A Framework and an Architecture for Context-Aware Group Recommendations

Tim Hussein, Timm Linder, Werner Gaulke, and Juergen Ziegler

University of Duisburg-Essen, Germany
firstname.lastname@uni-due.de
http://www.interactivesystems.info

Abstract. In this paper, we propose a generic framework to generate context-aware recommendations for both single users as well as groups. We present the the concept of *context views* and a corresponding architecture implementing the framework as well as exemplary recommendation workflows for group recommendations.

Keywords: Context, Recommender Systems, Adaptation.

1 Introduction

Recommender Systems [3,9] are a well-established means for improving the user-experience in e-commerce, collaborative environments, or e-learning. Although existing approaches are quite successful, we see potential for improvement especially regarding two aspects:

1. Most recommender systems aim at single users, not groups.
2. Recommenders usually don't take the particular usage context into account.

We think that the usage context contains valuable information to improve recommendation quality, because recommendations, especially those for groups, highly depend on the current context (for instance the social or historic context as well as time or location). As a solution, we propose a generic framework for generating recommendations for both single users and groups with respect to the context at hand, where context can be virtually anything from external circumstances (time, place, etc.) to information derived from system interaction (browsing history, downloads, etc.).

Jameson and others [7,8] distinguish between four sub-tasks for group recommendations: (a) Acquisition of preferences, (b) Recommendation Generation, (c) Presentation and (d) Achieving a consensus. Although (c) and (d) may be important for many systems, we focus on group preference acquisition and recommendation generation (and combine this with context-awareness).

2 Context Views

A truly adaptive system should consider both the users' current state and context information to select the most important entities and concepts with regard to the

G. Kolfschoten, T. Herrmann, and S. Lukosch (Eds.): CRIWG 2010, LNCS 6257, pp. 121–128, 2010.

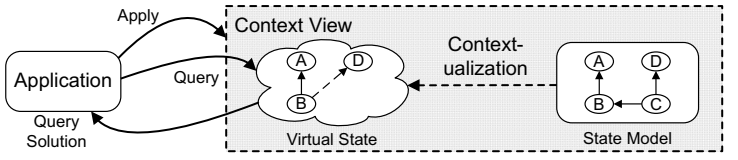

Fig. 1. A client triggers a context view on the state and queries the resulting virtual state. The contextualization process transforms the elements of the state. In this contextualization process, C is not important (and thus removed), whereas a virtual connection between B and D is generated (dashed) as a contextualization result. Virtual new entities could be added as well (although this is not the case here).

current situation. *Context views* apply contextualization operations to a state resulting in a *virtual state* representing a certain view on the state. The name context view refers to the view concept in database theory, where a sequence of operations leads to a *virtual table* that can thereupon be queried as if it were a regular one (Figure 1). Each view shifts the focus to other information from the state model thus enabling different forms of adaptation.

Context, in our understanding, can be virtually anything from external circumstances (location, weather, etc.) to internal resources (click-stream, state of the application, etc.). This rather broad understanding of context goes along with the often cited context-definition by Dey & Abowd [2]. Applying the *view* concept to context-adaptivity, we define a context view as a sequence of contextualizing operations leading to a virtual state. Contextualizing operation can be any means that are able to analyze initial states (with respect to the context), draw conclusions and create virtual states thereupon. In the *context* of a certain project, a company's chat application could perform operations like this:

```
(1) Identify all users working on the same project & attending the same chat.
(2) Mark past chat protocols of sessions, in which at least 2 users working
    on that particular project participated.
```

This example highlights the project context, but any other aspect like the current location could be used as well. The contextualization does not affect the original state at all. Technically, the contextualization process is spawned with a copy of the original state. From a more abstract point of view this means, that only events change the current state. Reasoning, on the other hand, leads to a cognitive model *reflecting* reality but does not change it per se.

A more detailed conceptual explanation of context views including an introduction of the underlying conceptual framework for context-adaptive applications has been published by the authors in an earlier publication [4].

3 Architecture

In this section, we present the architecture of a Java-based client-server architecture called *Hybreed Context Views*[1], implementing the context view concept.

[1] See `http://www.hybreed.orgformore(especiallytechnical)information`

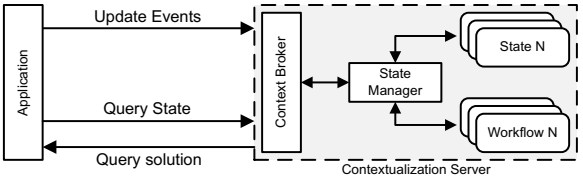

Fig. 2. Overview of the client-server architecture. The access type depends on, whether the server is embedded in an application or deployed on a remote destination.

The core of Hybreed Context Views, the so-called *contextualization server*, makes use of standardized web-service protocols (SOAP/WSDL) and serves the following purposes (see Figure 2 for an illustration):

- Managing the distinct user states.
- Aggregation of the user states to a global state.
- Applying context views to a state, either user or global, to obtain a *virtual state*. Each context view contains a workflow of contextualizing operations.
- Handle state queries. Clients can query a user global state, or (in most cases) trigger the creation of a virtual state to be queried.

A state is updated each time the client registers an event with the server. Such an event may be that the user's location has changed, or that he has clicked on a certain item. Moreover, a client can query a *virtual state*, which is created on-demand and is identified by the name of a context view. Finally, a client may ask for a *virtual global state*, which is a contextualized view on the global state of all active users; i. e. this virtual global state is generated by executing a workflow on the combined state of all users. By storing the states of all users on a central server, it is possible to make statements about a group of users and the group's context as a whole, especially in a collaborative scenario. Furthermore, since the actual contextualization process takes place server-side, the whole application logic is also placed there, leading to thin client implementations. In fact, the client in our implementation is basically limited to a thin wrapper around a SOAP-based communication protocol, i. e. a web service proxy without any functionality of its own. This makes it possible to benefit from sophisticated contextualization techniques even when performance considerations play a big role, for instance in a mobile environment on a smart phone. In the following sections the main functionalities of the contextualization server are illustrated in more detail.

State Management

One of the main purposes of the Hybreed Context Views framework is the management of states and virtual states of individual users and the entire group of active users. Whenever a client notices a change in the user's current state, it informs the server that the respective state needs to be updated (Figure 3).

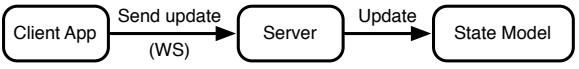

Fig. 3. In case of a client-side state update, the server updates the state model

Depending on the configuration, the event update may trigger follow-up actions on the server like inference operations or subsequent sensing activities.[2] In this way, the server always maintains a model of the current state, which in our implementation is stored as an RDF/OWL model using the Jena Framework[3]. Such an event-update message looks like this (simplified):

```
<sensor-id>AppSensor</sensor-id>
<statement>
 <subject>UserX</subject>
 <predicate>starts</predicate>
 <object>ChatApp</object>
</statement>
```

In order to retrieve and deduce statements about a group in its entirety, we introduce a so-called *global state*, which combines the states of all users into one that contains the sum of all this information, for instance by merging the sets of RDF statements of all active users. This is done each time the global state is requested and at least one of the user-specific states has changed; otherwise, we can fall back to a previously created and cached global state[4]. Another important consideration besides the performance aspect is, of course, multi-user synchronization; since multiple users can and will be active in the system at the same time and each one might want to access that global state while the other one is possibly just causing a change in it by updating his own state, it is imperative to synchronize the accesses in order to maintain thread safety. In our implementation, this is achieved by multiple locking mechanisms based upon the multiple readers / one writer approach.

Simple State Queries and Context-Aware Queries

Clients can now query their state to ask for non-contextualized information from the state layer (Figure 4). Following the database metaphor, this is like executing a query on a raw, materialized table within a database. In our case, all query strings must follow the SPARQL query format and can be SELECT, ASK, DESCRIBE or CONSTRUCT queries[5]. By querying the state, clients

[2] Example: Imagine, a system senses a user's current location by resolving his/her IP address to geographical coordinates. If a user now sends the event that the IP address has changed, the server should refresh the location as well.

[3] `http://jena.sourceforge.net`

[4] Preference aggregation by averaging single ratings is out of the scope of this paper.

[5] `http://www.w3.org/TR/rdf-sparql-query`

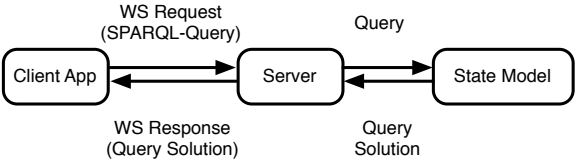

Fig. 4. A simple request

can not only request the information that they have previously supplied to the system. Instead, they can also query for information that was automatically inferred by the internal sensing engine (cf. "Components" section below), like a geographic location that was deduced from the IP address supplied by the client by contacting an appropriate external web service.

In context-adaptive scenarios, we can additionally provide the name of a context view as a request parameter in order to obtain a *virtual state*. This concept works for single users' states as well as for global ones.

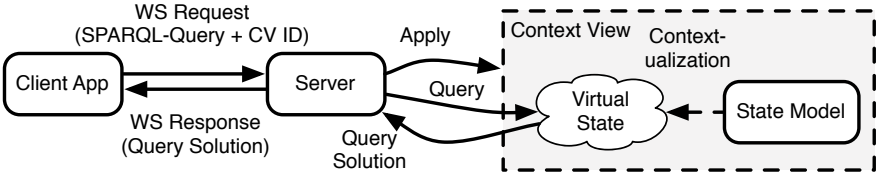

Fig. 5. Context-aware requests add a reference to a Context View (CV ID). The server then applies the appropriate Context View and queries the resulting virtual state.

Components

In figure 6, we see the main components of Hybreed Context Views. The ContextBroker and StateManager classes are the core of the system and are responsible for processing the users' requests as well as managing the states of the individual users. Since the framework can also be integrated into an existing Java application without necessarily exposing a web service endpoint, such an application can directly communicate with the central ContextBroker instance.

When an event is registered with the ContextBroker and is subsequently sent to the StateManager component, it passes the event to an implementation of the SensingEngine interface that is provided by the application. In essence, a sensing engine processes events, optionally inferences new information from these events and stores this information in the current user's state, which, in our case, is represented by a RDF model. As soon as the client intends to query a specific virtual state, the ViewEngine that is injected into the StateManager triggers a contextualization process: This class is provided with a set of context views by the application, each of which is identified by a unique name and links to

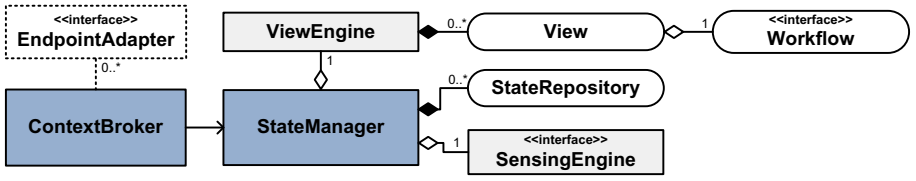

Fig. 6. Class diagram: Overview of the Hybreed Context Views architecture

an implementation of the Workflow interface that describes how a state is to be turned into a contextualized, virtual state (Exemplary workflows for context views are described in the next section).

4 Algorithms for Group Recommendations

The contextualization algorithms are not part of the Context Views framework; instead, a *workflow* interface is provided that can be implemented by any suitable contextualization framework (Figure 6). For demonstration purposes, we implemented a *Hybreed RecFlows*[6] library containing different recommendation algorithms ranging from item-based and user-based collaborative filtering, simple top-n or rule-based algorithms to different spreading activation techniques. Each algorithm is implemented in several variations and can be used in conjunction with a Context Views contextualization server. We now describe some of the algorithms in detail.

Rule-based. The example in Section 2 used a simple rule-based approach to highlight the most relevant elements. Personalization by means of fixed rules often is a good trade-off between administration and implementation effort on the one hand and good results on the other.

Spreading Activation. Spreading Activation was introduced in the 1970s [1] and has successfully been used in several research areas in computer science, most notably in information retrieval or e-commerce [6]. The basic concept behind Spreading Activation is that all relevant information is mapped on a graph as nodes with certain "activation levels". Relations between two concepts are represented by a link between the corresponding nodes. If for any reason one or more nodes are activated, their activation level rises and the activation is spread to adjacent nodes (and the ones related to them and so on). Thereby the flow of activation is attenuated the more it strides away from the initially activated node(s). In the end, several nodes are activated to a certain degree that are semantically related to the elements originally selected.

This technique can directly be adapted to context views as all relevant information already is encoded in a graph. If, for instance, user A and B attend a chat, the nodes representing A and B could be initially activated. The activation

[6] RecFlows stands for "Recommendation Workflows".

would now spread through the graph, increasing adjacent nodes. As a result, exactly those elements that A and B have in common (for instance a project both are working on) would receive the most activation.

Collaborative Filtering. Collaborative filtering [3] methods are supposed to be the most widely implemented recommendation techniques. Transferred to context views and collaborative work, users could be regarded as belonging to the same group, if they have certain features in common. In this case, instead of items in a shopping cart, recently opened documents could for instance be the foundation to compute similarity upon, resulting in something like "users who opened Roadmap for Project X also opened Work Packages of Project X".

Hybrid approaches. Hybrid recommender systems can be applied as well (in fact we did just that in a previous publication [5]). For instance could both algorithms mentioned above be combined by first applying collaborative filtering to identify certain elements and, in a second step, using the the results as initial nodes in a spreading activation process for refining.

These approaches only sketch the basic ideas in order to demonstrate the generic applicability of context views regardless of the algorithm. In reality, things like privacy or rights management have to be considered as well, but the principles and ideas of context views should be clear now.

5 Summary

The definition "any information that can be used to characterize the situation of an entity [is context]" [2] leads to the implication that for instance the membership in a certain group can be regarded as context as well – which affects the notion of context-adaptive systems in turn: Not only should context-adaptive systems exploit external observations like time, location, etc., but also take any other information into account "that can be used to characterize the situation".

We introduced *context views*, which can be used to identify the most important elements of a situation with regard to the particular contextual perspective. A system can thereupon use them for adaptation purposes. In Section 3, we described an architecture for this conceptual framework: We implemented a contextualization server that can make use of arbitrary personalization techniques to generate context-aware group recommendations as a service. After that, we explained how several well-established techniques can be used for context views, either solely or in combination.

Context views can be used in group-based scenarios, but in fact they are meant as a more general concept for arbitrary context-adaptive systems. However, in this paper, we focused on the derivation of the concept and its implementation as well as its applicability for group-based work. This paper focused on the construction of shared preference models (identified as (c) according to Jameson & Smyth [8]). Although (a) and (b) are possible with this framework and architecture as well, we skipped the details for the sake of simplicity, and will publish our experience with these approaches in a different publication.

Acknowledgements. The research presented in this paper is part of the CONTici project funded by the German Research Foundation (Deutsche Forschungsgemeinschaft).

References

1. Collins, A.M., Loftus, E.F.: A spreading activation theory of semantic processing. Psychological Review 82(6), 407–428 (1975)
2. Dey, A.K., Abowd, G.D.: Towards a better understanding of context and context-awareness. In: CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness. ACM Press, The Hague (2000)
3. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Communications of the ACM 35(12), 61–70 (1992)
4. Hussein, T., Gaulke, W., Linder, T., Ziegler, J.: Improving collaboration by using context views. In: Proceedings of CAICOLL: First Workshop on Context-Adaptive Interaction for Collaborative Work (2010)
5. Hussein, T., Linder, T., Gaulke, W., Ziegler, J.: Context-aware recommendations on rails. In: Workshop on Context-Aware Recommender Systems (CARS-2009) in conjunction with the 3rd ACM Conference on Recommender Systems (ACM RecSys 2009), New York, NY, USA (2009)
6. Hussein, T., Ziegler, J.: Adapting web sites by spreading activation in ontologies. In: Proceedings of International Workshop on Recommendation and Collaboration. ACM, New York (2008)
7. Jameson, A.: More than the sum of its members: Challenges for group recommender systems. In: Proceedings of the International Working Conference on Advanced Visual Interfaces, pp. 48–54. Gallipoli, Italy (2004)
8. Jameson, A., Smyth, B.: Recommendation to groups. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web: Methods and Strategies of Web Personalization, pp. 596–627. Springer, Berlin (2007)
9. Resnick, P., Varian, H.: Recommender systems. Commun. ACM 40(3) (1997)