the filter selection lists to be able to display longer lists with this focus and context technique. Every time he makes a change to the current set of filters, he gets a preview (figure 2 (2)) of about seven songs, that would be anywhere in the playlist if he decided to confirm his selection. When he presses the play button, a playlist matching his filter criteria is fetched from the server and the application switches to the listening mode. The listening mode is again split into three subparts: the listening history (figure 3 (3)), the currently playing song (figure 3 (1)) as well as the playlist with the album of the currently playing song and similar songs as alternatives (figure 3 (2)). These parts are displayed as a continuous chain of song icons (icons containing the cover, the mood as color and the title

Figure 2: Filter Screen: (1) Filter Selection Pane, (2) Preview Pane

Figure 3: Play Screen: (1) Currently Playing Song, (2) Playlist and Alternatives, (3) Listening History, (4) Selection Information

and the artist as text), passing the screen from right to left. The user can scroll through the chain by rotating the controller knob and has different interaction options, depending on the part where the currently selected song is located. When the currently playing song is selected, the user can rate it with a love or hate rating or switch to the filter specification. We also provide him information about the way he or the system selected this song to play (figure 3 (4)) (for example by a filter combination or because he chose to listen to the album of a song he had heard earlier). When scrolling to the playlist part, the user can select a song in the playlist to play. When the playlist part is in focus, he can also insert songs from the album or the similar songs. Those are displayed above and below the playlist. In the history part, the user is able to insert songs he heard before into the active playlist. Some markers are linked to the history chain that indicate certain events, helping the user to locate songs he listened to before. Possible events are for example switching on/off the car or opening doors. The history is linked to a folksonomy profile, and gets periodically synchronized with it. This way we can provide a global listening history in the car that covers the everyday music listening custom of the user and is linked to all of his other listening devices.

4.3.2 *Content Preloading*

The UI-Prototype also has to deal with the latency problems that rise among a mobile music streaming client. These can be especially critical in an automobile, because it can

move fast through regions with variable network coverage. In our prototype we try to overcome these problems by preloading enough music in advance. We always try to completely preload the actual song and the next song in the playlist. This is enough for a seamless music listening experience within cities or regions with adequate network coverage.

5. BACKEND FOR THE AGGREGATION OF ONLINE CONTENT

The backend runs on a server with a servlet container. It has two main components: firstly, in a weekly process, actual metadata from different sources has to be collected and cached in a database. Secondly, the backend offers a web service, which can be accessed by the frontend. This service is designed in a RESTful style [3]. The data is transferred between front- and backend in XML format.

5.1 Caching Referenced Content

In order to support the user in his playlist generation process best, the system must react to the user interactions very fast. Short reaction times reduce the risk of distraction from the main duties and responsibilities of the vehicle-driver. Thereby, it reduces the cognitive disorder. Another reason for decreasing the latency is the consumer acceptance. Most users expect a fast answer from the system after giving their input.

For those reasons, all information which has to be delivered very fast is cached in a relational database management system. In the concept of the playlist generation process it is very important for the driver, that he gets an immediate reply to his filter choice. For instance, the user chooses in his first option the filter genre g1 = rock. The system quickly provides a direct preview of 8 to 10 rock-songs. In the next step, he chooses the filter mood m1 = happy. Afterwards, a new preview has to be presented with 8 to 10 songs, which are in the genre g1 and are rated in the mood m1. This direct preview gives the driver a direct and clear response to his query.

5.2 Aggregating Metadata

There is no exclusive provider in the World Wide Web who can deliver all required metadata. Therefore, the system has to combine the information of many different metadata-providers. In order to get good seed songs for the particular filter combinations chosen by the user, the backend has to cache song metadata, tagged with the following attributes:

- name of track, artist and album
- year of release
- genre and mood information
- ratings of the popularity
- user specific ratings
- album cover
- download URL of the audio data

[14] compared data from two sorts content providers: the commercial providers which offer expert music classifications and ratings; the social networks which deliver data, provided by their users. We added to our prototype a third sort; the crawlers which analyze occurrences of artists names and tracks in the internet (music reviews, radio charts, blogs, etc.) to deliver actualized information.

5.2.1 Metadata Provided by Experts

In the caching process, sources from both groups are used. In order to collect metadata for the genre filter, the genre taxonomy from the commercial provider Rhapsody[1] is applied. The taxonomy has a fine genre structure with 19 top genres. Each genre can be extended to up to two levels of subgenres. Altogether Rhapsody classifies its songs in 542 different genres. A top fifty song list can be requested for each genre. In this first caching process metadata of approximately 13100 songs from about 3000 unique artists, released on about 4300 albums is being collected which is enriched with genre-information. The metadata received from Rhapsody also contains the song's year of release. This information can be used directly for the year filter. In the next step the database is enriched by mood-metadata. We are using expert data from Gracenote[2]. For the conception of the prototype, we had access to mood metadata for 3795 songs. Therefore, the co-occurrence between genre and mood metadata is about 29 percent in the cache database. For our use case - providing a fast preview of 8 to 10 songs - we consider that this co-occurrence is sufficient.

5.2.2 Cloud Metadata Provided by the Folksonomies and Crawlers

Furthermore, the database must be enriched with rating information for each song. To achieve this, the system collects data from a so called folksonomy and a web crawler. The social network Last.fm[3] and the web service The Echo Nest[4] provide exhaustive and innovative ratings on demand. Particularly the "hotttnesss" and "familiarity" factors of the Echo Nest are very good indicators how common and hip a song currently is. The Echo Nest does not only take into consideration the sales volumes. It additionally crawls blogs, music portals, social networks and other web pages and counts the references to artists or songs [7]. In its open web service called the audioscrobbler[5], Last.fm presents information on how often their users listened to a particular song. This playcount information combined with the number of Last.fm listeners' results in an additional ranking.

5.2.3 Combining Different Metadata Sources

Unfortunately it is not trivial to combine the different metadata sources. It is common that a song has many different spellings on the web. This problem especially increases through the free text tagging modality in the social networks. The spelling problem can be illustrated very nicely by the song "Guns N' Roses - Knockin' on Heaven's Door". [6] has identified that there are more than 100 ways to spell this song. Even if some providers try to avoid it, a request of a single song can still generate a list of multiple hits. In order to get the right hit, each song in the response list has to be analyzed. A song can be identified by its track name, the artist name and the name of the album on which the

[1] http://www.rhapsody.com/
[2] http://www.gracenote.com/
[3] http://www.last.fm/
[4] http://the.echonest.com/
[5] http://www.audioscrobbler.net/

song was released. In order to combine them, the names have to be normalized. The rules of the text normalization can be viewed at [2]. Additionally we analyzed some algorithms which try to compare two strings with regard to their spelling sound (e.g. soundex [15]) and algorithms which compute the edit distance between two text sequences (e.g. levenshtein distance [8]). When we need to disambiguate between multiple results of a provider, we calculate a matching rate between the reference data which is being searched and the different solutions. The closer they are to the reference, the higher is the rate. For example, a candidate that matches the track name and the artist name gets a higher rate than a candidate that only matches the track name.

6. USE CASES

In order to illustrate the potential of our prototype, we identified and implemented the following scenarios.

6.1 Playlist Generation

As already described, in the caching and aggregating process the system collects about 13.100 songs and stores them into our database. The playlist generation begins with collecting local metadata from the database.

After the user has specified his playlist criteria to the system, in other words the different filters have been set; the best matching songs are selected from the database. "Best" means in this context: all the songs that are in the intersection of all filters. These songs are the anchors for the new playlist. To fill the gaps between them, different providers are called for some similar songs for the particular anchors.

Using this method, a set of a few songs is gathered for each anchor, using services presented in section 5.2. In the next step, these sets of songs are converted to the final playlist. To support a smooth crossover between the anchors, the similar songs of two neighboring anchors are interlaced. This playlist generation process is initiated in parallel to the preview request by the user. This way, the playlist is ready to be delivered as the user requests for it.

6.2 Proposing Alternative playlist

As described in section 4.2.2 the user has some interaction options to control his playlist. He can insert the whole album of a currently playing song into his playlist. He can add similar, perhaps unknown music to the playlist at any time as well. The basis of this interaction is the suggested playlist, generated by the prototype. There are several strategies to respect the user's filter choice.

[11] summarizes some approaches found in the literature. Possible strategies for our prototype are: (a) to weight the chosen filter considering the sequence of their nomination (the first choice is more important than the second and so on) or (b) the finer a filter is adjusted the more important this filter is for the user. The strategies (a) and (b) do not require the user to interact; the weighting is implicit.

If the restrictive playlist generation process (limited by an intersection of all filters as described in section 4.2.1) fails, [11] suggests to relax these filters. A possible reason for the generation of the playlist to fail is that the user adjusted too many filters or filter combinations that cannot match with any known songs. If so, the playlist is built up by an aggregation of the filters, considering a relaxing strategy.

6.3 Social Use Case

The usage of the prototype is not limited to the car. Rather it connects the mobility of the car with the persistence of the social networks in the World Wide Web. This makes the system more ubiquitous. A driver can register his Last.fm account-data into the system. The system monitors the listening behavior while driving and stores this data in the user profile of the backend database. This data is sent to Last.fm (aka. scrobbling). Thereby the user can browse his own music history, generated in the car, on every device with access to internet.

Moreover, if the user connects other devices to the Last.fm services, he can reuse his global music history in the car. The prototype synchronizes the user's listening history in both ways; from the car into the social network and vice versa.

7. CONCLUSIONS

We have presented a new kind of music player for the automotive domain. Our prototype is organized in a three-layered architecture. We have developed an innovative user interface including the fuzzy multicriteria search in order to minimize user interactions while maintaining a refined user control. We introduced a caching system to reduce latency and give the user immediate and representative examples of the playlist queries. Moreover, we presented aggregation techniques of cloud services to overcome resource limitations of vehicle head units in terms of computation and software updatability and to achieve metadata enrichment over different music services. Our prototype shows through different use cases (playlist generation, alternative music recommendation and social use case) that the integration of cloud services gives new perspectives to the infotainment applications in the automotive domain far extending the current radio services.

8. FUTURE WORK

The potential of the prototype in terms of ergonomics and architectural efficiency needs to be confirmed with a user study. We plan to perform it in a vehicle under real driving conditions. In this study we will also compare different criteria relaxation strategies.

In the prototype stage we also implemented a quite simple preloading mechanism. This is enough for regions with good network coverage, but could result in leaks in areas, where the coverage is worse. This mechanism will be improved in such a way, that the system always tries to preload as many songs as possible from the active playlist, while in areas with good coverage. This way we can guarantee maximum play duration, if the user does not change the playlist.

9. REFERENCES

[1] A. Berenzweig, B. Logan, D. Ellis, and B. Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

[2] D. Ellis. Text normalization conventions for artist, album, and track names, 2010. [Available online at `http://www.ee.columbia.edu/~dpwe/research/musicsim/normalization.html`; accessed 9-June-2010].

[3] R. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, Citeseer, 2000.

[4] finetunes. musiclens - in tune with you, 2010. [Available online at `http://finetunes.musiclens.de/`; accessed 9-June-2010].

[5] O. Goussevskaia, M. Kuhn, and R. Wattenhofer. Exploring music collections on mobile devices. In G. H. ter Hofte, I. Mulder, and B. E. R. de Ruyter, editors, *Mobile HCI*, ACM International Conference Proceeding Series, pages 359–362. ACM, 2008.

[6] R. Jones. The guns n roses issue, 2010. [Available online at `http://cdn.last.fm/rj/gnr_kohd100.txt`; accessed 9-June-2010].

[7] P. Lamere. Music machinery - hottt or nottt?, 2010. [Available online at `http://musicmachinery.com/2009/12/09/a-rising-star-or/`; accessed 9-June-2010].

[8] V. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. In *Soviet Physics Doklady*, volume 10, 1966.

[9] A. Lillie. MusicBox: Navigating the space of your music, 2007.

[10] musicovery. interactive webradio, 2010. [Available online at `http://www.musicovery.com/`; accessed 9-June-2010].

[11] O. Noppens, M. Luther, T. Liebig, M. Wagner, and M. Paolucci. Ontology-supported preference handling for mobile music selection. In *Proceedings of the Multidisciplinary Workshop on Advances in Preference Handling*, Riva del Garda, Italy, august 2006.

[12] E. Pampalk and M. Gasser. An implementation of a simple playlist generator based on audio similarity measures and user feedback. 2006.

[13] S. Pauws and S. van de Wijdeven. User evaluation of a new interactive playlist generation concept. In *Proc. Sixth International Conference on Music Information Retrieval (ISMIR2005)*, volume 11, page 15. Citeseer, 2005.

[14] M. Sordo, O. Celma, M. Blech, and E. Guaus. The quest for musical genres: Do the experts and the wisdom of crowds agree? Philadelphia, USA, 2008.

[15] The National Archives. The soundex indexing system, May 2007.

[16] M. Tönnis, V. Broy, and G. Klinker. A survey of challenges related to the design of 3d user interfaces for car drivers. In *3DUI '06: Proceedings of the 3D User Interfaces*, pages 127–134, Washington, DC, USA, 2006. IEEE Computer Society.

[17] R. van Gulik and F. Vignoli. Visual playlist generation on the artist map. In *Proceedings of the International Conference on Music Information Retrieval ISMIR*. Citeseer, 2005.

[18] F. Vignoli and S. Pauws. A music retrieval system based on user-driven similarity and its evaluation. In *Proc. ISMIR*, pages 272–279. Citeseer, 2005.

[19] T. Westergren. Pandora radio - the music genome project, 2010. [Available online at `http://www.musicovery.com/`; accessed 9-June-2010].