YapApp
Final report

Arnoud Andeweg	Sander Latour	Haska Steltenpohl	Franklin Widjaja
6211097	5743044	5909821	6390382

December 21, 2010

Abstract

We designed a knowledge media application based on a user scenario, where the user was specifically looking for quiet and new business places to work. The application, called YapApp, gathers data from the user, GPS, the Internet, the user's friends, dataloggers and businesses. This data is processed in YapApp, to determine which data is useful. All processed data is represented in an OWL/RDF format, to provide a reasoning logic that can run queries in our ontology and filter the available data. When a search is performed, categories such as "quiet" or "new" can be included to search for these kind of businesses. The application has a self learning system that keeps track of the user's preferences. The system is designed to encourage the user to contribute to the system. The user can give feedback about the recommendations given by the application so it will enhance the learning algorithm of the system.

Contents

1 Introduction

Our goal was to design a platform independent knowledge media system that uses multiple existing location centric API feeds and allows users to contribute audio, text or photo information to the system based specifically on their location. Users should be able to retrieve intelligent information from the system about any area. Our design of the application was based on a scenario, in which a user profile is described. The subject in this scenario is Luc, a 27 year old French writer. He came to Amsterdam to find interesting shops to explore and to find a peaceful place to relax and write. He writes for an entertainment blog in Paris and is a smartphone user. Luc has been to Amsterdam a few times and is looking online about what is new in Amsterdam. He also wants real time information about any business he encounters. Based on Luc's preferences the system recommends types of businesses.

We chose 6 shops from "de 9 straatjes" in Amsterdam as the shops that would transfer comments about their business: a jeans shop, a second hand item shop, a cocktail bar, a coffee shop, a jeweller and a bistro.

To summarize, our objective was to design an application based on Luc's preferences. The application should tell him where to go when he is looking for a quiet place to write at and give information about any business he is interested in. In the following sections we will describe how this application is designed. In section 2 we describe the application design concepts. It is an overview of all processes and features that we used to design the application. In section 3 we describe the data sources, how the data is collected and the process of transforming that data into useful information. At the end of this section we describe how this information is displayed to the user. In section 4 we describe how the businesses can operate YapApp to appeal to their clients.

2 Application design and concepts

The application will allow users to quickly understand where they can find the type of business they are looking for; in real time based on real time information and by using a mobile phone. In this section we describe the concepts that we used to design the application.

2.1 Information gathering

The application gathers information from various sources:

- Social networking sites (facebook, twitter, YapApp, Hyves), to learn about the user's preferences and keep track of the user's friends.
- Other websites such as weather information sites.
- User of the application (feedback to the application)
- GPS, to keep track of the locations of the user and the user's friends.
- Various kinds of data loggers to measure the temperature, make images inside the business to measure the amount of people and nois.

2.2 Displayed Information

- Temperature: the temperature outside and inside a business are both measured and displayed, so the user instantly sees the difference between the inside and outside temperature. When it is very hot or cold outside, this feature could motivate the user to go to a more comfortable business location.
- Quietness: the user is interested in quiet places. The noise level is measured and based on this value the user can decide what business locations are comfortable enough to work at.

- Music: the music that is currently playing is displayed. Users can decide to go to business locations that play music that is more appealing to them.
- Comments: businesses create comments (advertisement) to attract costumers. The comments consist of a short description about their products and some of them have photographs included to give an impression of their business.

2.3 User contribution

The system is designed to encourage the user to contribute to the system. The user has the option to quickly respond to recommendations given by the application that will enhance it's learning system. Recommendations for other users can be given as well by textual input or placing images of the business place.

2.4 Information searching

A search is performed by textual input. A range of features can also be selected that will be included in the search for a business, such as a "quietness" feature for businesses that have little noise, and a "new" feature, for businesses that are new.

2.5 Self learning system

The application has a self learning system that keeps track of the user's preferences. After the system has made a recommendation, the user can manually give feedback on that recommendation, which enables it to learn about the user's preferences. Feedback is automatically registered as well when recommendations are ignored by the user.

2.6 Processing data: OWL

All processed data is represented in an OWL/RDF format. The owl format provides a reasoning logic that can run queries in our ontology and filter the available data.

3 YapApp, a users perspective

3.1 Data Sources and processes

In this section we will describe the data sources used in our model (figure 1) For each data source the process of collecting data and the process of transforming data into a form (information/knowledge) that YapApp can use for reasoning, will be described. The processed information can be mapped onto the wishes (through learning or explicitly stated) of the YapApp user. This process of mapping wishes will be discussed in section 3.2. The reason why these data sources were chosen are described in the previous section.

3.1.1 Temperature

With the use of dataloggers the inside and outside temperature will be logged. Every 60 seconds data is collected and is sent to YapApp. The metric is in degrees Celcius. The data (number of degrees) will be stored as a number in such a way that there is a reference to time and place (company).

3.1.2 Quietness (sound level and number of people)

Quietness will be a combination of sound level and crowdedness (number of people) present in a shop/business. The combination of these two is processed and represented as the Unique Quietness Score (UQS).

The sound level is collected and measured by a decibel sensor. The decibel value is sent to YapApp every 60 seconds. The crowdedness will be measured by taking pictures. Every company that is in the YapApp network has a webcam installed. Every 15 minutes a picture is taken and send to the YapApp server.

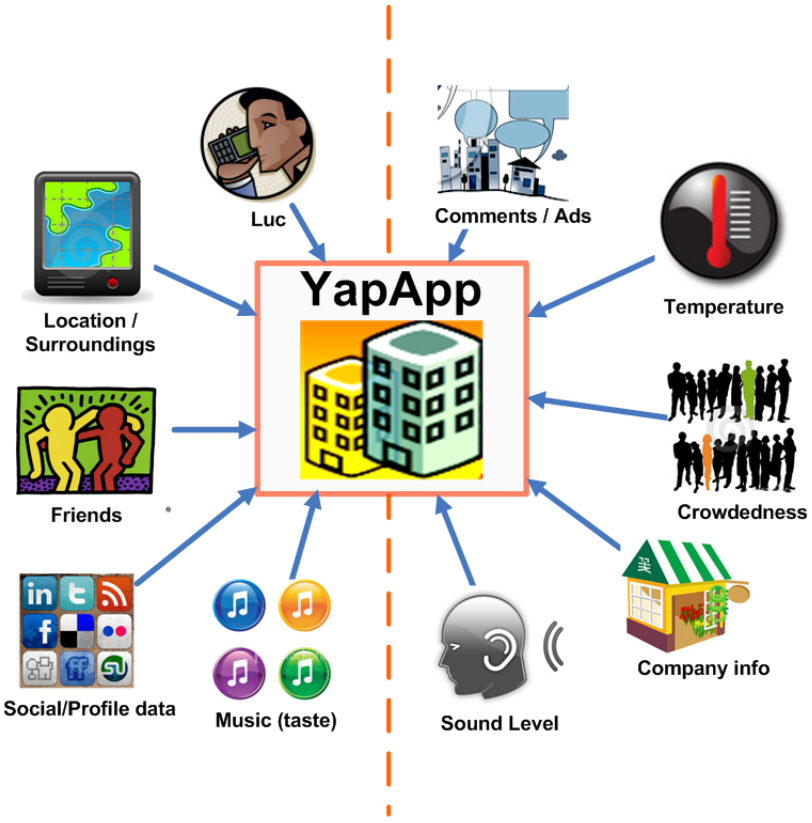

Figure 1: Data sources used by YapApp. On the left is the user data and on the right the company data.

The number of decibels is stored in YapApp and is transformed in a value ranging from one to ten. A value of one is equal to 0 dB (treshold of hearing) and a value of ten is equal to 120 dB (for example a rock concert and close to the treshold of pain). The measurement of crowdedness by taking pictures is described in the following steps:

1. Every company has a webcam installed
2. Every 15 minutes one picture is taken
3. Compare picture with null picture
4. Make corrections for the brightness scale
5. Learning system that matches amount of people with differences in picture
6. Scale (one to ten)

Together, the sound level and the number of people form the Unique Quietness Score (UQS) by taking the weight average of both scores. The weight averages are learned by the system. Where UQS=0 is extremely quiet (sound and people) and 10 is not.

3.1.3 Music (now playing)

Companies often rely on automated sound systems to provide music for their customers. We suggest an ontology based integration with such systems. The music that is playing in the company will be linked with Musicbrainz[4]. Musicbrainz is an music database that is also accessible in rdf triples. It is based on an ontology as seen in figure 2.

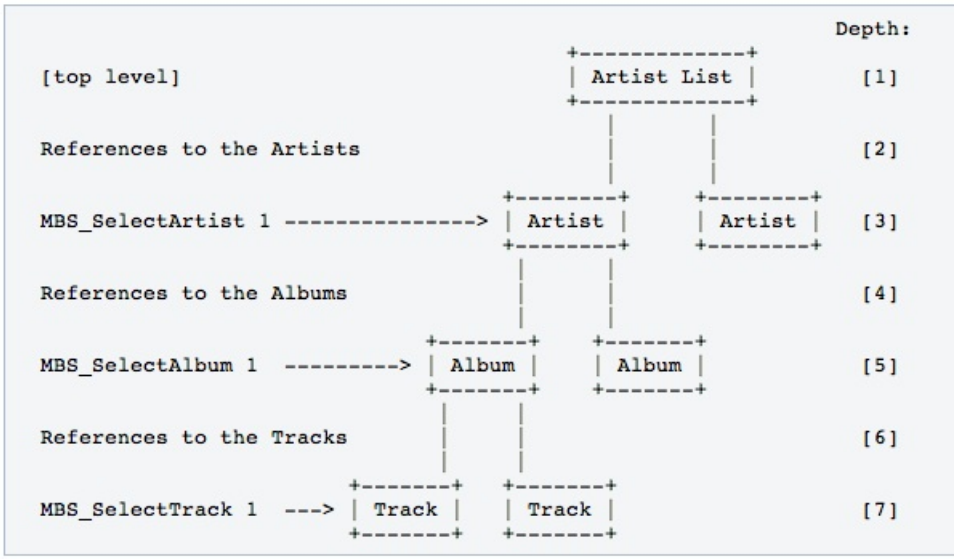

Figure 2: Musicbrainz ontology.

When a song is being played in the system, it will automatically match the song with the musicbrainz triples. This will tell the system what the length of the song is, the title of the song etc. It will also tell us which genre the song belongs to (rap, country, classic). When all this information is gathered we can match this to the user preferences (described below).

The reason that we decided to match the music preference to a music ontology is because it makes it easy to suggest music similar to the music that the users listen to. When a user likes Michael Jackson's music for instance, it is probable that he likes pop as a music style.

3.1.4 Music (taste of user)

To give the application the ability to recommend businesses based on music taste, we have to identify the music taste of the user first. There are multiple ways to accomplish this depending on what the user uses to listen to music. The supported resources for the application will be described in this section.

Last Fm People use last FM to keep track of their music taste by using a so-called "scrobbler". This "scrobbler" is installed on the user's device and sends the played music to the last FM server. Last FM records this and matches the music taste with others in order to recommend new music to their users. This information is available through the api of last FM. To get this information the application asks the user to log in to last FM. Next, the application retrieves the user's taste of music through the following functions within the last FM api;

- user.getLovedTracks (http://www.last.fm/api/show?service=329, this will get the most popular tracks)
- user.getPlaylist (http://www.last.fm/api/show?service=313, this will get the playlist of the user)
- user.getRecommendedArtists (http://www.last.fm/api/show?service=388, this will collect the best matches according to last fm)

The collected information is matched to our OWL (described above) and recorded in the respective user database (rdf). This is in the form of triples that link the user to a certain music taste. Another option is that the user searches last FM for friends that are also YapApp users.

Music library The devices that the users use are a good source to analyse music taste. The music that people decide to put on their mobile devices is often the music they like to listen to. It might not be a comprehensive reflection of the music taste from the user but it will certainly tell us some information about the sort of music that the user enjoys listening to. To analyse the music files in the different operating systems we need to know how to access this data. This is done in different ways for each platform.

On the Android platform the necessary data can be acquired with the public static final class "MediaStore.Audio". On the iphone the music data can be acquired with the class "MusicTableViewController.h".

3.1.5 Comments/Ads (Companies)

The companies can store comments or advertisements in the YapApp system. These comments will be shown in the virtual representation of the shop (See section 3.4 for a visual representation). These comments can be inserted via an account on the YapApp website. Different comments for different targets are possible (man/female, age groups, music style, friend of someone who is inside, etc.).

3.1.6 Internet

The Internet is a source of additional information that YapApp can use in its profiling and learning of the user or for its reasoning. The location or the weather can be relevant in finding the best answer for Luc's query. The following list of sources is not the complete list, but serve as good input for the reasoning and profiling processes of YapApp:

- Friends:
  It can be useful to know where your friends are or have been. Friends can be found trough the YapApp network, the phones contact list, or from other social networks.
- Social/ profile data:
  We consider this as a bonus as YapApp is designed to learn and profile by itself. If additional information is available than it can be used. Examples of external social sources are Facebook, Twitter, Hyves and last FM.
- Weather:
  Weather information is used in the reasoning. It can be accessed by numerous sites and services. The Weather.com API is an example of such a service. When a user is looking for a restaurant and is by foot, then the current type of weather should be considered as well by the system.
- Locatix'on/Surroundings:
  YapApp is always 'aware' of its location and surroundings in order to give (good) suggestions and to give directions from the current location. In order to be 'aware' the YapApp uses the GPS-feature (current location) of the mobile device in combination with Google Maps (surroundings). This information is used in the reasoning, but also for giving directions, or showing the distance (minutes/km) to a company from the current location.

3.1.7 Business information

YapApp displays business information such as: address, telephone number, the type of business, and menu (if applicable) to assist the user in choosing or contacting the business. This information is send by the companies themselves. They must insert this information via their account on the YapApp site. Alternatively, this information could be put on the business's website in an agreed format so that YapApp can collect the information from it.

3.2 From query to result

In the previous sections we described the collection of information. Gathering all these sources is ofcourse only usefull if you can somehow search and organise it. The way YapApp makes this possible is depicted in figure 3. The details of this process will be discussed by answering the following questions.

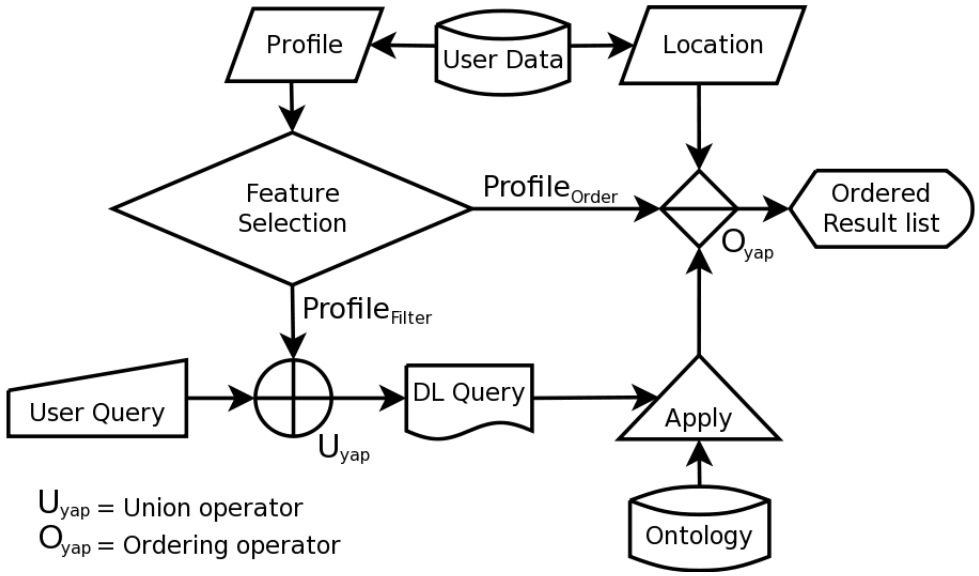

Figure 3: The flow from query to result

- How is all that information organised?
- How can you query all these sources?
- How much do you need to specify in your query?
- How does the system know what really is important?
- How can we filter out potential goods suggestions?
- How can we order those suggestions for you?

3.2.1 How is all that information organised?

All companies are stored as rdf resources in our ontology. These resources are described by rdf triples that specify the values of the above mentioned sources. Surely because these bits of information are not all static this storage needs to be constantly updated. Every time one of the sources sends a value towards the YapApp central server, the YapApp server will alter that part of the ontology. This way the ontology will always supply a representation of the current situation whenever it is queried.

Besides companies, also products are stored and linked to the companies using *sell* relations. The benefit of having the products in our hierarchical storage is that you can query things like restaurants that sell vegetarian food and receive results containing restaurants that for example sell a quiche.

Having an hierarchical storage is not the only advantage of using an ontology instead of using for example a statistical method. Perhaps the biggest advantage is that you store your knowledge in a declarative way enabeling you to not only to generalize more easily but also to explain why certain results apply to a query.

3.2.2 How can you query all these sources?

The ontology can be queried with description logic. One way of achieving this is by formulating queries in the Manchester OWL Syntax[2] and let them be evaluated by a reasoner such as Pellet[1] or FaCT[3]. The features of the companies such as the type of music they are playing or the type of food they are serving can all be represented in discrete formulas such as ***sell some Vegetarian_Food*** or ***isPlaying some (Jazz or Soul)***

which can be easily combined using conjunctions. These logical representations of features can be inserted in the query by the user by activating te feature buttons in the search screen (See also 3.4).

3.2.3 How much do you need to specify in your query?

In order to give the user quick and easy access to all this information you don't want to force the user to specify everything every single time. You would want the system to know what the user generally cares about and only asks the user to specify what he is specifically looking for right now. In practise this means that the query constructed by the user is enriched with what we already know. That way the user will only would have to specify that he wants to eat something and we will add that it has to be a vegetarian dish. We will store this knowledge about our user in a user profile which is defined as follows.

User Profile The generalization of a selection of the queries that were presented to the system before. The generalization is achieved by finding the most specialized expression that will contain every selected value, which is done by using the hierarchy in case of discrete values. Continuous values will be generalized in the obvious manner. Not every value from every query is used for the obvious reason that even these queries contain noise, for example if the user searches for a club that plays rock music just because he is together with someone that enjoys such music but does not like it himself. You therefore want to make a selection of what is and what isn't indicative. That selection can be made by setting a threshold on the frequency of the value which will outrule the outliers. This threshold is dependent on the expected variation of that feature, which has a negative relation to how important that feature is to the user. If a feature is very important than it will be very likely that almost every value is indicative, and therefore a lower threshold can be used.

Union operator The combination of the user query and the user profile can be described as a special union operator (U_{yap}). This operator is exactly like a normal union operator except that it lets the user query overwrites the user profile. In other words, if a feature is described in the user query then that value will be used regardless whether that feature has the same value in the user profile. To define this operator more formally, you can view the user query and user profile as collections of feature-value pairs and the union of these collections would be every pair from the user query collection combined with every pair that describes a feature not yet described in the user query collection. Or: $U_{yap}(A, B) = A + (B - A)$.

3.2.4 How does the system know what really is important?

Ofcourse the notion of importance is not predetermined and we therefore need to learn which features are more important to the user than others. We do that by applying feature selection to a dataset gathered from earlier classifications. These classifications are positive or negative feedback to suggestions and therefore are classifications of those collections of features. It is our goal to be able to predict as good as possible which suggestions would fit the request best, but that fit is not equaly determined by every feature. One feature is just more important than the other. In other words, one feature is a better indicator of the future classification than the other. Feature selection can find these indicative features and split between the an arbitrary amount of important features and the rest. Given the fact that feature selection requires a dataset to apply the machine learning techniques on, it is clear that you need some sort of initial dataset in order to avoid having a long startup period where the systems has to learn everything. This initial dataset can be constructed by a test group causing the system to start with knowledge of what an avarage person thinks is important and over time will evolve to have knowledge over what the user thinks is important.

Breaking the habit A problem with this approach is that it will never suggest something that wouldn't normally fit into the user's profile. But the user's preferences may change rapidly over time and so do trends. It is therefore important that the system can suggests something outsite of the user's normal circle from time to time. This can be based on suggestions presented by the YapApp to friends of the user or based on the suggestions and feedback of people with profiles similar to that of the user.

3.2.5 How can we filter out potential goods suggestions?

We can use that split of our feature space to select the features that we'll use in our filter. It goes without saying that if a company has a different value then is defined in the enriched query on a unimportant feature it is not necessarily a bad suggestion. So we use the best features from the feature space to filter to results by running the description logic query consisting of the conjunction of the relevant query parts as was described earlier. These results are all potentionally good suggestions.

3.2.6 How can we order those suggestions for you?

As a user however you might want to have a bit more information in order to be able to discriminate between the various options. We therefore want to order this list using the other features in the feature space, assigning points to suggestions for every feature that matches the enriched query. Added to those points are credits for being in the near vicinity. Further using that score to order the list. This processed is called the ordering operator in figure 3.

3.3 Feedback

3.3.1 Passive feedback

The user can provide passive feedback to the system by its actions, like reading more about a company and actually stepping through the door. These actions are registered by the system as an agreement of Luc with the choice of the system. Opposite signals are send when a result is skipped and a result that was ranked lower is being checked, etc. This feedback is stored and used when the user does not provide active feedback.

3.3.2 Active feedback

The user can explicitly say that he liked or disliked the suggestion in the user interface. The active feedback will overwrite any derived passive feedback. It is very probable that the user will use the feedback system as a conditioning tool, and only respond when a suggestion was either very good or very bad. When the user does not have an outspoken opinion about a restaurant he will let the system decide what to do with this suggestion using the passive feedback.

3.4 User Interface

In this section we describe the user interface, what can be adjusted in the system and how it is displayed. The main user interface flow is described but not the user settings screen. In the user settings the user can enter their account credentials to set up the application with last FM and other social applications, it is also possible to set the notifications and user preferences manually.

Home screen
On the home screen, the top 5 matches are shown. These suggestions depend on the location and the profile of the user. On this screen the user has the option to tap on maps, which will result in a Google street-view map with comments from the businesses. Using the settings button the user can adjust account settings, notifications and override profile data (e.g. set preference to vegetarian etc.). The feedback button results in a context sensitive feedback screen. The user can give feedback about the company or place based on the location of the user. See also the result screen.

Searching in YapApp

Search Screen
The search button takes the user to the next screen where the user can specify the companies he is looking for. The user can construct his search query using keywords and features. Features can be turned on by tapping on them, which will adjust the query (See also 3.2). Selected features are more bright than unselected features. The list of features is ordered by importance (See also 3.2.4) except for the first eight features. They are on fixed positions for consistency, but can be changed by dragging them.

Selecting the category
One of the features is the category a company should be in. When the user clicks on this feature a popup will show which allows him to quickly search through the category hierarchy. The most used categories are displayed on top. These categories relate to the hierarchy in the ontology (See also 3.2.1).

The query in its visual form
When the user is done specifying his requirements he can see a visual representation of the internal query on the screen. It is in that sense a WYSIWYG (*What You See Is What You Get*) interface. The search process can be finished by tapping the magnifying glass to execute the query.

Result Screen

This takes us to the result screen which shows the suggested companies. The best suggestion is showed in more detail. You can inmediately ask directions by tapping on the map. YapApp also shows a picture and some of the features that gave this company the highest ranking. The other suggestions are showed in less detail. You can up/down-vote each suggestion which will give feedback to the system (See also 3.3). With the "←" (back) button the user can go back to adjust the search query. This can be done at any stage of the user interface. You can also start a new search right away by tapping on the search button again.

This screen is kept similar to the home screen to make it easier to learn the interface. It therefore also offers the same functionality to give instant feedback, change settings or search in the vicinity of the user.

Company information in YapApp

Company Home Tab

When the user taps on one of the suggestions the company information screen is shown. This screen contains three tabs for Home, Advanced and Ratings. This screen also has buttons to give feedback, take a picture or leave a comment. The Home tab shows the user a list of the last company comments in the chronological order, the available features, an option of getting directions and a category icon.

Company Advanced Tab

Behind the Advanced tap we find more detailed information about the company. When the company is from the category "restaurant" it will show the menu, what kitchen it is etc. For any type of company it will show the opening times, a detailed explanation of the features, the address etc..

Company Ratings Tab

When the user taps on the Ratings tab the user sees the comments that other users wrote about the company. There is also the option to rate the company on a scale of one to five. We decided not to show these comments earlier in the user interface since we want to put the emphasis on the company comments. By showing these earlier in the interface it gives the companies more incentive to write comments and keep these up to date. It also makes it easier for companies to differentiate themselves from other companies.

4 YapApp, a company's perspective

There are two sides to the YapApp, one is in the hands of the users on the street who use the application to find businesses of their preference, and the other is operated by the businesses. The businesses use the system to write comments, update their profile and handle the "social democracy". This "social democracy" feature we suggest for the business site YapApp is the ability to see what type of YapApp users are in the neighbourhood. When this information is available to the business they can adjust their shopping environment to match those users.

4.1 Data

The data needed to enable "social democracy" come from the users. The users share their geographic location and their profiles with the YapApp server. Within these profiles all the preferences are stored from a specific person, the kind of music they like, what they prefer to eat, and all the other information described in section 3.1.
Additional data that is needed come from the businesses themselves. The system needs to know what type of customer services the business offers. For instance, if the business plays music in their shop, the data that is needed is: what music is playing now and in the near future. The same goes for the room temperature of the business, the amount of people in the business, the UQS (universal quietness score),the inventory of the shop and the items on sale.

4.2 Processes

Having all this data, the system can register how many YapApp users are in the neighbourhood and are coming on their way (we can think here in a range of one kilometre). If users are in this range, the system can analyse what the best configuration is in order to attract the most customers (YapApp users). The system can calculate the music style that will satisfy the majority of the surrounding customers, making the company automatically a better suggestion for those users which will make it show up more in the result list. These changes are administrated by the YapApp company software or by the business themselves. An automated system will always make sure the business is in par with the majority of the YapApp users in the neighbourhood. In the case of human control, the agent will suggest the optimum configuration. A colour graph showing the growth in customer attraction will persuade the business side user to go for the optimum condition.

The changes made by the system in the business environment will always confirm to a certain standard range the business prefers to work with. This means that if a business is a skater shop with boards, low hanging pants etc. the music style will never change to classical, the music loudness will always be in the high ranges and the light colour will match or complement the overall "look" of the store. Within those ranges the agent is free to make the necessary changes to meet the demands of most customers.

4.3 User interface

Color-coding is an effective way of signalling the state of the system. When the configuration of the system is set so that it does not attract any of the YapApp users in the neighbourhood, it will show a red background. The more users the business attracts, the more green the screen will be. To prevent the system from being too intrusive to the people in the shop, no notifications are given. It is completely up the company itself to check its status.

The system can be minimalized to a small screen or a simple indicator on top or bottom of the computer. This last option would only show the user the colour coding of the system state (preferable green), clicking on it will result in opening the actual program.

5 Discussion

One of our design concepts was that users should be able to use the application without configuring any preferences. The application uses a machine-learning algorithm that learns through feedback about the users preferences. The downside is that the user needs to use the application intensively before good recommendations can be made. Users also have to be made aware that in order to get good recommendations, their feedback is essential. Even when the user manually sets preferences, feedback is still required to keep the application in par with the users needs. Some preferences, such as UQS, are hard to learn. The UQS may vary for different type of companies, users may want a silent coffee place but want loud music when shopping for clothes. Therefore we suggest a category dependent classification of UQS, what UQS a user prefers for a clothing store is not necessary true for other company categories. The combination of a learning algorithm and an OWL ontology is that the application is able to learn these settings and easily store these in rdf triples. These triples, based on the ontology, are the bases for communicating the information with the user. It is where the application gets its information from. It also helps the application suggest similar things to the user by simply comparing user ontologies.

An issue that still remains is privacy. To form a profile for a user the application asks access to the users social networks. Because of these privacy concerns, users might be unsusceptible to sharing their social network information with third party applications such as YapApp. Information is also gathered from users that do not use the application. Businesses gather data that is send to the system such as the amount of noise that their customers produce, and the location they are at. Images are taken from customers to determine the quietness in a business place. YapApp users themselves can also take pictures from the company and upload these to the server. Other privacy concerns are friend's locations, imagine a situation where a YapApp user buys lingerie for his mistress and the users wife is suggested to go to the same store because her husband went there. These situations are unwanted for the user and for YapApp, users therefore need to be made aware of these risks. A way to make users aware of these risks is to automatically turn this feature off until the user manually turns it on, from then on the user shares his or her visited companies with his or her friends.

References

[1] Clark and Parsia. Pellet: The Leading OWL 2 Reasoner for Java. `http://clarkparsia.com/pellet`. (Accessed 21 December 2010).

[2] Nick Drummond. The Manchester OWL Syntax. `http://www.co-ode.org/resources/reference/manchester_syntax/`. (Accessed 21 December 2010).

[3] Ian Horrocks. The FaCT System. `http://www.cs.man.ac.uk/~horrocks/FaCT/`. (Accessed 21 December 2010).

[4] MusicBrainz. RDF documentation. `http://musicbrainz.org/doc/RDF`. (Accessed 21 December 2010).