Breemen, A., & Bartneck, C. (2002). An Emotional Interface for a Music Gathering Application.
Proceedings of the Philips User Interface Conference (PUI2002), Eindhoven.

An Emotional InterFace for a Music Gathering Application

Albert van Breemen
Philips Research, Software Architecture Group
Prof. Holstlaan 4 (WDC-1.034)
5656 AA Eindhoven
The Netherlands
albert.van.breemen@philips.com

Christoph Bartneck
Technical University of Eindhoven
Faculty of Industrial Design
Den Dolech 2, 5600 MB Eindhoven
The Netherlands
christoph@bartneck.de

ABSTRACT
Listening to music while traveling is a pleasant activity. The latest MP3 players demonstrate that storage and management of music will not be a problem in the near future. Besides listening to music the user might also want to gather new music from the internet. We propose a music gathering application that helps the user to collect music and that is able to proactively search and download music based on the user's music preferences. Furthermore, we developed an emotional interface character that provides instant and natural feedback on the status of the application.

Keywords
Emotions, music, character, agent

INTRODUCTION
Listening to music is a pleasant activity while traveling. Many people would not like to miss it and thus carry Walkmans, portable CD Players or MP3 Players to shorten their journey. Recent developments in data compression [3] and data storage [15] made it possible to store vast music collections on mobile devices, such as the iPod [1] or the iPaq [6]. The problem of data storage will become negligible in the near future. To illustrate, the first generation of MP3 players used memory modules to store up to 85 songs (256Mb). Current generation MP3 players use hard disks to store up to 10.000 songs (30Gb). It is certain that the storage capacity will increase even more in the future. The playback and music management functionality of these devices have also been solved to a large degree. Even huge music collections that reside on the latest MP3 players have usable interfaces that enable the listener to quickly browse and play the desired music [1]. The jukebox functionality of mobile devices is therefore not in the scope of this study.

Besides listening to music, the user may also use the time on a journey to search for new music and gather it through the internet. Obviously, the mobile device needs access to the internet to perform these tasks. Wireless networks become increasingly popular and will possibly become standard in public spaces dedicated to traveling, such as airports, trains and buses. Several airports, such as the Changi Airport in Singapore and the Copenhagen Airport, already offer their customers free wireless network access. Some mobile devices, such as the iPaq offer wireless network cards that enable them to use such networks. With this technology the user will be able to access the internet and gather the desired music. Gathering music includes the download and exchange of free available music and the purchase of copyright protected material. In the latter case, however, the user would like to purchase the music from the cheapest available sources.

The music scene develops very fast and new albums are released every day. The user cannot be aware of the latest releases and trends all the time. Therefore the music gathering application should proactively search and download music based on the user's preferences.

The remainder of this article is structured as follows. In the first section we discuss the proactive music gathering application. We focus on the functional requirements and the application's architecture. Next, we discuss the problem of presenting the status information of the application to the user. This motivates the subsequent section in which the user interface design is presented, in particular the usage of an emotional character that provides natural and instant feedback to the user. A simplified OCC model [10] is used for the character's emotion synthesis. Afterwards we present the results we have obtained thus far. Finally, we draw conclusions and discuss future research directions.

PROACTIVE MUSIC GATHERING
Functional Requirements
In order to derive a set of functional requirements we first created several scenarios. Scenarios are short stories that capture the essential features and characteristics of a problem, application or vision. In our case we developed scenarios of travelers who want to obtain new music. By analyzing the scenarios, we were able to derive a set of functional requirements for our application architecture. In the scenarios we did not incorporate Digital Right Management (DRM) issues and paying models even though they would play an important role for a commercial use of the application. Our research focuses on the search, download and user interface aspects of proactive music gathering. The analysis of the scenarios focused on the *information need* of the application to function properly. We identified four needs of information that are essential for realizing proactive music gathering.

First, the application should have information about the *existence of music items* such as specific songs and albums.

This information is needed in order to know what music items in general exists and can be downloaded.

Second, the application should know what kind of music the user likes and what specific requests the user has with respect to obtaining particular music. The application needs thus a *profile of the user*. This profile should contain information about the preferences of the user with regard to particular music aspects (e.g. artist, year, label, title), as well as information about the whole music collection and the user's music playback behavior.

Third, the application should have *meta-data about music items*. For instance, the application should know the artist, title or release year of a song or album, it should know which songs are on a particular album as well as how many tracks there are on the album. Meta data is needed in order to reason which music items are liked or disliked by the user.

Last, the application needs information about places where to download the music items, e.g. information about *download sites on the Internet*.

Application Architecture

To come to an overall architecture for the music gathering application we've applied several composition principles. First, we've made a difference between *non-agent* and *agent components*. The non-agent components of our architecture reflex traditional components such as collections/databases and media (mp3) playing software components. The agent components reflects components that actively make decisions and whose behavior can be explained by adopting an intentional stance [7] by attributing believes, desires (goals) and intentions to them. A second composition principle is to use a *central agent* versus *support agents* in our architecture. The central agent tackles the problem "*what* music items to obtain" and creates application level goals, while the support agents provide the central agent with relevant information from the Internet and are responsible for the "*how* to obtain a particular music item" problem. Finally, we've used the *mirroring of external (Internet) resources* composition principle. This means that for every relevant information source on the Internet we've designed an agent that knows the protocols to obtain that information and that knows how to translate it into an internally specified format that is understood by the components of the architecture.

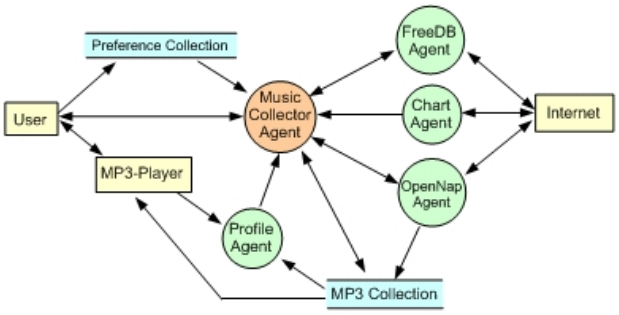

Figure 1: Proactive music gathering application architecture.

Figure 1 illustrates the different components of the music gathering application. The non-agent components are the Preference Collection, the MP3 Player and the MP3 Collection. These components are internally structured using traditional software engineering techniques. The agent components are:

- *Music Collector Agent*, a central agent who reasons about what music items to obtain.
- *OpenNap Agent*, a support agent that handles the problem of downloading MP3 files from OpenNap servers on the Internet.
- *Chart Agent*, a support agent that monitors particular Internet sites with hit chart information. When new chart information becomes available this agent parses the Internet site and sends new hit chart information to the Music Collector Agent.
- *Profile Agent*, a support agent that generates a profile of the user based on information about the user's mp3 collection and on the user's playback behavior.
- *FreeDB Agent*, a support agent that knows how to access the FreeDB Internet site [8] to obtain information about the tracks of an album.

The internal architecture of the individual agents is attuned to the problems they tackle and therefore each agent has a different internal architecture. The Music Collector Agent must make inferences about the music items the user likes. We have adopted a Belief Desire Intention (BDI) architecture [4; 16] for the Music Collector Agent. The roots of the BDI architecture can be traced back to the work of Bratman, Isreal & Pollack [4] who discussed the issue that an architecture for resource-bounded agents, such as our Music Collector Agent, should incorporate means-end reasoning, weighing of competing alternatives and an interaction mechanism for these two processes. In particular, the BDI architecture is aimed at constraining the amount of practical reasoning. We have used the BDI architecture because it is a flexible reasoning mechanism that is dedicated to operate in such a practical problem domain as the Internet.

The OpenNap Agent must effectively download MP3 files from OpenNap servers. Because OpenNap servers are highly uncertain, dynamic and non-episodic worlds the OpenNap Agent's architecture is based on reinforcement learning techniques [14]. Servers and users, for example, come and go in an unpredictable manner. Some users do not share files, others have a limit on the number of uploaders they serve and not every server shares the same set of files. Reinforcement learning provides mechanisms to learn a model of this uncertainty while acting in the problem domain. Subsequently, this model is used to determine the best actions that can be performed; that is, to select those servers and users that have a high probability of returning desirable results.

To incorporate DRM and payment models another dedicated agent would replace the OpenNap Agent. This new agent would have to implement the protocols needed to buy music from a particular source. Also, this new agent would be able to compare different internet sources and purchase the desired music from the cheapest source.

The task of the Chart Agent is relatively simple compared to the two previous agents: it periodically parses Internet sites (html documents) with hit chart information. It has a dedicated architecture for this purpose that consists of a sensor to retrieve the html document, a html parser to extract the relevant hit chart information and a database to store the information.

The Profile Agent's architecture is based on techniques to calculate statistics about the MP3 Collection and the user's playback behavior. For every music aspect, such as artist and genre, a histogram is calculated that contains information about how many songs in the user's music collection contain that specific aspect. The histogram calculates, for example, how many songs in the collections are from David Bowie and how many songs are Jazz. Furthermore the histogram calculates the relative preference for each of these aspects. These histograms are used by the Music Collector Agent to decide which music items to download.

Finally, the FreeDB Agent has, just like the Chart Agent, a dedicated architecture that implements the protocol to access the online FreeDB music database. FreeDB is an open source online database with music album metadata.

User Information

The music gathering application is complex and many problems may occur during the execution of the application. Let us consider the OpenNap agent as an example. OpenNap servers can be characterized as highly uncertain, dynamic and non-episodic worlds. Sockets are used to connect to an OpenNap server. However, it is unpredictable whether a particular OpenNap server will be available at a particular moment. Once a socket connection has been made, the OpenNap Agent has to log into the OpenNap server. Not all OpenNap servers allow everybody to log in (private OpenNap servers) and most servers have set restrictions such as the number of connected users and the number of files that these users have to share. If a client has logged in, it can start searching for files. The result of a search request depends on the content being shared by other. So, a search query returns a list of users that share the particular file. If the list contains items, the client can start requesting the other client to start a file transfer. File transfers may also lead to difficulties. Most clients restrict the number of uploads they serve and firewalls may render the transfer completely inoperative.

The user does not need to know all the details about the events that are generated by the application. When the user takes a quick look at his or her MP3 player he or she may only want to roughly know how well the music gathering progresses. If the progress is unsatisfactory the user may want to take actions to resolve the problem. In the next section the design of the user interface is discussed, in particular how it summarizes the information complexity described above.

INTERFACE DESIGN

The design of interface for mobile devices faces several challenges. First, the screen size is small compared to the size of standard computer screens. The popular iPaq, for example has only a 240x320 pixel screen. The screen resolution is just as low as on computer screens which makes it inconvenient to read longer passages of text or to identify small icons.

Second, the mobile context in which these devices are used differs considerably from standard PC workplaces. Mobile devices are usually held in one hand and the other performs the control actions. The devices are also used outside of the user's home or workplace.

Third, one can distinguish the various mobile devices by their input methods. Several devices use a small keyboard or keypad [12], while others use a pen [6; 11]. Also combinations of the two main input methods are available [9]. None of the input methods can yet compete with the efficiency and effectiveness of the standard PC mouse and keyboard. Hence, long writing tasks are usually done with help of the PC. However, the control of simpl graphical user interfaces (GUI) works rather well on pen-based devices.

Another input method for mobile devices is speech. Several mobile phones do already use voice dialing, which is a primitive form of a speech interface. In the future speech might become a more dominant input method for mobile devices, since it would allow the easy entry of text. One of the difficulties for speech-based interfaces is that the two partners can never be sure that the information transmitted will be perceived successfully and understood correctly. If the means to ensure intelligibility and interpretability of the messages do not result in a successful communication, the speakers have to resort to feedback. Dialogue control acts form the majority of this feedback [5]. The meaning of a certain message might also be amplified by employing gestures, body posture and emotional facial expressions. To be able to employ the full range of dialogue control acts and to amplify the meaning of a message with emotional expression the mobile device needs an anthropomorphic entity to execute facial expressions and gestures. This entity could employ the dialogue control acts simultaneously to the speech acts, so that the user has natural and constant feedback about the status of the information transmission. However, the noisy environments in which mobile devices are used, such as train stations and buses, pose a considerable challenge for speech systems. Current speech recognition software is not sophisticated enough to operate in these environments and therefore the speech input method was not used for the music gathering application.

Last, the complex information of the status of the music gathering should be available to the user at a glance, similar to checking the time on a watch. Given the small screen size and resolution any of the standard GUI elements, such as multiple progress bars and text logs, seem unsuitable. Therefore, the interface for the music gathering application is optimized to a screen size of 240x320, which is a standard size for current mobile devices. The interface is split into four tabs that correspond with the four steps the user has to perform to gather music.

Search tab

In the search tab the user enters the artist, album or song he or she would like to gather (see Figure 2).

Figure 2: The search tab.

The result of the search is displayed in form of a hierarchical tree structure, ordered by artist, album and song. If the user, for example, is looking for music of the band "Galaxy 500" then the result field will display the albums of this band and within these the songs that belong to each album. The user may now select any combination of albums and songs that he or she would like to gather.

Status tab

The status tab provides feedback on the current status of the music gathering (see Figure 3).

The numerous aspects of this status, such as the number of available servers, speed of the download and the availability of chart information are too complex to be visualized given the small screen size. Therefore a comic character face is used to provide a natural and instant feedback to the user. The character is based on eMuu, an embodied emotional character developed by Bartneck [2]. The character uses emotional facial expressions to communicate the status of the gathering application to the user. A simplified OCC emotion model [10] is used to map the numerous events and actions to emotional states and their intensities (see Figure 4).

The subsection chosen from the OCC model focuses on the well-being type, creating a character that is able to communicate its internal emotional state to the world. The well-being type emotions are mapped to a set of three different emotional expressions: happiness, anger and sadness. In short, all the positive events and actions will result in happiness, all the negative events will result in sadness and all the negative actions will result in anger. The distinction of what is an event and what is an action is based on accountability. It is impossible to blame a person for the failure in the internet, but if a specific person cancels the download of the user then the character has a person to be angry at. The intensity of each emotional state is based on certain variables as described in Table 1.

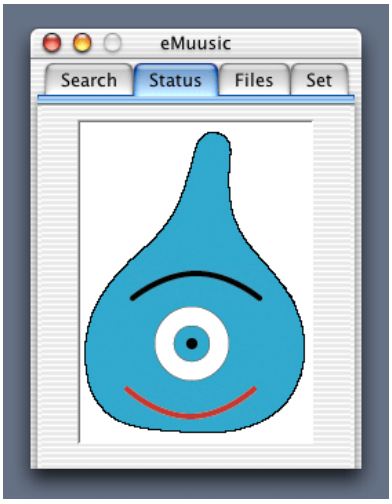

Figure 3: The status tab.

We have identified four events in our application that are relevant for the synthesis of emotions. First, a *NewChartInfo* event is generated whenever the Chart Agent has obtained new hit chart information from the Internet. New chart information makes the character happy. The second event is the *NewGoal* event. The Music Collector Agent generates this event when it has decided to obtain a new song or album. Creating new goals makes the character happy as well. The third event is the *NewOpenNapInfo* event. It is generated by the OpenNap Agent when new information about OpenNap servers has been found. Because this information increases the likelihood of obtaining songs, the character will be happy when this event occurs. Finally, the *SearchResult* event is the fourth event in our application that is relevant for generating emotions. The *SearchResult* event is generated by the OpenNap Agent after it has searched for users that share a particular song. When there are users sharing the song the character will be happy; otherwise it will become sad.

Besides events we have identified several actions of agents that are relevant for the synthesis of emotions. The user can be considered an agent and performs two kinds of actions. Either the user performs an *UserRequest* action to instruct the music gathering application to download a particular song or album, or the user performs a *CancelUserRequest* action to abort downloading a particular song or album. The character will become happy when the user requests to

download a song or album and it will become angry when the user cancels a request, especially when the application has almost completed the download.

Table 1: The intensity of events and actions.

Events (happy & sad)	**Variables to calculate intensity**
NewChartInfo	Probability of happening, number of new hits
NewGoal	Probability of happening, goal type
NewOpenNapInfo	Probability of happening, number of new OpenNap servers
SearchResult	Number of results
Actions (happy & anger)	**Variables to calculate intensity**
UserRequest	Last time user made a request, type of music item requested
CancelMusicItem	The progress status of the request
GetAlbumInformation	Probability of success, actual success or failure state of the action
ConnectToAnyServer	Probability of success, number of failed ConnectToSpecificServer actions
ConnectToSpecificServer	Probability of success, last time a successful connection was made
DownloadFromAnyuser	Probability of success, number of failed DownloadFromSpecificUser actions
DownloadFromSpecificUser	Probability of success, last time a successful download occurred
DownloadedSomeBytes	Probability of success
DownloadAbortedByPeer	Probability of happening, progress status of download

The FreeDB Agent performs the *GetAlbumInformation* action when the Music Collector Agent requests information about an album. When the action succeeds and information is found about an album, the character will become happy. Otherwise the character becomes angry. The OpenNap Agent performs five actions. The *ConnectToSpecificServer* action is part of the *ConnectToAnyServer* action. Both actions are used to connect to an OpenNap server. The *DownloadSomeBytes* actions is part of the *DownloadFromSpecificUser* action, which in itself is part of the *DownloadFromAnyUser* action. All three actions are performed when the OpenNap Agent wants to download a song. Finally, the *DownloadAbortedByPeer* action is performed by the peer (user) from which the OpenNap Agent is downloading a file. This action makes the character angry.

The emotional intensity of the events and actions is calculated by using relevant variables that are listed in Table 1. The intensity of a *NewChartInfo* event, for example, is based on the probability of this event to happen and the number of new hits that has been retrieved. The character will by happier in case the probability of the *NewChartEvent* is low and the number of new hits is large. The intensity of the *CancelMusicItem* action is based on the progress of the request. The more effort, in terms of download completion, has been made to fulfill the request the angrier the character will be if the request is canceled. Finally, *ConnectToAnyServer* action is a composed of several *ConnectToSpecificServer* actions. In order to connect to a server the application has to try several specific servers. The intensity of the *ConnectToAnyServer* action depends on how quickly the application can normally connect to a server (probability of success) and the number of times it had to try a specific server before it had a connection.

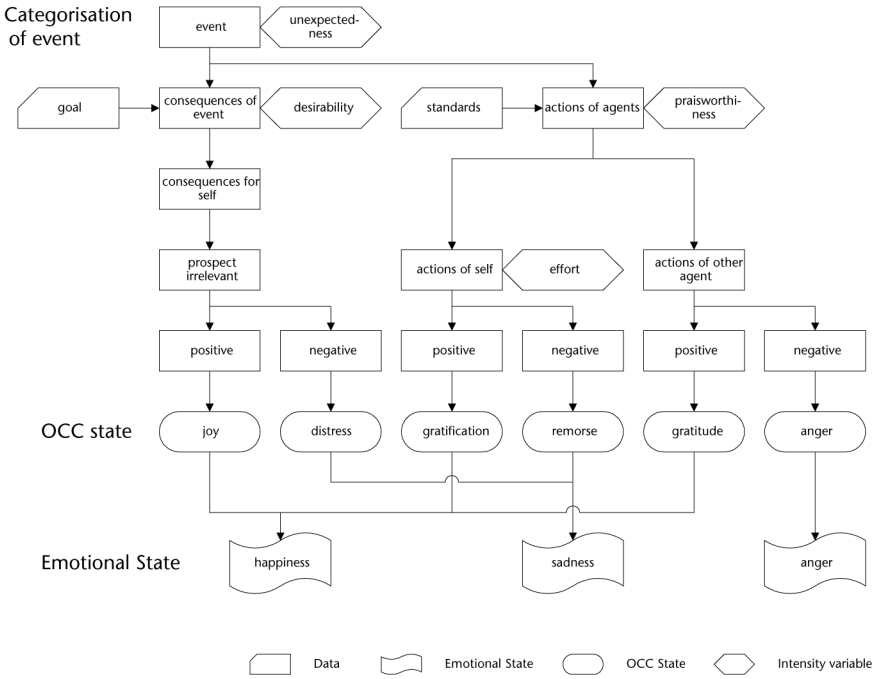

Figure 4: The simplified OCC model of the music gathering application.

Files tab

The files tab displays the files that are currently in the download directory of the application. All songs that the user downloaded, including the ones being currently processed, are shown in an hierarchy tree. This tree structure allows the user to select any combination of artist, album and songs and to perform actions on the selection. The user may, for example, listen to a song to check its correctness and quality or retry downloading songs that have not been completely downloaded due to an error. Moreover, the user can delete songs of any artist or album or move them to the music library of his or her jukebox application.

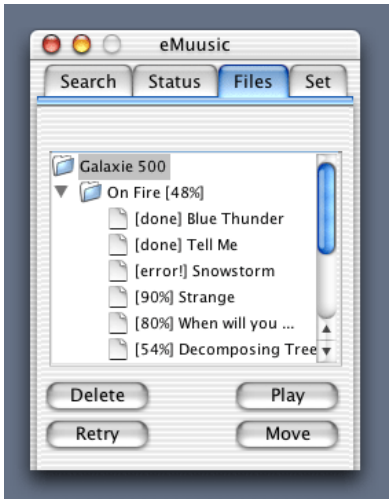

Figure 5: The files tab.

Settings tab

In the settings tab the user can adjust the system preferences. The proactive music gathering can be switched on an off, the user's music profile can be edited and the desired music quality for the downloaded songs can be selected from a predefined list.

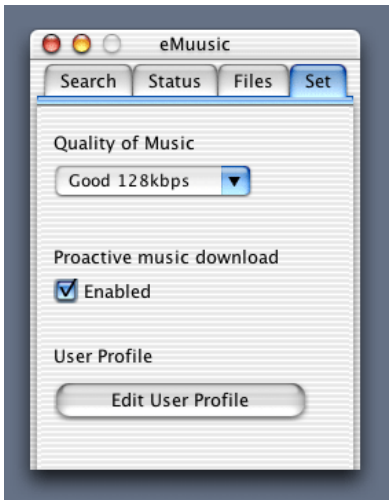

Figure 6: The settings tab.

EVALUATION

Several tests were performed to evaluate the application's performance. The performance was highly satisfactory. The application was able to combine the user's profile with real time hit chart information from the Internet to search and download songs from OpenNap servers. Because the OpenNap Agent incorporated reinforcement learning techniques, the performance of connecting to servers and downloading from users increased over time. The participants of an informal usability evaluation were particularly satisfied with the possibility to download complete albums, because the current peer-to-peer network clients only allow the download of single songs.

A typical problem of adaptive systems, like the music gathering application, is the creation of the initial user profile. This problem is often referred to as the "bootstrap problem". Our application is able to generate a good initial user profile by analyzing the metadata of the user's existing music collection. A first test revealed that many MP3 users have a collection of at least 300 MP3 files, of which about half contain usable metadata. The initial user profile generated from these MP3 files turned out to be quite satisfying and many users commented that the profile matched their preferences. A more formal user test is in preparation.

CONCLUSION

We described a music gathering application with an emotional interface character for mobile Internet-enabled MP3 players. The storage capacity and jukebox functionality of mobile MP3 players are not a critical issues anymore. However, the acquisition of new content remains a problem. We proposed a proactive music gathering application that automatically obtains music from the Internet based on the user's profile. The application's architecture consists of several agent and non-agent software components.

The numerous aspects of the status of the proactive music gathering application, such as the number of available servers, speed of the download and the availability of chart information are too complex to be visualized given the small screen size of a mobile MP3 player. Therefore a comic character face based on eMuu [2] is used to provide a natural and instant feedback to the user. We have implemented the user interface by using four tabs that resembles the steps the user has to perform to gather music.

Further research

Speech technology could possibly improve the usability of the music gathering application. The user would be able to enter his or her search query, select actions and check on the status of the gathering by using speech. The screen character could provide natural feedback of the status of the dialogue by providing conversational and emotional facial expressions.

The music gathering application should cooperate closely with the jukebox application on the mobile device. The user should be able to use a "gather more" action in the jukebox

to gather more music of a particular artist that the user was just listening to. The music gathering application should be able to quickly hand over its downloaded music to the jukebox.

A key component for the success of consumer electronics is personalization. Users of mobile phones even pay for customized icons and melodies. The music gathering application can be extended to include several characters and the possibility to download even more through the internet.

The character could even become the personal DJ for the user. Supported by proactively downloaded music it would generate personalized playlists that the jukebox application would use to create a radio program for the user. The personal DJ could be context aware and generate activity-attuned playlists for birthdays, romantic evenings or parties. The downloaded content is not limited to music, but could also include the latest stock market information or traffic news. The personal DJ could also help improving the accuracy of the application's user profile by engaging the user in a game-like setting. In a playful fashion the application could receive direct feedback on the user's music preferences.

Currently, the Chart Agent and the FreeDB Agent parse html documents from the Internet to obtain the desired hit chart and music album information. However, the layout of these internet pages and hence the html code change without notice which made it necessary to repeatedly adjust the agent's html parsers. This is a common problem for internet applications and there are three solutions available. First, the parser can be changed every time the html document changes. We used this simple solution for our html parser. Second, the parser could be equipped with additional intelligence that enables it to handle arbitrary changes in the document structure. Advanced methods from the area of artificial intelligence would be necessary, which are beyond the scope of this study. Third, the relevant information in the html page can be labeled with metadata using the Extendible Markup Language (XML) or the Resource Description Framework (RDF) standard. First internet music databases exist that use the RDF standard [17] to publish their metadata [13].

REFERENCES

1. Apple. (2002). *iPod*, from http://www.apple.com/ipod/
2. Bartneck, C. (2002). *eMuu - an embodied emotional character for the ambient intelligent home.* Unpublished Ph.D. thesis, Eindhoven University of Technology, Eindhoven.
3. Brandenburg, K. S., G. (1994). The ISO/MPEG-1 Audio Codec: A Generic Standard for Coding of High Quality Digital Audio. *JAES, 42*, 780-792.
4. Bratman, M.E. , Isreal, D.J., Pollack, M.E. (1988). Plans and Resource-Bounded Practical Reasoning. *Computational Intelligence, 4*(4), 349-355.
5. Bunt, H. C. (1989). Information dialogues as Communicative Action in Relation to partner Modelling and Information Processing. In F. N. Taylor & D. G. Bouwhuis (Eds.), *The Structure of Multimodal Dialogue* (pp. 47-73). Amsterdam: Elsevier Science Publishers.
6. Compaq. (2002). *iPaq*, from http://www.compaq.com/products/handhelds/pocketpc/
7. Dennett, D. C. (1981). Intentional Systems. In J. Haugeland (Ed.), *Mind Design* (pp. 220-242): Bradford Books.
8. FreeDB. (2002). *FreeDB*, from http://www.freedb.org
9. Nokia. (2002). *Nokia Communicator 9290*, from http://www.nokiausa.com/communicator
10. Ortony, A., Clore, G., & Collins, A. (1988). *The Cognitive Structure of Emotions*. Cambridge: Cambridge University Press.
11. Palm. (2002). *PalmPilot*, from http://www.palm.com/
12. Psion. (2002). *Psion Revo*, from http://www.psionusa.com/PersonalMobility/Revo/index.html
13. Swarts, A.(2002). MusicBrainz: A Semantic Web Service, *IEEE Intelligent Systems*, 76-77
14. Sutton, R. S. B., A.G. (1998). *Reinforcement Learning - An Introduction*. Cambridge: MIT Press.
15. Toshiba. (2002). *Tosiba introduces world's highest capacity 1.8-inch Hard Disk Drives*, from http://www.toshiba.com/taissdd/news/press44.shtml
16. Wooldridge, M. (2002). *Multi-Agent System*. Hoboken, NJ: John Wiley & Sons.
17. W3C, (2002), *RDF Standard*, from http://www.w3c.org/RDF/