Context-aware recommendations on rails

Introducing a context-aware recommender construction kit

Tim Hussein, Timm Linder, Werner Gaulke, Jürgen Ziegler
Interactive Systems and Interaction Design, University of Duisburg-Essen
Lotharstr. 65, 47057 Duisburg, Germany
{tim.hussein, timm.linder, werner.gaulke, juergen.ziegler}@uni-due.de

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering, Selection process, Search process*

General Terms

Design, Management, Human Factors

Keywords

Recommender Systems, Personalization, Context-awareness, Hybrid recommender systems, Semantic recommender systems, Ontologies

ABSTRACT

In this paper, we introduce a framework for modular generation of context-aware recommendations. The components of this framework include context sensors, recommender algorithms and utility modules (converters and filters), all realized as so-called *services*, which can flexibly be combined in terms of a recommender construction kit. Different areas of an application (e. g. a web portal) thus can be powered by a distinct recommendation-providing service chain. In this way, different recommender techniques can be used in parallel, either separately or combined, with or without incorporating contextual information, which makes it a framework for context-aware, hybrid recommendation generation.

1. INTRODUCTION

With the help of recommendations, large collections of products or services are made accessible. Recommender systems support users by recommending content considered as being particularly interesting for them. Recommender systems play an important role in handling large amounts of information. Often, the content and artifacts a person might be interested in, depend on the specific situation: The current location, season, user role, temperature, etc. Context-aware recommender systems try to exploit the usage context in order to improve the recommendation generation process.

Unfortunately, to this day, no commonly accepted technical definition of context does exist. Just for the term "context", there are more than 150 definitions from various disciplines. One of the most frequently cited definitions was proposed by Abowd, Dey and others [1]:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

This rather broad definition lets the designer decide what he or she considers as relevant contextual information. In this paper, we introduce a technique that incorporates both context information derived from system interaction (clickstream, history) as well as information about the external circumstances (location, time, season, etc.).

First, an overview of existing approaches and their shortcomings is given. Based on this overview, we introduce DISCOVR, our approach of a context-aware recommender framework. Along with a conceptual overview of the DISCOVR architecture, we will outline some of its capabilities. After the technical overview, we present a validation by an example.

2. CONTEXT-AWARE AND HYBRID RECOMMENDER SYSTEMS

Combining different techniques has been one of the main research interests within the field of recommender systems as well as incorporating contextual information [3, 4]. Context can be used in several ways to enhance recommender systems. Shepitsen et al. for instance introduce an agglomerative hierarchical clustering algorithm for social tagging systems [11]: A user-selected tag is used for a context-dependent limitation of the selected set of clusters considered for the recommendation process.

A content-based model for adaptive recommendations with the help of a use context is described by Kim and Kwon [9] using a set of four ontologies from which the "use context" for a user is derived: Product, location, shopping record and consumer. Most valued products are taken from the product ontology with help of the consumer preferences and shopping record. The most valued recommendations are taken and displayed in more detail. Products are ordered in a concept hierarchy from broad to most specific. If the user chooses a concept, the context switches and more specific information is shown. In this way, context is used to control the information detail of recommendations.

Adomavicius et al. add contextual information as additional dimensions to the given user and item dimension in an collaborative filtering approach [2]. The recommendations are derived from the item ratings under the given context.

Contextual information for both the involved items and recommendation process itself is proposed by Loizou et al. [10], who use an ontology containing information about items and the recommendation process. The ontology is build with the help of web services and expanded over time with new data to match the current recommendation context at runtime. Suchlike information could for instance contain metrics for usefulness for specific users. To extract recommendations, this ontology is mapped into a vector space from which only relevant parts are sliced out.

All theses systems work well in their specific environments, but we think there is still a research gap regarding systematic, generic and extensible integration of context information into the recommendation process. In this paper, we introduce a framework that addresses these questions and, beyond that, can be very useful for hybrid recommender system prototyping as different recommender techniques and context sources can smoothly be combined.

3. THE DISCOVR FRAMEWORK

Like many other context-aware approaches (including some of the systems presented in Section 2), DISCOVR utilizes semantic background information represented as ontologies. All item- and context-related information to be incorporated by the system has to be modeled in an ontology[1], for instance *[CD Garth Brooks - Beyond the Season] → [suitable for holiday] → [Christmas]*. Surely, this is a laborious task, but DISCOVR is mainly a framework for prototypical development and the scenarios will more likely be several hundred items and concepts rather than millions.

The components introduced in the upcoming section exploit this information and build context-aware recommendations upon.

3.1 A service architecture for recommendations

We identified three major areas of concern for context-aware recommendations: *Sensing* the user's context, *generating recommendations* thereupon and finally *presenting* the recommendations to the user in an appropriate fashion.

One important idea that was present throughout all our concepts was to realize the different components in the form of separate *services*. Each service has some kind of input and some kind of output, so that they can be connected to each other. Our first thought was then to arrange the services in a pipeline structure, where all services work on a shared semantic model that is passed through the pipeline and modified in turn by each service.

Unfortunately, this concept proved to be too inflexible, so we deviated from the pipeline idea by arranging the services in a *service process*. Instead of linearly connecting the services, they were now set up in a directed dependency graph built by a dedicated component, the *service processor*. The major advantage of this approach is that the output of one service can be exploited by an arbitrary number of services at any later point in time during this process.

Another important problem was the design of the service interfaces. Whereas in our first concept, all the inter-service communication was done using a shared semantic model, we faced one major problem with this solution: A service that is built upon the output of a preceding service would have to know too much about the internals of that service in order to interprete its result. Since semantic models can basically contain statements of any kind, their form would have to be restricted in a way that the interpreting service could work with them. That, however, negates the point in using semantic models at all.

In our current approach, we chose a way in between. While each service can present its data as a semantic model, for instance for visualization and explanation purposes, most of the communication between services takes place "on rails" in the way that the output type of one service matches the input type of another. We divided the input types (which we call *triggers* because they trigger the receiving component) and the output types into five distinct categories (Figure 1):

- *Semantic models*,
- *Sorted lists of resources* from our domain model, with the sorting order based upon the importance of the given resource,
- Lists of *weighted resources* with assigned weights between 0 (unimportant) and 1 (very important),
- *Numeric values*,
- *Textual data.*

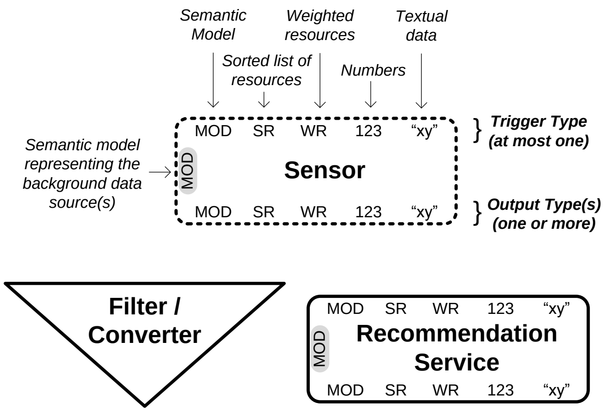

Figure 1: Types of components of our service architecture, together with possible input and output data types.

Each service accepts triggering input of one specific type at most, which may be provided by one or more triggering services. The output of a service, on the other hand, can be manifold, depending on the particular implementation.[2] Additionally, each service can possess one or more

[1]It may be useful to split the information into several ontologies for later reuse of parts in other scenarios

[2]Item-based recommendations for instance could be delivered both as ordered or weighted resources.

background data source(s) in the form of semantic models (RDF triples) if needed.

In Discovr, we distinguish three different categories of services (Figure 1): *Sensor*, *recommendation* and *utility services*, all of them now explained in more detail.

3.1.1 Sensor Services

A sensor is a service that acquires information about the user's context: Information derived from system interaction or information about external circumstances like the current temperature, which might be determined by contacting a web service. For a sensor to operate, it might first be necessary to retrieve already known information from another sensor: A weather sensor might need to know the user's current location, for example.

Most sensors' output has the form of a list of weighted resources (for instance closest cities along with their degree of adjacency). Some sensors, on the other hand, simply return literal values like strings or numbers (IP address, geographic coordinates).

3.1.2 Recommendation Services

Recommendation services use their input in order to generate recommendations (context-aware ones, if the output of sensors is being incorporated). Typically, recommendation services require a background data source containing items to recommended and their relations to each other in order to work properly.

The output of recommendation services is usually twofold: First, a recommendation service produces a list of ordered or weighted resources that can (after a possible filtering step) be presented to the user. Second, a recommendation service also returns a semantic model that is a sub-model of the background data source input and can be utilized as the background data source of yet another recommendation service, such that these services can be chained.

3.1.3 Utility Services

The last kind of services are utility services. Such a service might *filter* its input according to specific criteria, like selecting only resources that have a specific type, or limit the amount of results. Another example of a utility service is a weighting service that assigns weights to a given, sorted list of resources according to a specific formula (virtually acting as a *converter* between sorted and weighted resources).

3.2 Recommendations on rails

We call this approach *recommendations on rails*, because it follows the two major principles of frameworks like Ruby on Rails, Grails, etc.: *Don't repeat yourself (DRY)* and *Convention over Configuration (CoC)*. All components make use of encapsulation to a great extent, so that they only have to implement their very core functionality and cover the rest by the use of filters and additional services (DRY). The possibility of reusing the result of a service several times within a single complex recommendation process follows the DRY principle as well.

Furthermore, due to the clear distinction between different types of in- and output interfaces, the services can flexibly be combined without the hassle of custom connectors. A component implementing the *Sorted resources provider* interface can always be used as an input for a *Sorted resource triggered* service following the CoC principle (like using rails).

4. EXPERIENCES

We developed a virtual shopping and leisure portal including about 500 items like DVDs, CDs, sport events, concerts, etc. to demonstrate DISCOVR's potential (Figure 2) and implemented a set of components as reusable building blocks for hybrid, context-aware recommender systems.

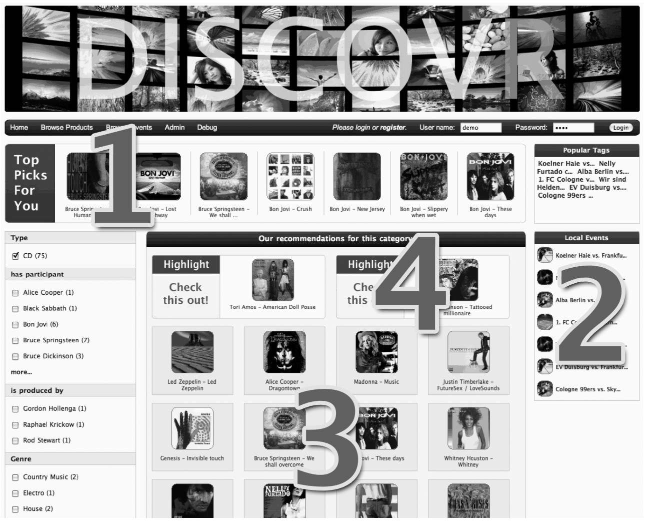

Figure 2: Screenshot of a web-portal powered by DISCOVR including different areas for recommendations. In this case: (1) is a 'traditional' recommendation block for product recommendations based on a Spreading Activation algorithm and the latest user clicks. (2) shows local events suitable to current weather conditions. In (3) recommendations for a chosen category (e. g. CDs) are presented with certain items (with special relevance to upcoming holidays) presented as highlights (4).

4.1 Portal architecture

We implemented Discovr on top of the Spring Framework[3] as a Java web application including interfaces and abstract classes reflecting the components introduced in Section 3. In addition, we realized several concrete sensor, recommender and filter modules for testing purposes: 8 Sensors for location, click-stream, weather, season, etc.; 3 recommenders (Item-based Collaborative Filtering, Spreading Activation and Rule-based) as well as 5 filters like a Sorted Resource Weighter or a Weighted Resource Filter.

We assume that the general principles of item-based CF and rule-based recommendations are commonly known among the readership. Spreading Activation is a concept proposed in the 1970s by Collins and Loftus [5] and was originally applied in the fields of psycholinguistics and semantic priming. Later, computer scientists adopted the idea: The principles have successfully been used in several research areas in computer science, most notably in information retrieval [6] or for predicting user behaviour [7]. The basic idea is that within a semantic network, certain elements are initially activated and spread this activation to adjacent elements. This activation flow runs through the network until a certain stop condition is met. In our case, the semantic model supplied

[3]http://www.springsource.org

by the background data source is converted into a directed graph, and those elements reflecting the input values are initially activated and spread this energy in a highly customizable fashion within the network. The service's output is then a list of resources together with their activation weights corresponding to the activation values obtained during the activation process. Details about our Spreading Activation Implementation can be found in one of our previous publications [8].

The portal makes strong use of the *MVC* design pattern, so that the information contained in the model can be displayed in various ways with the components only being loosely coupled.

Recommendations are generated using DISCOVR's service framework. A *service processor* builds a dependency graph of all registered services and then executes them in order. These services are realized as Java beans that are set up using an XML configuration file. Possible configuration parameters of a given service include the services that trigger this service, the background data source (which is, in most cases, the domain model), or service-specific items like certain filter criteria for a filtering service. A virtual service editor is planned as a future extension to make the whole setup of the service chain configurable at run-time.

After the services have been processed by the service processor, the output of any service at any part of the service process can then be presented to the user, for example in the form of (context-adaptive) recommendations. To achieve this, a so-called *view preparer*[4] converts the service's output, for instance a list of weighted resources, into a human-readable form by adding images, labels, descriptions and so on. The resulting model is then integrated into a JSP template for display to the user, as shown in Figure 3.

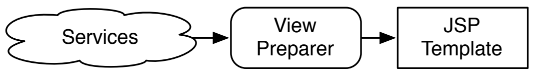

Figure 3: View Preparers preprocess the service output for a desired information block within the portal (hyperlinks, images, etc.).

4.2 Context-aware recommendations

Now we want to give two examples of service processes that produce context-aware recommendations. The first one (Figure 4) creates recommendations for events, that are a) taking place in (or close to) the user's city, b) are suitable for the current weather conditions and c) reflect the user's general preferences, like his music taste or his favorite sports club, which we deduce from the user's click-stream.

As can be seen in the schematic, the examined service process has actually two starting points demonstrating that these processes do not have to be linear. In the left branch, the IP Sensor retrieves the user's current IP address and returns it as a character string. The Geographic Coordinate Sensor uses this IP address to determine the user's present location (via a publicly accessible web service). The resulting geographic coordinates then trigger the City Sensor, which now finds the closest cities by comparing the coordinates to those stored in the domain model[5]. The result is a list of weighted resources ('Duisburg → 1.0','Cologne → 0.2', which means that Events in Duisburg are highly recommendable and events in Cologne at least a bit).

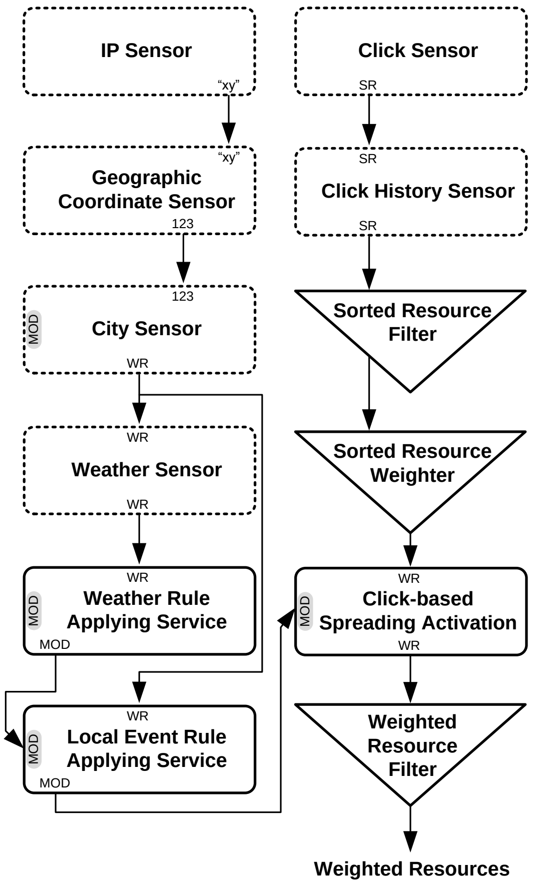

Figure 4: Example of the interaction of several services. The result of the Weather Sensor (e. g. sunny) is used as a trigger for a rule-based selection service that restricts the domain ontology model to those events that appear interesting under the given circumstances (e. g. outdoor events). Then, another rule uses the City Sensor's result to further restrict the ontology so that only events in the user's city are included. Finally, a Spreading Activation Recommender highlights those events that are related to the products and events that the user has already clicked on. The end result can be limited in its size and filtered to include only events of a specific type (e. g. concerts).

Now that we have established the cities closest to the user, this result is fed into two other services: The Weather Sensor uses the closest city and determines the weather at this location (using another web service). The result, e. g. '*sunny_weather → 0.8*', then triggers a rule-based selection service that filters the domain model using a given set of rules such that the remaining sub-model only contains individuals that are associated with *sunny_weather*. Second, the

[4]We use Apache tiles to define certain independent blocks inside one page, each one powered by a view preparer.

[5]The domain model is a ontological representation of the application domain as described in Section 3.

results of the City Sensor also serve as a trigger for another rule-based selection that takes the model from the preceding selection as a background data source and specifically selects events from the sub-model that are located in the user's current city.

Subsequently, the right branch (Figure 4) is being processed: The products and events the user has clicked on are obtained using the Click Sensor and recorded (for the remainder of the session) using the Click History service. After filtering these clicked items by limiting the item count and assigning weights to them depending on the time that has passed since the click[6], they trigger a *Spreading Activation*-based recommendation process. This process takes the semantic model from the left branch as a background data source, i. e. it works on the model containing the local events that fit to the current weather.

Finally, the resulting list of weighted resources can be restricted to e. g. the ten most important items. The final weighted resources list is now formatted and displayed to the user as described in section 4.1.

With the recommendation process illustrated in the second example (Figure 5), we can recommend products to the user that are associated with upcoming holidays (e. g. Christmas) and fit to the item the user is currently looking at. For example, when the user's last click was on a romantic movie, we would recommend (romantic) movies that are, in some way, related to Christmas.

To achieve this, the output of a Holiday Sensor triggers a rule applying service restricting the domain model to only those elements related to just this holiday. This sub-model then serves as the background data source for an item-based collaborative filtering triggered by the user's last visited item. For the sake of brevity, the additional background data required for the item-based collaborative filtering approach (clicked items of other users) is not modeled here.

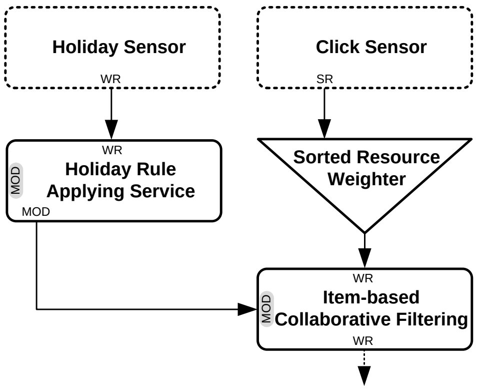

Figure 5: Another example of a service process. In this case, a combination of a rule-applying selection service and an item-based collaborative filtering approach has been chosen.

5. CONCLUSIONS

With DISCOVR, we introduced a framework for systematic, generic and extensible integration of context information into the recommendation process. DISCOVR facilitates the process of rapidly prototyping hybrid recommender systems as it provides an extensible architecture for smooth integration of various recommender techniques and context information.

[6]We assume that current clicks are more important than older ones.

6. ACKNOWLEDGEMENTS

The research presented in this paper is part of the CONTICI project, in which the Universities of Duisburg-Essen, Siegen, Hagen, and Aachen take part. CONTICI is funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).

7. REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer.

[2] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems*, 23(1):103–145, 2005.

[3] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

[4] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[5] A. M. Collins and E. F. Loftus. A spreading activation theory of semantic processing. *Psychological Review*, 82(6):407–428, 1975.

[6] F. Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6):453–482, 1997.

[7] W.-T. Fu and P. Pirolli. Snif-act: A cognitive model of user navigation on the world wide web. *Human-Computer Interaction*, 22(4):355–412, 2007.

[8] T. Hussein and J. Ziegler. Adapting web sites by spreading activation in ontologies. In L. Bergman, J. Kim, B. Mobasher, S. Rueger, S. Siersdorfer, S. Sizov, and M. Stolze, editors, *Proceedings of the International Workshop on Recommendation and Collaboration (ReColl)*, 2008.

[9] S. Kim and J. Kwon. Effective context-aware recommendation on the semantic web. *International Journal of Computer Science and Network Security*, 7(8):154–159, 2007.

[10] A. Loizou and S. Dasmahapatra. Recommender systems for the semantic web. In *ECAI 2006 Recommender Systems Workshop*, 2006.

[11] A. Shepitsen, J. Gemmell, B. Mobasher, and R. Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In *Proceedings of the 2008 ACM conference on Recommender Systems (RecSys)*, pages 259–266, New York, NY, USA, 2008. ACM.