

# Context-aware Recommendations on Mobile Services: The m:Ciudad Approach

Andreas Emrich, Alexandra Chapko, Dirk Werth

Institute for Information Systems (IWi) at the  
German Research Center for Artificial Intelligence (DFKI)  
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany  
{andreas.emrich|alexandra.chapko|dirk.werth}@iwi.dfki.de

**Abstract.** The European FP7 research project m:Ciudad - a metropolis of ubiquitous services - aims at the empowerment of users to create services on mobile terminals. The project demonstrates various scenarios in which users either act as creator of services or interact with the system to search for services or service construction components. The search and recommendation process in the system facilitates retrieval of related entities and shows the results to the users according to their preference and contextual information. The paper demonstrates an approach to integrate contextual information with other search attributes to enable efficient service retrieval and recommendation in mobile user and application scenarios. The specifics of a mobile environment are taken into consideration and are reflected in the design of the m:Ciudad Search and Recommendation Engine. We discuss how context-awareness and proactivity can be implemented and utilised for mobile services.

**Keywords:** recommendations, mobile services, proactivity, m:Ciudad, user-generated services

## 1 Introduction

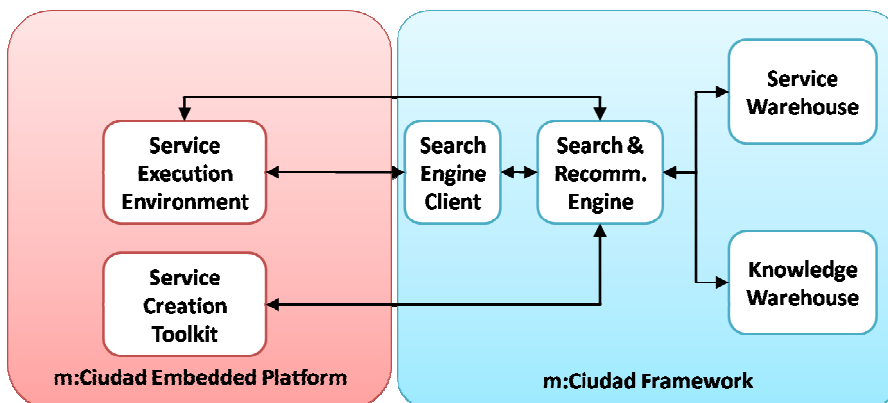
Nowadays digital information of any type is available at any given time and at almost any given place. At the same time the amount of generated digital information is growing tremendously. The amount of digital information in 2006 was estimated to be 161 billion gigabytes which is about 3 million times the information of all the books ever written [1]. By 2011 the amount is estimated to be 10 times the size it was in 2006 [2]. An unlimited access to an apparent endless amount of digital information demands for highly efficient search mechanisms which incorporate as much information about the user as possible. However, personalized search mechanisms are not sufficient to tackle the issue of information overflow. Personalized and context-aware recommendations are therefore a crucial factor to discover not only the sought-after information but to provide information beyond what is currently being requested by the user.

The search and recommendation engine build in the European funded project m:Ciudad [3] takes the requirements for future discovery tools described in the paragraph above into account. In the following m:Ciudad as a whole and its search and recommendation engine in particular are described. The search and recommendation engine focuses on how context information of mobile users is incorporated in the search and recommendation processes.

This paper discusses, how m:Ciudad can deal with a large number of short-lived services, and how ontology-based metadata can be used to evaluate services according to their relevance. In the end an outlook is provided which describes how the proposed mechanisms can be evaluated and extended in future work.

## 2 The m:Ciudad Architecture

The goal of m:Ciudad is to enable mobile users to create services on the mobile device by combining existing service elements. In addition users will be able to provide data for a service, i.e. participating in a multi-provider scenario or using content provided by a service. For all three scenarios a user needs to become a member of the m:Ciudad community by installing the m:Ciudad bundle on the mobile device.



**Fig. 1.** m:Ciudad Architecture

The architecture of m:Ciudad (see fig. 1) consists of a server part, the m:Ciudad framework and an embedded platform which resides on the mobile device. Due to the limited bandwidth and processing power of mobile devices data storage and processing is conducted on the server side. The most important components on the server-side are the Service Warehouse, the Knowledge Warehouse, and the Search and Recommendation Engine. Information about the created services is stored in the Service Warehouse and the Knowledge Warehouse whereas operational information about the service is stored in the Service Warehouse and meta information about the service is stored in the Knowledge Warehouse. For handling search requests and for

the provision of proactive recommendations the Search and Recommendation Engine requests information from the Service Warehouse and the Knowledge Warehouse. On the mobile device the Service Execution Environment is responsible for downloading and running services. Services are created in the Service Creation Toolkit. In the following a more detailed description of the different components is provided.

### **2.1 The Service Creation Toolkit**

The Service Creation Toolkit enables the user to create services. Two approaches for service creation are investigated. On the one hand a question-based approach is analysed. In this scenario the user is asked questions and depending on the answers the service elements are recommended to the user. Technical details of the service creation is transparent for the user. The block-based approach is another approach which is evaluated in the m:Ciudad project. Service creation is based on the logical connections of autonomous black boxes which are high-level parts of a service. It will also feature the annotation of services, which are stored in the Knowledge Warehouse.

### **2.2 The Service Execution Environment**

The Service Execution Environment is the core of the m:Ciudad Embedded Platform and resides on the mobile device. Its main function is to handle the execution of services. Furthermore it interacts with the Service Creation Toolkit and enables testing of new services on the mobile device before publishing them.

### **2.3 The Service Warehouse**

The Service Warehouse is the module in the m:Ciudad Framework which acts as a repository of services and interacts and communicates with several modules both in the Embedded Platform and in the m:Ciudad Framework. Depending on the role of the user, different data is retrieved from the Service Warehouse. In case the user acts as creator of a service, service templates are downloaded from the Service Warehouse. A user interested in providing data for an existing service downloads the full service code from the Service Warehouse. Finally, a user acting as consumer of a service is provided with the User ID of the user providing the service.

### **2.4 The Knowledge Warehouse**

The Knowledge Warehouse is focused on describing services using semantically enriched data. It facilitates search and discovery of services for the m:Ciudad community. This component will be used as a part of the global m:Ciudad architecture in collaboration with the Search and Recommendation Engine to support the service discovery, selection, and creation. The Knowledge Warehouse resides on the server side and is linked to different types of ontologies to represent domain

knowledge in different applications. The domain ontologies specify common concepts and their relationships in the domains of application for m:Ciudad (e.g. tourism, traffic, entertainment). However, searching the contents is not the main target in Knowledge Warehouse design. The services and contents are only annotated in a way that supports finding relevant services based on different search criteria. This will be done in cooperation with the Service Warehouse and the Search and Recommendation Engine.

### **3 The m:Ciudad Search and Recommendation Engine**

The m:Ciudad Search and Recommendation Engine (S&RE) comprises a new way for formulating queries, which are used both for search and recommendation scenarios. Moreover, it supports filtering and ranking based on ontology-based service descriptions. For the ranking and also for proactive recommendations the specifics of the mobile environment are leveraged to boost the quality of search and recommendation results.

#### **3.1 Query Formulation**

In m:Ciudad, there are different sources of information that can be leveraged for searches or proactive recommendations:

- *Search items / keywords*: These are terms that are defined by the user and are the only parameters that are explicitly defined by the user. The information is submitted as the user requests information from the system.
- *Context information*: This information is submitted automatically by the mobile device itself. The information includes for example GPS location, the current time, etc.
- *Profile data*: The user profile information is stored on the server-side. It is referred to whenever a user submits a query. For optimization purposes this information is only transmitted if it deviates from the general user profile of the user.

The unique ID of the user is also sent to the system along with the other parameters.

The search mechanism is different from “classic” approaches that rely on a set of search terms. We use the concept of “query extension” [3] to expand the query by adding user-related and contextual information. By this means, we want to achieve a higher quality of search results and provide a personalized and context-aware search mechanism.

For recommendations, there are no explicit search terms. However, there is profile information from the users and contextual data sent by the mobile terminal. The definition of the query is sufficient for both “pull” and “push” recommendations. For “push” recommendations, a background service continually monitors the user’s current situation and provides proactive recommendations when a certain “threshold of change” is reached. For “pull” recommendations, the process is similar to a search

without any search terms. In this case, the results are suggested by the system based on detected user interests and preferences.

In the following, we consider a query term  $t$  to be the atomic part of a search query  $q$ :

$$q = (t_1, t_2, \dots, t_n)$$

**Def. 1.** Definition of the Query Formulation

Furthermore, we define a query term  $t_i$  as following:

$$t_i = \langle c, n, v, w \rangle$$

**Def. 2.** Definition of a search term

- $c$  = category of the query term; i.e “keyword”, “context-information”, or for service creation a “input/output block”
- $n$  = unique name of the query term within a category within a single query; (e.g. “name” for the name of a user, or “name” as the name of the service)
- $v$  = value of the parameter (e.g. the actual name of the user, his current location, the actual search term, etc.)
- $w$  = weight of the query term within the query

As the query formulation shows, also requests from the Service Creation Toolkit can be handled by this generic query formulation. The category can be used to describe, whether the given search term is associated to an input or to an output block. The weight of the query term can be defined by the user, or can be determined by the system. A more detailed description on how these weights are actually defined and used, is provided in the following. When a query is formulated, weights are allocated to the different query terms. This could be done explicitly by users in an advanced search interface or implicitly. Some basic considerations are listed below:

- *Search terms*: If search terms are defined, i.e. we have an actual search and not a proactive recommendation, they should have a rather high weight.
- *User profile information*: a profile can contain the interest of a user, i.e. cooking and football. Users can define weights for different items in their user profile, i.e. cooking  $\rightarrow$  0.3 and football  $\rightarrow$  0.5 specifies that a user is more interested in services about football than cooking. When executing a query, the user profile information is matched to the other query terms in order to re-rank the profile information according to their relevance.
- *Context information*: In the following we only elaborate on location and date / time. We will consider these aspects in the design and the latter prototype development. However, context data could include more complex information such as the mood of the user and environment variables. The very generic query formulation as shown above also allows for such scenarios.
  - *Location*: The handling of locations is depending on whether a service is location-based or not. If a service is location-based, another important question is, whether this information is used as a hard criteria, i.e. the specified location **MUST** be within a certain range, or whether it is a

parameter that should be used for the ranking. For the first case, we assume the weight for location to be 1.0, i.e. it will be the most important parameter for selection; if a service is not within the specified range, it will not be selected.

- *Date / time*: The handling of date and time is depending on whether a service is time-dependent or not. If a service is time-dependent, another important question is, whether this information is used as a hard criteria, i.e. the specified timestamp **MUST** be within a certain range, or whether it is used as a soft criteria and thus a parameter that should be used for the ranking. For the first case, we assume the weight for location to be 1.0, i.e. it will be the most important parameter for the selection; if a service is not within the specified time range, it will not be selected.

In the following an example of a query is provided. A user enters the terms “cooking” and “football” and assigns the weight 0.3 to cooking and 0.5 to football. The resulting query  $\langle t_1, t_2 \rangle$  looks as follows. For  $t_1$  category = keyword, name is empty, value = cooking and weight = 0.3. For  $t_2$  category = keyword, name is empty, value = football and weight = 0.5.

Overall, there is a need to re-process these weights after the different query terms have been transferred to their ontological representations. We need to re-calculate the weights of search terms with respect to their relevance to the other terms in the query. Sections 3.3 and 3.4 describe, how a query is processed.

### 3.2 Creating the Search Index

To allow for an effective and fast search, we need an infrastructure that holds references to services according to some classification. For our scenario, where we have potentially millions of services, we cannot query the whole semantic infrastructure upon every search request. Therefore we need to build a search index.

This is the basic data structure that allows for a fast filtering. First, all query terms have to be transformed into an ontological representation. Querying for an ontological representation results in a list of service references that match the given ontological representation of query  $q$  (see definition 3).

$$f(o_{q,1}, \dots, o_{q,n}) \longrightarrow (s_1, \dots, s_m)$$

**Def. 3.** Definition for mapping ontology representations to list of service instances

For the fast and efficient access to service meta data, we hook into the lifecycle events of services in order to create a search index. This search index maps the generic meta data description structure to service instances. The generic meta data description structure comprises a set of ontology classes that have been used throughout the description of the service. When a service is created, all possible combinations of ontological representations will be mapped to the concrete service reference. For instance, a user creates a service named Foodball which is about the best cooking recipes for watching a football match. The most obvious ontological representation

for the service Football in the search index will be  $OR_1 = \{\text{football, cooking}\}$  (see fig. 2). OR stands for ontological representation. However, since football and cooking are part of an ontology, the different super-classes of Football and Cooking have to be considered as well for referencing this service.

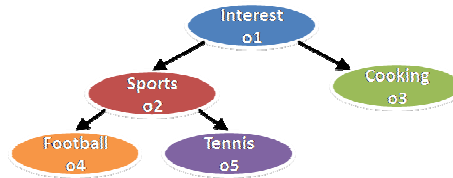
$S_1 = \{\text{„football“, „cooking“}\}$

$OR_1 = \{\text{Football, Cooking}\}$

$OR_2 = \{\text{Sports, Cooking}\}$

$OR_3 = \{\text{Football}\}$

$OR_4 = \{\text{Cooking}\}$



OR_ID	O_ID
1	o4
1	o3
2	o2
2	o3
3	o4
4	o3

OR_ID	S_ID
1	1
2	1
3	1
4	1

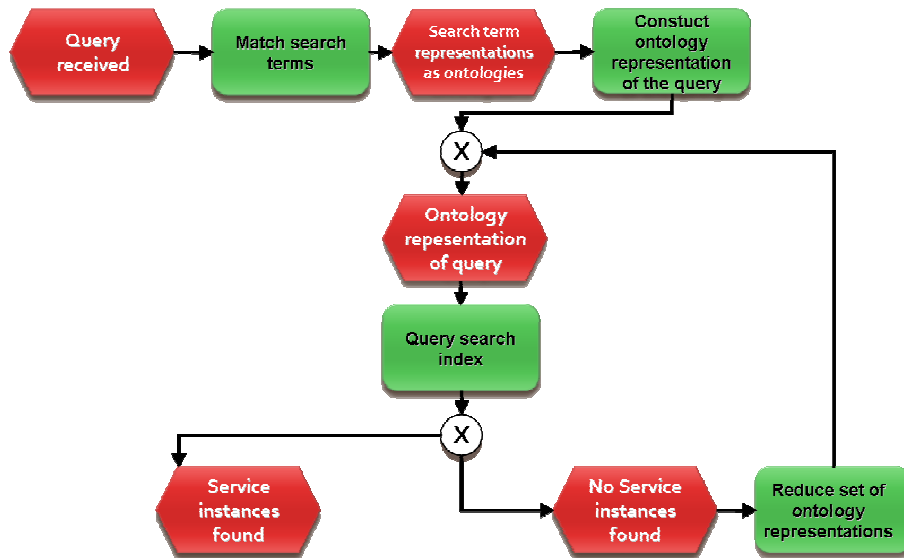
Fig. 2. Search Index Creation

The different ontology representations are created according to the inheritance hierarchy of the respective concepts. The ontology representation includes all the references to the contained ontologies (usually a URI – in fig. 2: o1, o2, etc.). Each service will be mapped to the ontology representations which can describe it. In accordance to how many generalization steps have been made, the weight of this representation can be adapted and saved with the representation in the search index, i.e. the more generalization steps have been made, the lower the weight of the respective ontology representation. For instance,  $OR_2$  is not as precise as  $OR_1$ , and so on. Many different services can be found according to the ontology representations. The weight of a given ontology representation and service id pair can be used to pre-select appropriate service instances according to their weight. Though we have high costs on the server-side for creating a service, we keep the costs low for the search and recommendation scenarios on mobile devices, as we are able to filter from millions of services to few of them.

### 3.3 Filtering

From a process point of view the filtering can be described as follows (see fig. 3). The filtering is triggered with a query and starts the matching of each search term. Having ontologies for each search term an overall ontological representation of the query can be built. This ontological representation is used to query the instance space where

each service instance is matched to an ontological representation. At this stage, two types of results are possible: either a list of service instances is found which match the ontological representation of the query. In this case the filtering process ends; Or no services instances are found so that the instance cache has to be queried again with a reduced ontology set. Reduction is conducted by removing the term which has the lowest weight.



**Fig. 3.** Filtering in m:Ciudad

The filtering process described in previous sections mainly aims at the discovery of service instances which match the overall search query. For some use cases we need to extend this mechanism with item-based evaluations that have even to be performed throughout the filtering.

For instance, a service might not be valid anymore or the user might not have access rights to the service. Another example is a service which provides information that is outside the physical range of the user.

As these interpretations already contain some information which are needed throughout the ranking, these intermediate results should be cached for the ranking later on. Depending on the search scenarios we have (location-based, time-dependent search, etc.), we need a configuration which describes, which parameters should be evaluated throughout filtering.

For all the different aspects that need an evaluation while filtering, this information needs also to be included in the table that maps ontology representations to service ids. For time-dependent search there could be an additional attribute that can be queried throughout the filtering.

### 3.4 Ranking

The ranking function (see def. 4) is a combination of weights of the different search terms and implicit and explicit user feedback.

$$r(q_s) = \sum_i r(t_i) + \gamma \cdot r(s) + \delta \cdot r(or_s) + \varepsilon \cdot r(inst_s) + \varphi \cdot r(ext_s)$$

**Def. 4.** Definition of the ranking function

- $r(q_s)$ , Overall ranking result of a service  $s$  which was returned as a result for query  $q$ .
- $r(t_i)$ , Evaluation of the search term  $t_i$
- $r(s)$ , Feedback about service  $s$
- $r(or)$ , Feedback about service description of service  $s$  which is represented via its different ontologies
- $r(inst)$ , Feedback about how often service  $s$  was installed
- $r(ext)$ , Feedback about how often service  $s$  was extended to a new service
- $\gamma, \delta, \varepsilon, \varphi$ , Parameters to emphasize or reduce relevance of function terms of ranking equation

At first the search terms of query  $q$  are considered in the ranking function. The sum  $\sum r(t_i)$  describes the overall evaluation of the different search terms  $t_i$ . Each search term is evaluated by its assigned weight and additional parameters. For instance search term  $t_1$  specifies that the desired services should provide information not older than an hour. If the services  $s$  meets this requirement the Boolean parameter 1 is assigned, if the services  $s$  does not meet this requirement the Boolean parameter 0 is assigned.  $r(t_i)$  in this particular example equals 1 multiplied with the weight assigned to term  $t_i$  for the first case and 0 in case service  $s$  does not meet the requirement. Hence,  $\sum r(t_i)$  adds the different evaluations of each search term  $t_i$  of query  $q$ .  $r(s)$  represents the feedback about the service provided by the users. Here, different aspects of feedback are considered, e.g. collaborative or hybrid mechanisms.  $r(or)$  is feedback about the description of the service, i.e. whether it was helpful.  $r(inst)$  and  $r(ext)$  are statistical parameters obtained from the Knowledge Warehouse.  $\gamma, \delta, \varepsilon, \varphi$  enable a variation of the relevance of different terms of the ranking function and thus, a variation of order of services in the result list.

Since the list of service instances is a list of ontological representations that matches the ontological representation of the search query, weights of the search terms can be directly used to evaluate the overall ranking of the service instance. Furthermore, parameters  $\gamma, \delta, \varepsilon, \varphi$  have to be adjusted in relation to the user role. For instance, if the user searches for services for consumption  $\gamma$  will be assigned a higher value than  $\varphi$ . In the reverse conclusion,  $\varphi$  is assigned a higher value, if the user acts as services creator.

### 3.5      **Enabling Proactivity in a Mobile Environment**

The m:Ciudad bundle on the Mobile Embedded Platform includes a browser, which can be used to access to the Search and Recommendation Engine specific URL in order to get the Search and Recommendation Engine main page; when this page is rendered in the user terminal, the user fills in all the parameters (s)he wants to include in the request and then clicks on the “Search” button which sends a request to the Search and Recommendation Engine. However, before this request is really issued, the following information has to be accessed and included in the request:

- *User ID*: The unique user ID to identify the current user
- *Context information*: All capability information by the current device (esp. location<sup>1</sup>; time can be determined in the backend automatically)

The User ID will be part of the user configuration and will be read from the m:Ciudad configuration file but the context information has to be requested from the Capabilities Manager which is accessed through the Service Execution Environment.

As the search terms and the user profile data can be edited within the Search Client, these should not be provided by the Service Execution Environment. The listed information above will be part of an HTTP session, so the Search Client can use it to load the user profile data from the Service Warehouse (in case of the user ID) and to weigh context data according to their importance.

To enable proactivity in the Search & Recommendation Engine, the Service Execution Environment should also permanently poll the Search & Recommendation Engine for proactive recommendations, as the Search & Recommendation Engine cannot directly address a mobile client without prior request. In this case the same data should be transmitted as with the search request. A response should indicate, whether there are recommendations for the user or not.

In order to keep the user effort as small as possible, the intervals for pushing information from the Search and Recommendation Engine should be quite large. As the Search and Recommendation Engine can keep track of all other parameters all by itself, it only needs to be notified for location changes. For the implementation of the prototype we consider an interval of 5 minutes to be appropriate for this purpose. The option should be investigated, whether the mobile device can determine a certain kind of variation threshold for the parameters location and time. According to that, an update request would be initiated, whenever a certain deviation of the previous location or time has been reached.

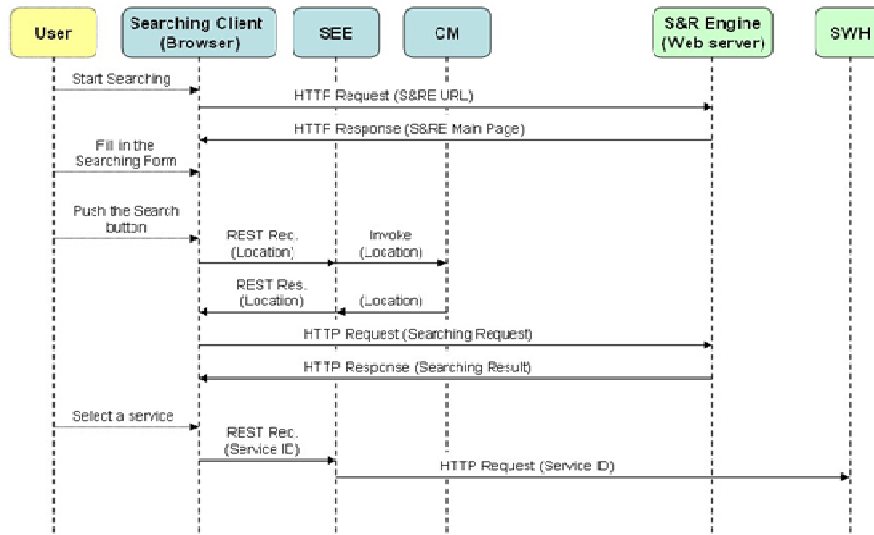
An efficient searching interaction between a user and the Search and Recommendation Engine will result in a service being selected for installation and execution on the mobile terminal. At this stage a new interaction between the Search Client and the Service Execution Environment will be required in order to pass the Service ID of the selected service from the Search Client to the Service Execution Environment; from then on the Service Execution Environment will launch the

---

<sup>1</sup> The option to include information from the network providers about the current location will be analyzed throughout the implementation of the m:Ciudad prototype.

process of downloading the service from the Service Warehouse and installing it in the terminal.

Figure 4 depicts the main interactions between the Search Client and the Service Execution Environment.



**Fig. 4.** Interaction between User, the Search and Recommendation Engine, and the Service Execution Environment

#### 4 An Instance Scenario

A group of friends is having dinner and spontaneously decides to go out afterwards. One of the friends, a member of the m:Ciudad community, starts looking for a service which provides information about cool events. He specifies via an advanced search interface, that the service has to provide information for events that are taking place in a time range of 5 minutes from now. Furthermore, the Search and Recommendation Engine knows from the user's profile that he likes Rock music. Thus, the system starts looking for services that provide information about events which include Rock music. Having found a list of services which match the different criteria, the filtering process removes all services which provide information about events in cities other than the user is in. The ranking function furthermore gives a higher rate to Rock events than to other events. Having filtered and ranked the services the Top 5 services are returned to the user. Figure 5 shows, how this query can be specified on the mobile phone.



Fig. 5. Sample Screenshot of the Advanced Search Interface

## 5 Related Work

In recent attempts context-aware recommender systems have been combined with ontologies to further improve the recommendation by adding semantics to the context. For instance, description of a meeting situation specified in terms of location, time and people can be extended with a qualitative, high-level description such as “Important Work Meeting”, “With Friends”. In addition, the mobile service description itself can be extended by a service class ontology/taxonomy describing a service as “Real-time Communication Service”, “Entertainment Service”, “Messaging Service”, etc. In other words, service and item representation can be extended across multiple dimensions and taxonomies. Using ontologies, the quality of content description is notably improved for machine processing and reasoning purposes. Domain ontologies enable semantic matching of objects and profiles instead of a simple keyword-based matching or comparison of syntax [5].

In m:Ciudad framework, contextual information such as “location” or “current available bandwidth” is subject to constant and quick changes. Implementation of a context-based recommendation system in a mobile environment is particularly difficult. In the following some of the different approaches of context-based recommender systems in a mobile environment are described.

Woerndl et al. [6] describe recommendation of mobile applications to users based on what other users have installed in a similar context. A hybrid recommender system is created by letting the user select between several content-based or collaborative filtering components. In addition, a rule-based module using information on point-of-interest information in the vicinity of the user is provided. By this means context-dependent information can be recommended to the user, e.g. services of “Real Madrid” can be proposed to the user when he walks in the vicinity of the Bernabeu stadium.

Chen [7] discusses a context-aware recommendation system that predicts the user's preferences on past experiences of like-minded users in "tourist activities" domain. For instance, a user seeks recommendations via a mobile device for activities in the relation to the current weather, location and travelling companions. The recommendation engine suggests activities which similar users have done in a similar context. In comparison to the traditional approaches described above, this approach defines a context similarity measure and includes it in the collaborative filtering rating equation, i.e. a collaborative filtering system is combined with context-aware recommendation.

Van Setten et al. [8] describe a context-aware recommendation engine for mobile tourist applications which uses ontologies. In their approach a hybrid system is constructed by combining a recommender system and a context-aware system. At first, the context item "location" is used as a hard criterion to select relevant services that are in the vicinity of the user. Then, the predicted interest of the user is used as soft criterion to order the remaining services. Requests are triggered either by the user himself or by a detected change of context. For instance, the systems provides proactive recommendations, e.g. nearby buildings, buddies or other buildings, depending on the location and the profile of the user. The recommendations are adapted automatically if the context of the user changes, i.e. if the sensors of the mobile device detects an increase of the user's speed, the application assumes that the user is driving and increases the radius of the area for providing recommendations. For the service description, the semantic web technologies, and in particular OWL, are applied to create additional annotations of service elements, e.g. service type, inputs, outputs which have to be filled in by the service provider.

Zhou et al. [9] developed a recommendation system which is based on user preference, situation context, and capability context for supporting context-aware media recommendation for smart phones. They define a recommendation model for dealing with a wide variety of recommendations. The model provides multimedia recommendations over multiple dimensions to generate multidimensional output. Context is represented via ontology-based context models which include the user's media preferences. For the evaluation of the recommendations three different approaches are combined, namely content-based, Bayesian-classifier, and a rule-based approach. The content-based approach measures the similarity between a media item and the preference of the user then the Bayesian Classifier calculates the probability of the item belonging to the situation context, finally a weighted linear combination of the subscores is applied to calculate the overall score.

## 6 Conclusions and Outlook

This paper discusses how proactive and context-aware recommendations are applied to mobile services in the m:Ciudad Search & Recommendation Engine. Ontologies are used to describe services, users, situations and their domain knowledge. The ontology-based data representations are used as the basis for search and recommendation. The approach for the query formulation is generic and can be used

in different service usage scenarios (service creation, service provision, "plain" service usage). Moreover, it serves as internal data structure for both searches and recommendations. The ranking takes specifics of the mobile system into account, e.g., it uses the number of service installations to refine the ranking. Proactivity is enabled by a triggered request for proactive recommendations from the mobile client.

M:Ciudad somewhat differs from the other approaches in similar domains, as it directly addresses a potentially huge number of services and features a filtering method, which allows for highly scalable filtering of millions of services.

Overall, the depicted approach can be used to semantically classify and describe different types of services, items, etc. across the border of a single application. The query formulation is generic and can be adapted to other domains and applications; it is not strictly bound to mobile platforms. Nevertheless, the specifics of a mobile environment are taken into account, and are considered while ranking the different service instances and enabling proactive recommendations in such an environment.

Future work will focus on using the semantics provided by the ontology infrastructure to derive facts from probabilistic measures taken from the user feedback and also the reasoning mechanisms. Moreover, by analyzing the historical data of searches and recommendations, optimization can be learned and reified in the ontology representations, in order to boost the performance of searches and recommendations.

In the m:Ciudad project, the Search & Recommendation Engine will be implemented throughout the next steps. A preliminary set of domain ontologies has been created or reused from existing ontologies, such as FOAF [10] and SKOS [11]. The prototype will be also evaluated against common metrics such as precision, recall and fall-out.

## 7 Acknowledgement

This paper describes work undertaken in the context of the m:Ciudad project, m:Ciudad - A metropolis of ubiquitous services (<http://www.mciudad-fp7.org/>). m:Ciudad is a medium-scale focused research project supported by the European 7th Framework Programme, contract number: 215007.

## References

- [1] The expanding digital universe, A forecast of worldwide information growth through 2010, An IDC white paper, March 2007.  
<http://www.emc.com/collateral/analyst-reports/expanding-digital-idc-white-paper.pdf>
- [2] The expanding digital universe, A forecast of worldwirde information growth through 2011, March 2008.  
<http://www.emc.com/collateral/analyst-reports/diverse-exploding-digital-universe.pdf>
- [3] m:Ciudad project homepage. <http://www.mciudad-fp7.org/>

- [4] A. Pretschner, S. Gauch, "Ontology based personalized search, In Proceedings of 11th IEEE International Conference on Tools with Artificial Intelligence, pp. 391-398, 1999.
- [5] A. Costa, R. Guizzardi, G. Guizzardi, and J. Filho, "CORES: Context-aware, Ontology-based Recommender system for Service recommendation", UMICS'07, 19th International Conference on Advanced Information Systems Engineering (CAISE'07), 2007.
- [6] W. Woerndl, J. Schlichter, "Introducing Context into Recommender Systems", Short Paper, Proc. AAAI'07 Workshop on Recommender Systems in e-Commerce, 2007.
- [7] A. Chen, "Context-aware collaborative filtering system: predicting the user's preferences in ubiquitous computing", Conference on Human Factors in Computing Systems, 2005.
- [8] M. van Setten, S. Pokraev, J. Koolwaaij, "Context-Aware Recommendations in the Mobile Tourist Application COMPASS", Lecture Notes in Computer Science, vol. 3137/2004, pp. 235-244, 2004.
- [9] Y. Zhiwen, Z. Xingshe, Z. Daqing, C.-Y. Chin; X. Wang; M. Ji "Supporting Context-Aware Media Recommendations for Smart Phones", Pervasive Computing, IEEE, vol 5, issue 3, pp. 68-75, 2006.
- [10] <http://www.foaf-project.org/>
- [11] <http://www.w3.org/2004/02/skos/>