Recommender Systems for the People — Enhancing Personalization in Web Augmentation [*]

Martin Wischenbart
CIS, Johannes Kepler University
Linz, Austria
martin@cis.jku.at

Sergio Firmenich,
Gustavo Rossi
LIFIA, Universidad Nacional de La
Plata and CONICET, Argentina
[firstname].[lastname]
@lifia.info.unlp.edu.ar

Manuel Wimmer
BIG, Vienna University of
Technology, Austria
wimmer@big.tuwien.ac.at

ABSTRACT

Web augmentation techniques allow the adaptation of websites on client side using browser extensions or plug-ins designed to run dedicated user scripts. However, while number and variety of such scripts from publicly available repositories have grown remarkably in recent years, they usually neglect the user's personal profile or individual preferences, and therefore fail to provide enhanced personalized services. At the same time recommender systems have become powerful tools to improve personalization on the Web. Yet, many popular websites lack this functionality, e. g., for missing financial incentive. Therefore, we present a novel approach to empower user script developers to build more personalized augmenters by utilizing collaborative filtering functionality as an external service. Thus, script writers can build recommender systems into arbitrary websites, in fact operating across multiple website domains, while guarding privacy and supplying provenance information. This paper discusses the architecture of the proposed approach, including real-world application scenarios, and presents our tool kit and publicly available prototype. The results show the feasibility of combining Web augmentation with recommender systems, to empower the crowd to build new kinds of applications for a more personalized browsing experience.

1. INTRODUCTION

In recent years web *augmentation* techniques, i. e., the addition of external content or behaviour to Web pages, have become a popular means for *end users* to adapt pages according to their own requirements, with reduced dependency on the website provider. Thereby, advanced users with knowledge of JavaScript, so-called scripters, write (user) scripts to modify web pages, which are then executed within the browser on client side – using dedicated browser plug-ins such as GreaseMonkey[1], and without the inclusion of the sites' webmasters. These user scripts are often publicly shared, and to date there are several large repositories providing a vast amount of various scripts for all kinds of Web pages and modification tasks. For instance, GreaseFork[2] has more than six thousand scripts, some of which are installed more than fifty thousand times. Another well-known repository, called UserScripts[3], hosts more than one hundred thousand scripts. Examples range from layout modification and tweaks (e. g., regarding video player size, video & audio customizations, etc.) on `youtube.com`[4], managing comments on `geocaching.com`[5], to improving navigation on `dropbox.com` by rendering a Tree View panel[6].

Unfortunately, as pointed out by a recent survey [8], and according to our own experience, current technologies for adapting the Web browsing experience still do not sufficiently support individual *personalization*, as it is provided by applications incorporating recommender system functionality. Consequently, with a single Web augmentation artifact (i. e., script) every user has the same experience.

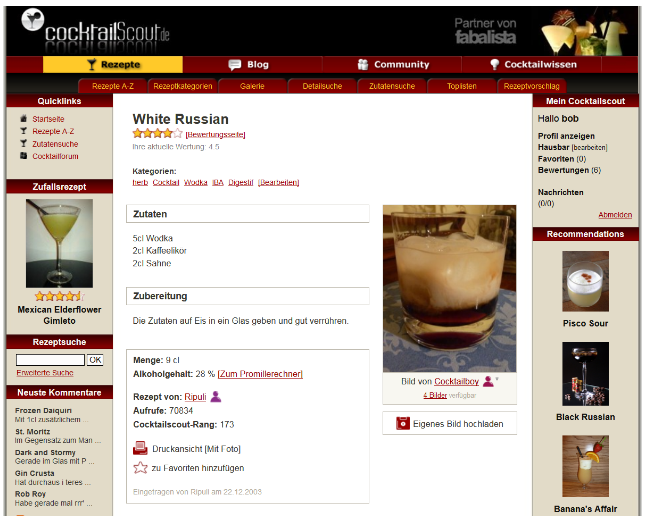
Recommender systems meanwhile have a longer standing history in various domains, such as e-commerce or music recommendations, and have become one of the most popular ways to personalize services and user experience. Commonly, they are classified [17] into content-based, collaborative, knowledge-based, as well as hybrid approaches. Although all these approaches rely on a user model, they differ in how they build this model, and they exploit different kinds of additional information and algorithms for presenting personalized recommendations of items to the user. For instance, collaborative approaches, also known as collaborative filtering or community-based, take into account the opinions of large amounts of users to make predictions about a specific user's preferences for items. Despite their potential, however, oftentimes websites do not implement recommendation services, either because of lack of economic incentives, or simply for lack of know-how on these techniques.

To alleviate these problems, we aim to introduce recommender system functionality for enhancing personalization in Web augmentation, combining the benefits of both approaches. To illustrate the use of collaborative filtering

[*]This work has been funded by grants Mincyt-BMWF AU/14/02, AU/14/09, ÖAD AR13/2015, ICM-2013-06624.

[1]`http://www.greasespot.net` (Firefox)
[2]`http://greasyfork.org`
[3]`http://userscripts-mirror.org`
[4]`http://greasyfork.org/es/scripts/943-youtube-center`
[5]`http://userscripts-mirror.org/scripts/show/75959`
[6]`http://greasyfork.org/es/scripts/4955-dropbox-plus/code`

for realizing Personalization in Web Augmentation Applications (PAA) we exhibit an example: The website `cocktailscout.de` is one of largest German language websites for cocktail recipes[7], having a community of almost 4000 registered users, who can search, rate and comment recipes for drinks. A rating mechanism is used for ranking of recipes, and averages and distributions of ratings for each drink can be viewed. Yet the users' individual ratings are not exploited to give them personalized recipe recommendations. Instead, the site provides a random link to a recipe on each page. Containing more than 1500 recipe items, however, the site is a perfect target for implementing a collaborative filtering recommender system. Based on the tastes of similar users, personalized recommendations for drinks could be presented in one of the sidebars, as shown in Figure 1.



Mexican Elderflower Gimleto

Frozen Daiquiri
Mit 1cl zusätzlichem ...
St. Moritz
Im Gegensatz zum Man ...
Dark and Stormy
Gerade im Glas mit P ...
Gin Crusta
Hat durchaus i teres ...
Rob Roy
Habe gerade mal rrr' ...

White Russian
[Bewertungsseite]
Ihre aktuelle Wertung: 4.5

Kategorien:
herb Cocktail Wodka IBA Digestif [Bearbeiten]

Zutaten
5cl Wodka
2cl Kaffeelikör
2cl Sahne

Zubereitung
Die Zutaten auf Eis in ein Glas geben und gut verrühren.

Menge: 9 cl
Alkoholgehalt: 28 % [Zum Promillerechner]

Rezept von: Ripuli
Aufrufe: 70834
Cocktailscout-Rang: 173

Druckansicht [Mit Foto]
zu Favoriten hinzufügen

Eingetragen von Ripuli am 22.12.2003

Bild von Cocktailboy
4 Bilder verfügbar

Eigenes Bild hochladen

Hallo bob
Profil anzeigen
Hausbar [bearbeiten]
Favoriten (0)
Bewertungen (6)

Nachrichten
(0/0)

Pisco Sour
Black Russian
Banana's Affair

Figure 1: Adopted `cocktailscout.de` website with personalized recommendations augmented (right).

Typically, the implementation of such functionality heavily depends on the *website provider*, and in the world of online shopping there is a lot of incentive for service providers to introduce such recommenders for increasing sales and profit. In contrast, for non-commercially oriented websites, or when items and monetization are not related directly, as with cocktail recipes, there is no such incentive for the site provider. For end users, however, recommender system techniques are commonly *beyond* the scope and to complex to be employed. Nevertheless, users of the CocktailScout website who are hobby JavaScript programmers, might be interested, and have the skills to implement a user script for utilizing the ratings and adding personalized recommendations of recipes. They do, however, lack access to a recommender system providing them with item recommendations.

Such a system could theoretically be implemented as a content-based recommender on client-side, given a catalog of items with features (such as a drink's ingredients), and the user's interests (such as preferences for ingredients). If no such item catalog is as available, however, the alternative, namely to collect all possible items manually in the Web augmentation script, seems to be a tedious task. For collaborative filtering, in contrast, such explicit *domain knowledge* is not required, but instead, only explicit or implicit user ratings (i. e., weighted relations between users and items) are required. Therefore, for the PAA approach we propose to share ratings and compute recommendations using collaborative filtering on a dedicated server, not least because of the current trend to cloud services, and protection of users' privacy. Consequently, providing a RESTful API for standard HTTP requests (and a corresponding object-oriented library in JavaScript, the prevailing language in the Web augmentation community), a recommendation service can be offered to the exemplary cocktail-drinking hobby JavaScript programmer. Providing a generic service, the approach has the potential to reach and benefit a large existing web augmentation community, and can be employed for *arbitrary websites*, under control of users and on client-side, and it may even go beyond single domains or the scope of a single site provider. Thereby, providing a simple and clear API and complete documentation is a key requirement, as we also discovered in the context of the composition language for building personalized recommenders and services in our research project TheHiddenYou [19].

In the upcoming Section 2 we discuss related work in the context of our approach. The following Section 3 presents an overview of the approach, including the tasks for scripters. Next, Section 4 introduces several real-world application scenarios, and demonstrates the approach as seen by the users. Section 5 elaborates on the architecture in detail, in particular the server part, and the PAA-API for scripters, which provides the functionality to store and retrieve item ratings, and to retrieve rating predictions and recommendations. In this context important issues are privacy and provenance, issues that are commonly disregarded in the Web augmentation community, but gain importance in the light of a central repository for collecting ratings. Finally, Section 6 discusses our prototype and some practical issues, before Section 7 presents conclusions and future work.

2. RELATED WORK

This section discusses related research in several fields, such as adaptive hypermedia, Web personalization, recommender systems, as well as Web augmentation, and finally compares our approach to similar approaches which are used in practice.

Research on Web personalization has been steadily growing, and in order to satisfy the huge number of end-users, several approaches for personalizing Web content have emerged, e.g. user profiling for personalization [14], or recommender systems [17]. In this context different ways for rating have been studied [24], and classifications of user feedback have been surveyed (including their correlation to ratings) [18].

Although usually recommenders work on server-side, some approaches for client-side personalization have been developed [2, 16]. In such scenarios, since different Web applications can share a single user profile (e.g. managed on client-side using a browser extension), and recommendations may cover different sites. Regardless whether personalization mechanisms work on server or client side, these mechanisms are usually specified by website owners, and they are always limited by the information available on the user profile. Meanwhile browser extensions monitoring user navigation can be used to populate user profiles (with navigation history, bookmarks, keywords, etc.) and thereupon recom-

[7]Highest global Alexa rank (`http://www.alexa.com/`) in a comparison of 13 German language cocktail websites.

mend relevant Web pages to users [10, 12]. Although there are some works aiming to define and extract [25] comprehensive profiles, and analyze their interoperability [5], it is difficult to implement an adaptation mechanism which is broad enough to contemplate every user requirement, especially while protecting privacy [20] and providing provenance information [21].

Web augmentation techniques are another way to achieve personalization; augmentation allows users to customize website user interfaces (UIs) in terms of content and functionality, according to their own requirements [11]. Most Web augmentation approaches are developed as browser extensions, and once installed by the user, they modify loaded Web pages, thus altering what the user perceives. In this way, end users with programming skills are the ones creating Web augmentation artifacts. However, most recent research about Web augmentation do not target personalization, but aim to provide tools (frameworks, or languages) to solve domain-specific adaptations (i.e., support recurrent tasks, automate tasks, improve accessibility, etc.) or raise the abstraction level in order to allow more users (without advanced programming skills) to specify how they want to augment their preferred Web sites. For instance, CSWR [13] aims to improve Web accessibility, and WebMakeUp [9] allows end-users to specify their own augmentations. All these approaches propose a way to customize the Web, but most of them work without an underlying user profile. Web augmentation may be employed for guiding the user through content, whereas the navigation mechanisms are not implemented by the content provider himself. Using a collaborative system for recommending items on the Web, in this context, represents a 'social mechanism' with an 'open corpus of documents'.

A similar kind of systems to adapt existing third-party Web content are intermediaries [3], which intercept the content in a proxy server and not on client-side. From our point of view, and in comparison with intermediaries, Web augmentation approaches are usually more powerful as adaptation mechanisms. Web augmentation tools usually extend the Web Browser, and consequenlty these tools give more information about the users activity than a those systems working on a proxy server.

Some authors have proposed personalization as a service [15], and nowadays there are several companies offering rating[8] and recommendation[9] as Web services. Despite similarities to the proposed PAA approach in terms of the employed technology, these approaches require changes in the "original" Web site, and these changes need to be performed by the provider, for instance, by integrating a JavaScript library. Furthermore, these approaches require an upload of a complete product catalog beforehand[9], thereby causing additional maintenance effort. In our proposed PAA approach, this catalog is built on-the-fly, i.e., product details are pushed to our repository alongside with ratings. Finally, whereas in recommendation as a service, concrete recommendations are rather generic; we argue that a scripter who is an active member from both Web site community and Web augmentation community may have further insight to exploit domain knowledge about human decision making in that community (cf. [6]) for giving item recommendations.

[8]Rating-Widget: `http://rating-widget.com/`
[9]Strands: `http://retail.strands.com/`

3. THE APPROACH IN A NUTSHELL

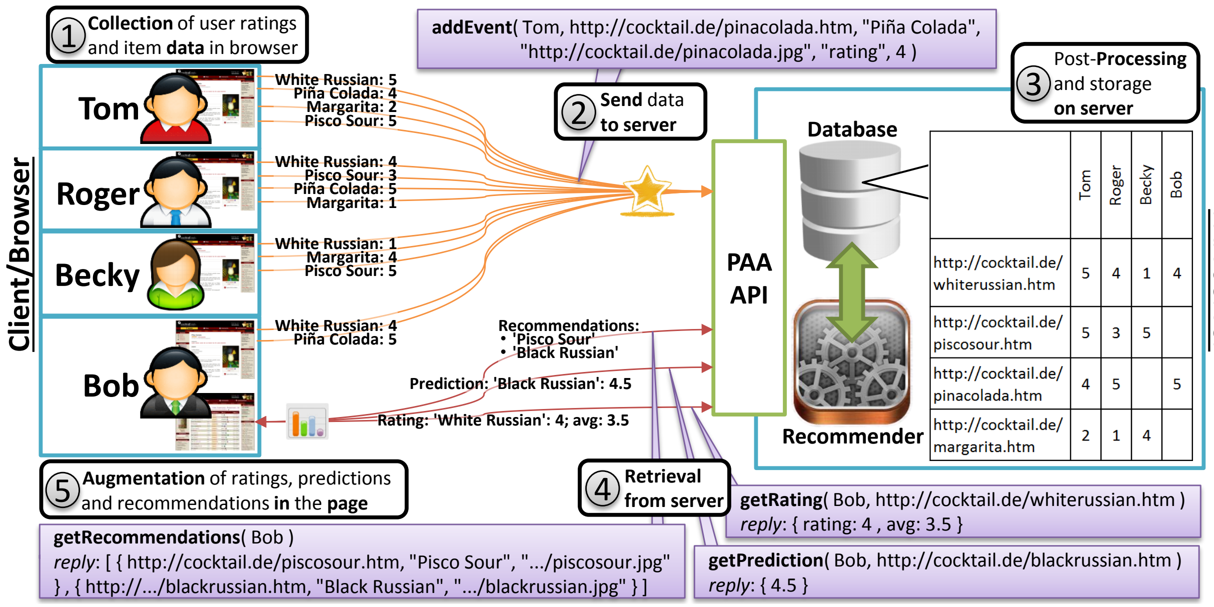
This section gives an overview of the approach, in particular as seen by the script writer, referring to the example outlined in the introduction. The complete architecture and technical details about the server will be explained in Section 5. In short, the PAA approach supports the scripter to make adaptations to the website, as outlined in Figure 2. At first ratings are collected in the browser by the scripter ① and sent to the server using dedicated API methods ②. After processing on the server ③, another set of methods may be used to retrieve previous ratings, rating predictions, as well as recommendations ④, to be finally employed to modify the page ⑤, for instance for link ordering, link hiding, link annotation, or link generation, as classified by Brusilovsky [4]. In the following, the five steps are outlined in detail.

Data Collection ①: Scripters may rely on an existing rating mechanism to measure the user's interest in an item. If no such mechanism exits, it can be implemented by the scripter, either in terms of an explicit rating (e. g., 1-5 stars, like vs. dislike, etc.) or implicitly computing a score (e. g., based on page visit, time spent on the page, activities such as posting comments or uploading pictures). For modeling the user's interest, we rely on *events* relating *users* with *items* (both identified with unique IDs) and including a numeric score for *rating*. For presentation purposes later on (cf. ⑤), here we also require additional features about an item, including a human readable *name*, or *meta-info* such as an image, to be shown as link in the web browser. For the use-case regarding cocktails, this means we will need the drink's URL, its name, an image URL, as well as a user ID and the numeric rating from the page. *Tasks for the scripter:* To extract all this information, the scripter usually reads the Web page's document object model (DOM): 1. Extract a unique user ID from DOM (or rely on user's login on the PAA server's web interface; cf. Sect. 5); 2. Extract a unique item ID from DOM; 3. Extract further information about the item from DOM, such as name (to be used as link text), or links (to be used for image links); 4. Extract the rating value, or compute a numeric rating from collected explicit or implicit user feedback (or rely on default scores of predefined event types; cf. Sect. 5).

Send to Server ②: As a next step, the previously collected data must be *sent* to the server, to ultimately *collect* a large number of such events as basis for the recommender algorithm. *Tasks for the scripter:* Send data to the server via the API, using our provided JavaScript library (parameter string, or object-oriented), or using HTTP POST requests.

Processing on Server ③: On the server events are stored, a *timestamp* is added, and they are processed to be in a format for being used by the recommender (cf. Sect. 5).

Retrieval from Server ④: As a next step, queries may be performed from the script, either for any page on the site or pages representing or containing items (i. e., automatically), or on demand of the user (i. e., manually, cf. 'pull recommendations' [23]). The following data can be retrieved from the server: firstly, previously stored ratings including meta-information, such as average ratings and their distributions; secondly, predictions for user ratings; and finally, recommendations for items, with the latter two being *computed on-the-fly* using a recommender system library on the server. *Tasks for the scripter:* Via several dedicated API methods, the scripter can query this information from the server, again using our provided JavaScript library or HTTP



addEvent(Tom, http://cocktail.de/pinacolada.htm, "Piña Colada", "http://cocktail.de/pinacolada.jpg", "rating", 4)

	Tom	Roger	Becky	Bob
http://cocktail.de/whiterussian.htm	5	4	1	4
http://cocktail.de/piscosour.htm	5	3	5	
http://cocktail.de/pinacolada.htm	4	5		5
http://cocktail.de/margarita.htm	2	1	4	

getRating(Bob, http://cocktail.de/whiterussian.htm)
reply: { rating: 4 , avg: 3.5 }

getRecommendations(Bob)
reply: [{ http://cocktail.de/piscosour.htm, "Pisco Sour", ".../piscosour.jpg" } , { http://.../blackrussian.htm, "Black Russian", ".../blackrussian.jpg" }]

getPrediction(Bob, http://cocktail.de/blackrussian.htm)
reply: { 4.5 }

Figure 2: Overview of the five steps of the proposed PAA approach as seen by end users and script writers, including a simplified representation of the communication between client and server.

POST requests.

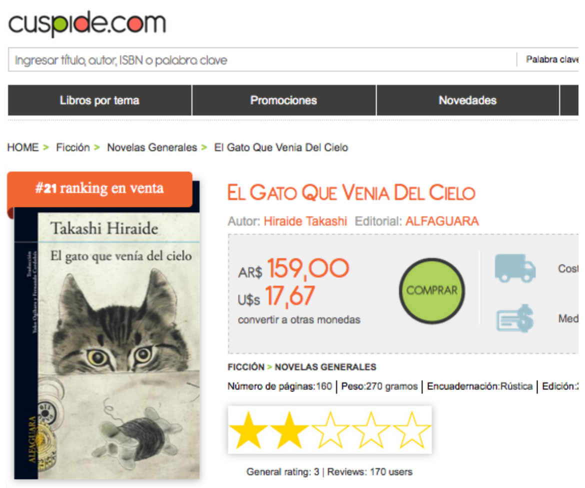
Augmentation in Page ⑤: The previously retrieved information may finally be augmented in the page. Previous and average ratings as well as their distributions may be shown as additional information on the item page, or for *link annotation* (cf. [4]), e. g., as popups for all links referring to drinks. Rating predictions may be employed for *re-ordering* items or links on the page, or, if the predicted score is below a certain threshold, for *hiding* a link (cf. [4]). Finally, the provided list of recommendations can be used to *generate* (cf. [4]) personalized links on the page, referring to items the user could be interested in, e. g., as a list of recommended drinks. *Tasks for the scripter:* The desired results are displayed on the page by modifying the DOM. For showing drink recommendations, this can be achieved by using the response from the server to add elements and set their properties. The complete script source code for this example is available online with our prototype (cf. Sect. 6).

4. APPLICATION SCENARIOS

This section elaborates on several user script use-cases that can be realized with our approach, and focuses on benefits for the end user.

Cocktail Recipes – Recommendation Scenario. In the previous sections and example we based on the example of `cocktailscout.de`, a site which has almost 4000 registered users and more than 1500 recipe items, and an existing rating mechanism. Only by exploiting these ratings on the PAA server, personalized drink *item recommendations* can be *generated* for the user.

Bookstore – Rating Mechanism. The Argentinean online bookstore `cuspide.com` provides functionality to comment on books, but does neither offer ratings nor personalized recommendations. In such scenarios without a re-usable rating mechanism, a rating widget may be added to the page (cf. Fig. 3), to show items *annotated* with previous ratings

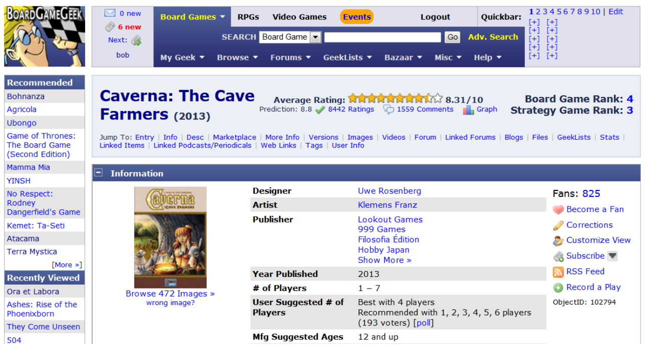


cuspide.com
Ingresar título, autor, ISBN o palabra clave
Palabra clave
Libros por tema
Promociones
Novedades
HOME > Ficción > Novelas Generales > El Gato Que Venia Del Cielo
#21 ranking en venta
Takashi Hiraide
El gato que venía del cielo
ALFAGUARA
EL GATO QUE VENIA DEL CIELO
Autor: Hiraide Takashi Editorial: ALFAGUARA
AR$ 159,00
U$s 17,67
convertir a otras monedas
COMPRAR
Cost
Med
FICCIÓN > NOVELAS GENERALES
Número de páginas:160 | Peso:270 gramos | Encuadernación:Rústica | Edición:
General rating: 3 | Reviews: 170 users

Figure 3: Example modified bookstore website with an augmented rating mechanism (bottom).

and statistics, as well as to *generate* personalized recommendations. For the scripter this scenario is slightly more complicated, since it requires the definition of what items can be rated in the first place, i. e., a definition of items to be processed by the user script.

Board Games #1 – Using Predictions. In addition to annotation and generation of links, to improve the visibility of relevant items for the user, *rating predictions* may be exploited for link *hiding* and *re-ordering*. This is particularly useful for sites with a large catalog of items, such as the board gaming community website `boardgamegeek.com`, with around 77.000 games grouped by publishers, artists, as well as various categories and families (and more than one million of registered users). Even though a rating mecha-


BoardGameGeek

0 new · 6 new · Next: · bob

Board Games · RPGs · Video Games · Events · Logout · Quickbar: 1 2 3 4 5 6 7 8 9 10 | Edit

SEARCH Board Game · Go · Adv. Search

My Geek · Browse · Forums · GeekLists · Bazaar · Misc · Help

Recommended
- Bohnanza
- Agricola
- Ubongo
- Game of Thrones: The Board Game (Second Edition)
- Mamma Mia
- YINSH
- No Respect: Rodney Dangerfield's Game
- Kemet: Ta-Seti
- Atacama
- Terra Mystica

[More »]

Recently Viewed
- Ora et Labora
- Ashes: Rise of the Phoenixborn
- They Come Unseen
- 504

Caverna: The Cave Farmers (2013)

Average Rating: 8.31/10
Prediction: 8.8 · 8442 Ratings · 1559 Comments · Graph

Board Game Rank: 4
Strategy Game Rank: 3

Jump To: Entry | Info | Desc | Marketplace | More Info | Versions | Images | Videos | Forum | Linked Forums | Blogs | Files | GeekLists | Stats | Linked Items | Linked Podcasts/Periodicals | Web Links | Tags | User Info

Information

Browse 472 Images »
wrong image?

Designer	Uwe Rosenberg
Artist	Klemens Franz
Publisher	Lookout Games 999 Games Filosofia Édition Hobby Japan Show More »
Year Published	2013
# of Players	1 – 7
User Suggested # of Players	Best with 4 players Recommended with 1, 2, 3, 4, 5, 6 players (193 voters) [poll]
Mfg Suggested Ages	12 and up

Fans: 825
- Become a Fan
- Corrections
- Customize View
- Subscribe
- RSS Feed
- Record a Play

ObjectID: 102794

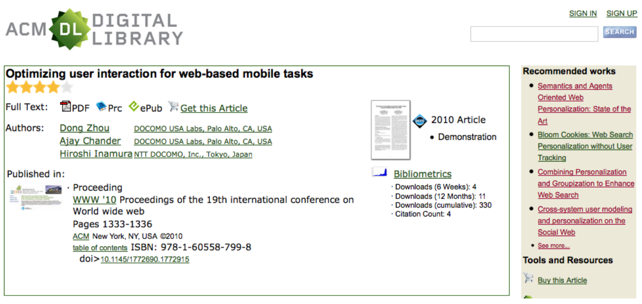
Figure 4: Example modified BoardGameGeek website with augmented recommendations (left) and rating prediction (center/top).

nism exits, personalized recommendations of games are not given to users. Instead, the site displays a list of 50 currently 'hot' games. To personalize this list using the PAA approach, e. g., it can be re-ordered according to rating predictions for these items and users, and games having a particularly bad prediction (e. g., below a certain threshold) can be hidden.

Board Games #2 – Feedback & Event Types. As an alternative to the previous scenarios, where users are required to give explicit ratings, other kinds of user feedback may be exploited, such as explicit feedback without numeric ratings, or implicit user feedback. On `boardgamegeek.com`, besides explicit ratings, users can become fan of a game, subscribe, tag, record plays, add games to collections, and much more. In addition to these explicit events, implicit feedback may be recorded directly in the browser (i. e., behaviors exhibited by the user while using the site; cf. [18] – PAA does not require a special browser for this functionality, but it can be implemented as scripts). Based on this, a script can compute a single numeric rating value to be communicated to the server. Alternatively, to relieve the scripter of this task, we further propose a mechanism to allow putting multiple events to the server, and compute an aggregated single rating value to estimate the user's interest in an item on server-side. Thus, for the scripter the task of computing a rating value is broken down to recording different user actions (i. e., events), and defining how the events should be accumulated to a collective score (sum, average, median, logarithmic, etc.). Details about how this is achieved will be explained in detail below in Section 5.

Scientific Literature – Cross Domain. Finally, since Web augmentation scripts can be defined to run on multiple websites, going beyond the scope of a single content provider, the PAA approach enables personalized link recommendations across multiple domains. This feature is specially useful for those Web sites sharing an underlying domain. For instance, for scientific literature search, a script can track user activities such as page visits and downloads on sites such as ACM DL, Springer, IEEE Xplore, or Science Direct[10]. In every site of this list, the same item (Paper) may be defined with similar properties (url, title, authors, abstract). Based on events collected from these user activities, a recommendations pane can be added to each of these websites to present potentially relevant scientific literature

[10] `http://dl.acm.org/`, `http://link.springer.com/`, `http://ieeexplore.ieee.org/`, `http://www.sciencedirect.com/`


ACM DL DIGITAL LIBRARY

SIGN IN · SIGN UP · SEARCH

Optimizing user interaction for web-based mobile tasks

Full Text: PDF · Prc · ePub · Get this Article

Authors:	Dong Zhou	DOCOMO USA Labs, Palo Alto, CA, USA
	Ajay Chander	DOCOMO USA Labs, Palo Alto, CA, USA
	Hiroshi Inamura	NTT DOCOMO, Inc., Tokyo, Japan

Published in:
- Proceeding
  WWW '10 Proceedings of the 19th international conference on World wide web
  Pages 1333-1336
  ACM New York, NY, USA ©2010
  table of contents ISBN: 978-1-60558-799-8
  doi>10.1145/1772690.1772915

2010 Article
- Demonstration

Bibliometrics
- Downloads (6 Weeks): 4
- Downloads (12 Months): 11
- Downloads (cumulative): 330
- Citation Count: 4

Recommended works
- Semantics and Agents Oriented Web Personalization: State of the Art
- Bloom Cookies: Web Search Personalization without User Tracking
- Combining Personalization and Groupization to Enhance Web Search
- Cross-system user modeling and personalization on the Social Web
- See more...

Tools and Resources
- Buy this Article

Figure 5: Example modified literature search website with augmented recommendations (right).

from any of the sites to the user, as shown exemplary in Figure 5.

5. ARCHITECTURE

As it was explained in the introduction, on client-side dedicated plug-ins enable the execution of user scripts to manipulate the DOM in the browser. Such user scripts, written in JavaScript language, may execute HTTP requests to external RESTful APIs. This is shown in Figure 6, alongside with the proposed PAA architecture and components of the server, which will be explained in more detail in the following. In PAA, requests can be made manually using standard scripting tools (e. g., functions provided by Firefox or GreaseMonkey), or using our JavaScript library, which provides functions for sending parameters concatenated as a single strings (e. g., `userName=Bob&rating=4&...`), or in an object-oriented manner. The latter option is shown exemplary in Figure 7, including code to extract the parameters from the DOM, and the interal library implementation is presented in Figure 8.

Processing of Events. As mentioned in step ③ in Section 3, events that are stored on the server are equipped with a timestamp, before all parameters are checked for validity, and events are stored to the database. Furthermore, provided `userName`s and item IDs (`itemIri`[11]) are mapped to numeric IDs, and stored along with the `rating` and the timestamp in a separate table to be accessed directly by the recommender engine.

Different Types of Events. To distinguish between different types of events, such as explicit numeric ratings and other kinds of feedback, the scripter can choose to supply a custom `eventType` (e. g., `'pictureUpload'`). Except for the distinction, this allows us to provide a default rating value for several pre-defined event types (following a classification of observable user behaviour found in literature [18]). These default ratings, however, are currently assigned in unscholarly manner, and thus can also be overridden by the scripter.

Rating Accumulation. Whereas events represent user actions, for the recommender engine we rely on ratings only. For this, the rating values provided with the events may be used directly. Defined by a parameter `insertAs`, first, events may simply be added as `'new'` ratings (i. e., several ratings per user for a single item are provided to the recommender engine). Second, the rating specified by the event may `'replace'` the previous one. Finally, as an alternative multiple events may be accumulated – in a way that can be configured by the scripter – to compute the rating value to

[11] commonly URIS/IRIS (`http://tools.ietf.org/html/rfc3987`)

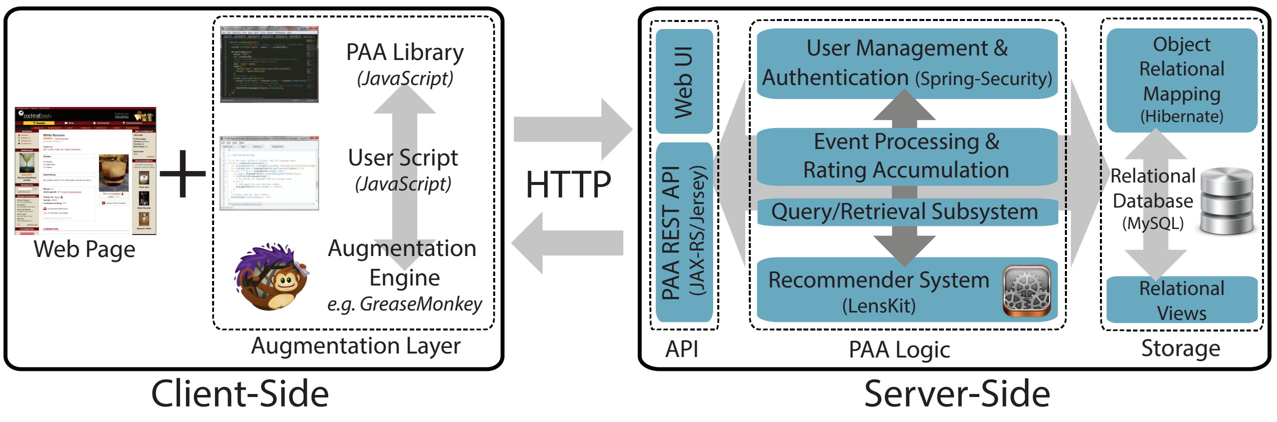


Figure 6: Architecture of the proposed PAA approach: client-side, server-side, communication via HTTP.

```
item_name = document.querySelector(".recipe-detail..."
item_iri = window.location.pathname;
```
Item data obtained from DOM

```
item = paa.item(item_name, item_iri);
item_img = document.querySelector(".recipe-image").src;
item.addMetaData("img",item_img);
```
Item creation and initizialization

```
username = document.querySelector(".mycs_username..."
user = paa.user(username);
```
User creation (username obtained from the DOM)

```
user_rating_value = document.querySelector(".curr..."
user = paa.rating(user_rating_value);
```
Rating creation (value obtained from DOM)

```
event = paa.event(user, item, rating);
paa.sendEvent(event);
```
Event creation and sending to server

```
paa.getRecommendations(user,receiveRecommendations);
```
Obtain recommendations and handle the result with *receiveRecommendations()*

Figure 7: Example JavaScript code for creating and storing an event object, and for retrieval of recommendations using the object-oriented PAA library.

```
getRecommendations:function (user, callback) {
  var pluginName = GM_info.script.name;
  var pluginNamespace = GM_info.script.namespace;
  var domain = window.location.host;
  var recsData = this.serializeRecommendation(user,
                pluginName,
                pluginNamespace,
                domain)
  GM_xmlhttpRequest({
        method: "POST",
        url: this.getRecommendationsURL,
        data: recsData,
        headers: {
          "Content-Type": "application/x-www-form...",
          "Accept": "application/json"
        },
        onload: function(response) {
              callback(response)
        } }); }
```
Define and serialize data (current script, website, user) for requesting recommendations from the server

Perform the request and execute the callback function (defined by the scripter) to handle the server's response (i.e., recommendations)

Figure 8: Example internal JavaScript library function of client-side API to request recommendations.

be used by the recommender. Since user scripts are typically run once per page load, this may be a particularly useful option to track implicit user feedback in terms of potentially many events regarding a single item. For this accumulation, ratings may be derived as '`sum`', '`average`', or '`median`' of event scores. While summing or averaging, weights for older events may be decreased, e. g., linearly or logarithmic. Furthermore, the `aggregationScope` can optionally be limited to a certain number of most recent events, or using a time window. As a result, the rating scale must allow float values, and consequently also the ratings provided from the user script can be continuous, allowing different scales or binary ratings. For the future we are planning extensions, and we aim to make this computation more customizable, for instance, to enable that different event types contribute to the accumulation in different manners.

Authentication & Web UI. User authentication is not required if while making API calls the user script provides a `userName`. Clearly this is security issue, because in this basic form there is no authentication involved. This option was enabled for the sake of simplicity, and for simple scenarios where security is not an issue. Alternatively, instead of providing a user name, the parameter `paaAuthRequired` can be set to `true`, enabling authentication via the PAA website (the current status can be queried using `getLoginStatus()`, to open the login form if necessary). Once logged in, the corresponding cookie is being included by the browser when making calls to the server's API. Via the Web user interface, users can furthermore manage their profile data and view coarse grained provenance information. Thereby, the user may be provided with insights to what data is being stored and how it is used, including information on why specific recommendations were given. This increases transparency and helps to gain the users' trust, and to satisfy users who might normally be reluctant to share personal data with providers of (commercially oriented) personalization services.

Recommender Engine on Server. For adding events, as well as for queries to the recommender engine, every API query must provide the script's `pluginName`, `pluginNamespace`, and the site `domain`. These define the operation domain for the recommender, i. e., the view for ratings that are seen by the recommender engine (also user names must be unique within this operation domain). For computing predictions and recommendations the current generic prototype implementation relies on the open-source framework LensKit[12], configured to perform item-item collaborative filtering for scoring items (cf. [7, 22], LensKit doc.[13]), and directly accessing the database for building models.

Provision to Client. Finally, for answering queries regarding ratings, predictions, and recommendations, again user names and item IDs are mapped to numeric IDs and vice versa, and previously stored `itemNames` and `meta` infor-

[12]`http://lenskit.org/` – Recommender Toolkit
[13]`http://lenskit.org/documentation/algorithms/item-item/`

mation (stored per event) are added where applicable. Example responses are shown online along with the prototype, as discussed in the following section.

6. PROTOTYPE & EVALUATION

The implemented PAA prototype is publicly available online[14], including our object-oriented JavaScript library with example scripts and instructions for their usage. The implementation bases on Jersey[15] for handling RESTful HTTP requests, and employs Spring Security[16] for authentication, which also allows the future integration of identity providers such as OpenID (supported by Google) or Facebook Connect, to lower the barrier for acquiring users. In general, persistence is realized with Hibernate[17], however, using relational views LensKit[12] accesses the underlying database directly.

The prototype shows the feasibility of *combining* Web augmentation with recommender systems. It was not yet optimized for performance (*prediction accuracy* and *response times*), yet it shows that the approach is suitable for empowering scripters to build new kinds of augmentations.

Since we base on established item-item collaborative filtering algorithms, the evaluation of prediction accuracy was not a goal for this paper. However, we do plan to do this in the future, along with an evaluation of our methods for computing the cumulative score from different event types. In this context, an interesting question is how well does the chosen LensKit configuration generalize for all kinds of items, and how can it be made configurable by the scripter.

Concerning response times, on one hand, the approach is limited by the connection to the server through Internet, however, with optimizations possible, for instance by providing API methods to combine several HTTP requests into one (e. g., storage of multiple events as a batch, or combining the storage of an event with a query for recommendations), or by reducing the amount of redundantly transferred event meta information. On the other hand, the computation of recommendations on the server is currently made on demand for each request, but could be made configurable, such that the scripter may define whether recommendations should be computed in advance, or if they have to reincorporate all the latest added events.

As with collaborative filtering recommenders in general, the known cold start problem is an issue to be considered, also for script writers. To mitigate this problem, in situations where ratings are publicly available, the scripter may add them as a batch, e. g., by extracting user ratings from `cocktailscout.de`, or using the XML API of `boardgamegeek.com`. Additionally, to support early development of new scripts, our prototype can be configured to append random items to the list of recommendations, until the recommender algorithm can compute enough actual ones.

Finally, so far our approach does not foresee the removal of items. In an open environment, where every end user can add items, maybe role-based user management, a reputation based approach, or a voting mechanism can be used to decide which items to remove. In this context, it also may happen that websites change their URL structure. While this is something that every Web augmentation approach has to deal with, this may cause additional database maintenance efforts if the IDs of items are affected.

[14]`http://paa.cis.jku.at/`
[15]`http://jersey.java.net/` – JAX-RS Reference Impl.
[16]`http://projects.spring.io/spring-security/`
[17]`http://hibernate.org/` – Object Relational Mapping

7. CONCLUSIONS & FUTURE WORK

We have shown the *feasibility* for providing collaborative filtering recommender system functionality as a *service* for Web augmentation. By providing a simple API and corresponding object-oriented JavaScript library, *user script writers* may employ ratings, predictions, as well as recommendations of items, to develop *new* kinds of recommender *applications* for arbitrary websites. Thus, end users benefit from personalization of the Web browsing experience through annotation, *re-ordering*, *hiding*, and *generation* of links. Several real world world application scenarios were discussed, and we presented our publicly available server prototype.

To improve and further evaluate the proposed approach, we foresee several lines of potential *future work*. Therefore, we are currently working on an extended prototype, mainly to add some minor features, improve response times, and add further means of configuration for the scripter. In a profound evaluation we aim to cover the issues outlined in the previous section: recommendation relevancy, the server-side computation of scores, response times and their effect on users' browsing experience, and methods to cope with the removal of items or changing URLs. In this kind of systems another very well known issue is the cold start problem, in this sense, we plan to provide scripters with some mechanisms to define how the script adapts the Web page when the available ratings are not enough. A possibility we are studying is to allow a scripter to use a single user profile in several scripts. In this context, we also plan to conduct several user studies, first, with scripters to evaluate the comprehensibility of the API for implementing pre-defined tasks, and how the provided service is being accepted. Second, we intend to evaluate performance, quality of recommendations, as well as scalability, in terms of a larger case study with users, employing experts or a number of end users, e. g., from the `boardgamegeek.com` community. There, publicly available ratings can be exploited to reduce the cold start problem, or to build a dataset for an offline evaluation of the employed recommender algorithm. Finally, with a complete prototype and documentation, downloads of our scripts and usage of the API can be evaluated in long term study.

In terms of functionality, we foresee several *expansions*. Support for different data formats for the API (e. g., JSON, XML, RDF) allows the reduction of technical heterogeneity, and enables simpler integration with arbitrary websites – also to give the scripters more options. Moreover, using RDF facilitates the use of semantic Web technologies and resources, not least to supply further structured information about items from resources such as DBpedia. Together with user profile data and item features collected in the browser, or extracted from online social networks, this information may be exploited by the recommender algorithms, for instance to better determine user-user or item-item similarities, or for improving recommendation results with content-based and hybrid recommender mechanisms on server side. In this context it is particularly interesting how well the current implementation recommends newly added items or items with few ratings, and if the algorithms can be made configurable for end users or scripters, to include items from the *long tail* for increasing serendipity in recommendations. In this context, the recommender system algorithms may also be configured via the API in a more elaborate way, for instance using LensKit's Groovy-based DSL[18]. Furthermore, feedback on recommendations may be exploited to improve recommendations (i. e., when the user follows a recommended link, or rates the corresponding item afterwards), and item recommendations that are constantly ignored may be excluded in the future. The set of items considered by the recommender may also be defined by the scripter, for instance, by providing a specific set or filter criteria via the API, as with a Recommendation Query Language (cf. [1]).

To *decentralize* the approach, giving even more control to the user script writer, we plan to investigate how the server functionality can be hosted by a *generic* "platform as a service" *provider* of cloud computing *services*, and if the API functionality can be provided as a downloadable and configurable *bundle*. Going beyond the server side approach, the question remains whether the architecture can be re-implemented to be independent of a single third party recommendation service provider, with recommendations being computed on client-side, and exchanging anonymized data via a dedicated data exchange server only, or relying on a peer-to-peer architecture.

Finally, we foresee the implementation of a browser plug-in to offer a graphical user interface for configuration and augmentation of recommendations, to be used by scripters or even end-users, based on a DSL to *define extraction and placement* of item and user information in the DOM (cf. [11]).

[18]http://lenskit.org/documentation/basics/configuration/

8. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowledge and Data Eng.*, 17(6), 2005.

[2] A. Ankolekar and D. Vrandecic. Kalpana - enabling client-side web personalization. In *Proc. of HT'08 - Hypertext 2008*. ACM, 2008.

[3] R. Barrett and P. P. Maglio. Intermediaries: New places for producing and manipulating web content. *Computer Networks*, 30(1-7):509–518, 1998.

[4] P. Brusilovsky. Adaptive navigation support. In *The Adaptive Web*, volume 4321 of *LNCS*. Springer, 2007.

[5] F. Carmagnola, F. Cena, and C. Gena. User model interoperability: a survey. *User Modeling and User-Adapted Interaction*, 21(3), 2011.

[6] L. Chen et al. Human decision making and recommender systems. *ACM Trans. Interact. Intell. Syst.*, 3(3), 2013.

[7] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1), 2004.

[8] O. Díaz and C. Arellano. The augmented web: Rationales, opportunities, and challenges on browser-side transcoding. *ACM Transactions on the Web*, 9(2), 2015.

[9] O. Diaz et al. Towards the personal web: Empowering people to customize web content. In *Web Information Systems Eng.*, volume 8786 of *LNCS*. Springer, 2014.

[10] D. Eynard. Using semantics and user participation to customize personalization. Technical report, HP Labs, 2008.

[11] D. Firmenich, S. Firmenich, J. Rivero, and L. Antonelli. A platform for web augmentation requirements specification. In *Web Engineering*, volume 8541 of *LNCS*. Springer, 2014.

[12] X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In *Proc. of 5th Int. Conf. on Intelligent User Interfaces*, IUI '00, New York, NY, USA, 2000. ACM.

[13] A. Garrido, S. Firmenich, G. Rossi, J. Grigera, N. Medina-Medina, and I. Harari. Personalized web accessibility using client-side refactoring. *Internet Computing, IEEE*, 17(4), 2013.

[14] S. Gauch, M. Speretta, A. Chandramouli, and A. Micarelli. User profiles for personalized information access. In *The Adaptive Web*, volume 4321 of *LNCS*. Springer, 2007.

[15] H. Guo, J. Chen, W. Wu, and W. Wang. Personalization as a service: The architecture and a case study. In *Proc. of 1st Int. Workshop on Cloud Data Management*, CloudDB '09, New York, NY, USA, 2009. ACM.

[16] Hendry, H. Pramadharma, and R.-C. Chen. Building browser extension to develop website personalization based on adaptive hypermedia system. In *Current Approaches in Applied Artificial Intelligence*, volume 9101 of *LNCS*. Springer, 2015.

[17] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, 2010.

[18] G. Jawaheer, P. Weller, and P. Kostkova. Modeling user preferences in recommender systems: A classification framework for explicit and implicit user feedback. *ACM Trans. Interact. Intell. Syst.*, 4(2), 2014.

[19] G. Kappel et al. TheHiddenYou - A Social Nexus for Privacy-Assured Personalisation Brokerage. In *12th Int. Conf. on Enterprise Information Systems (ICEIS)*, 2010.

[20] A. Kobsa, B. P. Knijnenburg, and B. Livshits. Let's do it at my place instead? attitudinal and behavioral study of privacy in client-side personalization. In *SIGCHI Conference on Human Factors in Computing Systems (CHI 2014)*, Toronto, Canada, 2014.

[21] L. Moreau et al. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6), 2011.

[22] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proc. of 10th Int. Conf. on World Wide Web*, New York, NY, USA, 2001. ACM.

[23] J. Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative filtering recommender systems. In *The Adaptive Web*, volume 4321 of *LNCS*. Springer, 2007.

[24] E. I. Sparling and S. Sen. Rating: How difficult is it? In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, New York, NY, USA, 2011. ACM.

[25] M. Wischenbart et al. Automatic data transformation: Breaching the walled gardens of social network platforms. In *Proc. of APCCM - vol. 143*, Adelaide, Australia, 2013. Australian Computer Society.