Multimedia Tools and Applications (2021) 80:6785–6809
https://doi.org/10.1007/s11042-020-09803-8

# Engaging end-user driven recommender systems: personalization through web augmentation

**Martin Wischenbart**[1] · **Sergio Firmenich**[2,3] · **Gustavo Rossi**[2,3] · **Gabriela Bosetti**[2] · **Elisabeth Kapsammer**[4]

Received: 30 November 2019 / Revised: 21 August 2020 / Accepted: 2 September 2020 /
Published online: 22 October 2020


**Abstract**
In the past decades recommender systems have become a powerful tool to improve personalization on the Web. Yet, many popular *websites lack* such functionality, its implementation usually requires certain *technical skills*, and, above all, its introduction is *beyond* the scope and control of *end-users*. To alleviate these problems, this paper presents a novel tool to empower end-users *without programming skills*, *without* any involvement of *website providers*, to embed personalized recommendations of items into *arbitrary websites* on client-side. For this we have developed a *generic meta-model* to capture recommender system *configuration parameters* in general as well as in a web augmentation context. Thereupon, we have implemented a wizard in the form of an easy-to-use browser plug-in, allowing the generation of so-called user scripts, which are executed in the browser to engage collaborative filtering functionality from a provided external REST service. We discuss functionality and limitations of the approach, and in a study with end-users we assess the usability and show its suitability for combining recommender systems with web augmentation techniques, aiming to empower end-users to implement controllable recommender applications for a more personalized browsing experience.

**Keywords** Web augmentation · Visual programming · Client-side personalization · End-user programming · End-user development · Controllability of recommender systems · Browser-side trans-coding

## 1 Introduction

Nowadays recommender systems are a popular means for personalizing user experience and services on the Web. They have a long-standing history in various domains and they are employed by a multitude of websites in areas such as e-commerce, movie databases,

✉ Martin Wischenbart
k0255857@students.jku.at

Extended author information available on the last page of the article.

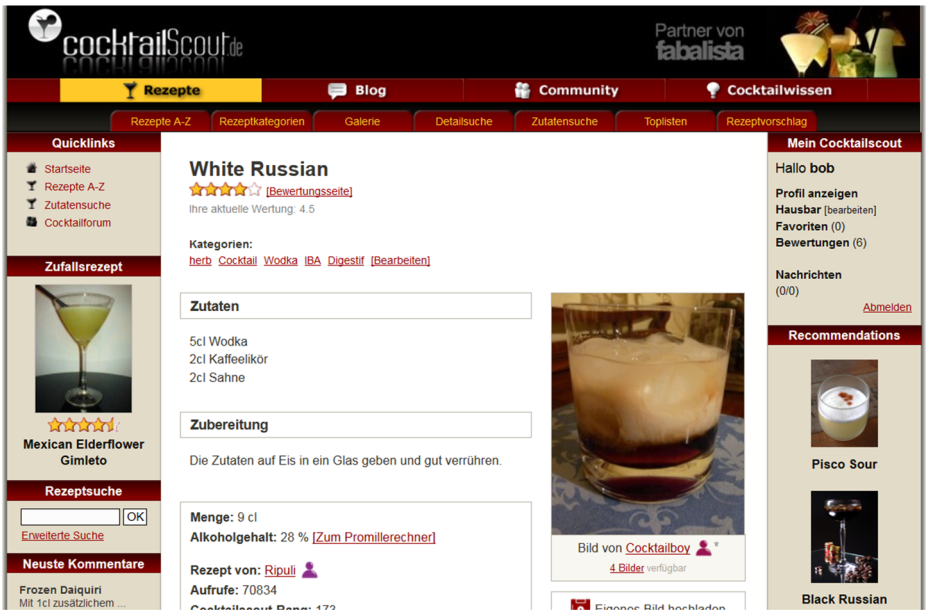


**Fig. 1** Adapted cocktailscout.de website with the random recipe shown by default (left menu bar, heading "Zufallsrezept") and the additionally augmented personalized recipe recommendations (right menu bar, heading "Recommendations"). The augmented content blends in with the site's style and the remainder of the right menu bar's content is just shifted further below [38]

food recipes, or music streaming. In spite of their potential, however, beyond well-known or commercial websites oftentimes recommendation services are not implemented by site providers, perhaps for missing financial incentives. As an example, cocktailscout.de, once one of the largest German language websites for cocktail recipes,[1] although it has a rating mechanism that is used for ranking of recipes, the site used to only provide a random recipe link on each page instead of giving personalized drink recommendations. Sharing of recipes was only possible explicitly, e.g., via email or different social platforms. Yet, containing more than 1500 recipe items the site would be a perfect target for implementing a collaborative filtering recommender system directly within the site. Based on the tastes of similar users, individually personalized drink recommendations could be included in one of the sidebars, as shown in Fig. 1. Traditionally adding such functionality would require modifications to be made on the site's server and, therefore, its implementation fully hinges on the *website provider*.

In order to reduce such dependencies on site providers, in recent years *web augmentation* (WA) techniques, i.e., the addition of external content or behavior to web pages on client-side, have become a popular means for *end-users* to adapt pages according to their own requirements. Using these techniques, we have demonstrated the feasibility for realizing *Personalized Recommendations in Web Augmentation Applications* ("PAA"), by

[1] Highest global Alexa rank in a comparison of 13 German language cocktail websites (http://www.alexa.com/, made in 2015). After a recent redesign functionality was reduced and the site's popularity dropped.

implementing an external service for collecting user's item ratings and offering collaborative filtering functionality via a REST ful API, as well as corresponding manually-written web augmentation *user scripts*,[2] executed in the browser. This work, including the CocktailScout example shown in Fig. 1, was presented at a past workshop [38]. Nevertheless, the manual implementation of such user scripts without a doubt requires a certain level of programming skills and basic knowledge about recommender systems and, therefore, they can only be programmed by more advanced users and most certainly not by typical end-users.

To tackle this problem, based on our previously proposed architecture [38], as well as studying existing systems and literature, we abstracted the process for building recommender systems to create a generic meta-model for their configuration and control. This model was thereon applied to the provision of recommendations with a collaborative filtering service by means of web augmentation, to determine the configuration parameters required for the manipulation of web pages. In this paper we show how these parameters can be acquired through a graphical wizard by end-users within the browser, consequently enabling recommender system user scripts to be generated semi-automatically. In this manner, this "RecSys-Creator" browser plug-in enables end-users to develop collaborative filtering applications for almost *arbitrary websites*, without the requirement of programming skills, executed on client-side, and with reduced dependencies on site providers. We conducted a user study centered on its usability, involving users with different technical backgrounds and varying levels of programming skills, and we hereby present and discuss the results.

The remainder of this paper is structured as follows: In the upcoming Section 2 we discuss related work. Next, Section 3 shortly summarizes our previously proposed architecture for recommender system user scripts. After that, Section 4 presents a generic meta-model for configuration of recommender systems, and, after that, an end-user development (EUD) approach to create recommender user scripts without programming skills or knowledge about the operation of recommender systems. Finally, an end-user evaluation study and its results are discussed in Section 5 along with shortcomings and limitations in comparison to other approaches, before Section 6 presents our conclusions and outlines potential future work.

All our source code including documentation, details on the evaluation survey as well as further material is available online at http://paa.cis.jku.at/.

## 2 Related work

This section discusses related research from several fields, including recommender systems in general, client-side personalization, web augmentation, and recommender system controllability. Since the beginnings of the Web, the web personalization research community has been expanding steadily, and in order to satisfy the increasing number of end-users, various approaches for user profiling, profile data integration and personalizing content or services have emerged.

Regarding the latter, a comprehensive survey [6] on recommender systems presents a classification of such systems and discusses different kinds of cold start problems (see also [35]) among other topics. Regarding the recommendation of content and people in

[2] "User scripts" or simply "scripts" are typically written by advanced users with knowledge of JavaScript, so-called "scripters". They are executed within the browser on client-side to modify web pages—using dedicated extensions such as Greasemonkey (http://www.greasespot.net).

social media, so-called social recommender systems were studied [33]. In this context it was also shown that through consideration of relationships from social networks recommendation accuracy can be improved [40]. Different ways for ratings have been investigated [36] and classifications of user feedback have been surveyed [25], including their correlation to ratings. While most of these recommenders work on server-side, recommender engines can also be implemented in JavaScript [32], running on client-side and relying only on a model to be rebuilt periodically by the server.

To adapt existing third-party web content externally, intermediaries [5] may intercept and modify content on a proxy server. Going further, several approaches for client-side personalization have been developed (e. g., [3, 23]). In such scenarios modifications and recommendations may cover different sites, since different applications can share a single user profile, for instance, managed on client-side using an appropriate browser extension. Entirely client-based, browser extensions which monitor user navigation can be used to populate user profiles (with navigation history, bookmarks, keywords, etc.) and thereupon recommend relevant web pages to users [16, 18]. Concerning comprehensive personal user profiles, an important issue to be considered is privacy, as it was also studied in literature [28]. In this context, it was proposed to enhance privacy and increase trust by storing user profiles on client-side and by providing users with explanations for recommendations [29]. A comprehensive overview of client-side approaches (peer-to-peer or with aggregation server), also focusing on anonymity and privacy, can be found in a PhD thesis [4]. In that spirit, using a middleware to exchange information between end-users allows to build privacy-aware recommender systems based on interest groups [15]. Beyond that, employing a federated learning architecture, it was also proposed to compute personalized recommendations on clients based on subsets of a global database, which are propagated to others using a publish/subscribe mechanism [31], however, without a deeper discussion of user interface and presentation issues.

To perform web personalization as a service [21], nowadays there is a multitude of companies offering rating and recommendation as web services (e.g., through widgets for rating[3] or more advanced systems for personalizing user experience and advertising),[4] however, requiring changes in the original website.

Less dependent on site providers, another means to achieve personalization is the application of web augmentation techniques, which allow users to customize website user interfaces (UIs) in terms of content and functionality, according to their own requirements [17]. Most web augmentation projects are developed as browser extensions, and once installed by the user, they modify loaded web pages, thus altering what the user perceives. In order to include more end-users into the development task and to satisfy their specific and personal requirements, end-user programming (EUP) tools have emerged that allow them to use web pages as an editable canvas. EUP was successfully applied in the context of web mash-ups: with NaturalMash [2] end-users without programming skills can create their own applications, and WebMakeUp [13] enables such end-users to rearrange HTML contents. In these tools the artifacts are specified in terms of an underlying meta-model, and using a specialized engine these specifications can be interpreted and executed when target web pages are loaded.

[3]Rating Widget, Testimonial Robot: rating-widget.com, testimonialrobot.com

[4]Yusp, Plista, Amazon Personalize, Strands: yusp.com, plista.com, aws.amazon.com/personalize, retail.strands.com

An alternative approach is to generate code based on a given set of input parameters [17]. In this manner, it is also possible to generate JavaScript code in the form of *user scripts*,[2] the most common artifact in the web augmentation communities. In these communities user scripts are often publicly shared, and a number of repositories provide a variety of scripts[5] for all kinds of web pages and modification tasks. Examples range from layout modification and tweaks on youtube.com[6] (e.g., regarding video player size, video & audio customizations, etc.), managing comments on geocaching.com,[7] to improving navigation on dropbox.com by rendering a Tree View panel.[8]

Despite the variety of available user scripts, however, according to our own experience and as pointed out by a survey [12], 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. In particular, with a single web augmentation artifact (i. e., user script) individual end-users usually experience the same effects. Recent research in this area mainly aims to provide tools (frameworks or languages) to achieve 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 websites. For instance, CSWR [19] aims to improve web accessibility (e. g., for people with disabilities), and the aforementioned WebMakeUp [13] allows end-users to specify custom website modifications and augmentations via a graphical interface (working with XPath expressions [34] in the background). Thus, whereas all the web augmentation approaches propose a way to customize the Web, most of them work without individual underlying user profiles. However, recommender systems and in particular collaborative filtering base on modeling both, user profiles and items. In this way, to provide an EUP approach for PAA, it is required to extract or generate this information from web content. For such functionality it was proposed to utilize semantic annotations [26], and in another approach the extraction is carried out through a visual process [11]. In the context of EUP for our own approach, semantic annotation is more suitable, since it enables users to define recommendable items through selecting specific parts from web pages as input for the recommender engine.

Finally, controllability has been studied in the context of applications that were designed with a recommendation service in mind. Knijnenburg et al. [27] show that giving the users, who have significant domain knowledge, explicit control over the weights of items and friends in collaborative filtering increases quality of recommendations as well as user satisfaction. Hijikata et al. [24] observed that some factors, such as the time and effort required by the user for his intervention, play an essential role in the user satisfaction. Harper et al. [22] noted a preference from users for recommendations received after they had a certain level of control. Ekstrand et al. [14] observed that a substantial fraction of users chooses to switch among recommendation algorithms until they find the one which satisfies them the most.

In this regard, a fundamental problem is that when site providers implement a recommender system, they traditionally display a predefined and limited range of items only, i.e., from within a single domain (e. g., only movies, only books) or from a single source (items

[5]For instance, GreasyFork (http://greasyfork.org) has more than six thousand scripts, some of which are installed more than fifty thousand times. OpenUserJS (http://openuserjs.org): more that 8.000 scripts; UserScripts-Mirror (http://userscripts-mirror.org): more than 100.000 scripts.

[6]http://greasyfork.org/en/scripts/943-youtube-center

[7]http://userscripts-mirror.org/scripts/show/75959

[8]http://greasyfork.org/en/scripts/4955-dropbox-plus

from a certain website). However, cross-domain recommendations are possible and can be beneficial for the user [9], and there are situations in which resources of interest are spread over multiple sites on the Web.

Motivated by this need for controllability and customization, for our approach we have chosen to semi-automatically generate user scripts for augmentation of collaborative filtering functionality, as we will elaborate in the upcoming sections.

## 3 Recommender systems through web augmentation

In this section we briefly discuss our previously proposed PAA approach, with the required steps for user script creation, covering different ways to modify web pages with a collaborative filtering service using manually written scripts.

### 3.1 PAA architecture overview

In our previous work [38], we presented a PAA *client-side library* to simplify the development of collaborative filtering user scripts. Working on top of well-known augmentation engines (such as Greasemonkey, Tampermonkey, Violentmonkey), this library aids the collection of required user ratings, i.e., weighted relations between users and items. The client-side components communicate with the storage system and the recommender engine on a dedicated PAA *server* through a RESTful API.

Individual clients retrieve item rating predictions (which may be exploited for link ordering, link hiding, or link annotation), or item recommendations (which can be used for link generation, as classified by [8]). Utilizing the cocktail recipe example from the introduction, the overall architecture of the approach is depicted in Fig. 2, including the three main steps, which are outlined in more detail in the following.

Going beyond the scope of this paper, relying on a central server the individual clients do not have access to other users' preferences, but to avoid potential attacks, appropriate authentication mechanisms and security protocols must be in place.

**Data collection and sending to server** ① For modeling the user's preferences, we rely on *events* relating *users* with *items* (both identified with unique IDs) and including the numeric score for *ratings*. Therefore, scripts may extract scores from an existing rating mechanism, introduce such a mechanism for collection of explicit ratings, or implicitly compute a score. For presentation purposes (i.e., link generation; see ③), we also require a human readable item *name*, to be inserted as link text later on. Additional information about an item may be added as *meta info*, such as an image URL. In the cocktail example we use the drink's URL (item ID), the user login name (user ID), the numeric rating extracted from the page, as well as the drink's name, and the optional picture (image URL). Afterwards the collected data is being *sent* to the server.

**Processing on server** ② On the server, events are stored and if applicable a numeric rating value is computed (e.g., when explicit ratings or implicitly collected events should be accumulated to compute a score on server side).

**Retrieval from server and augmentation in page** ③ After that, different data can be retrieved from the server: firstly, previously stored ratings including additional information, such as average ratings and their distributions; secondly, predictions for ratings of

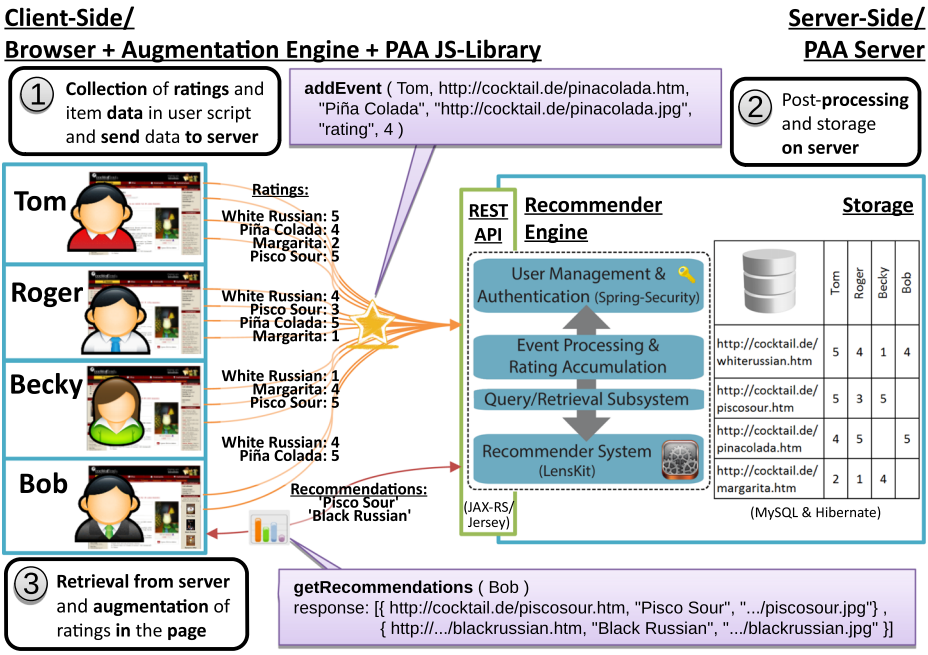


**Fig. 2** Architecture of our previously proposed PAA approach with client- and server-side including an overview of steps from collection of ratings (1) to augmentation of modifications in the page (3)

(previously unrated) items; and finally, recommendations for items (with predictions and recommendations being *computed on-the-fly* using the recommender system library on the server). Such queries may be triggered from the same or a different script automatically, or on demand (i. e., manual 'pull recommendations').

Finally, the retrieved information can be augmented in the page in different manners: ratings and their distributions may be added (*link annotation*, e.g., as pop-ups for all links referring to drinks). Rating predictions may be employed for *re-ordering* item links or for *hiding* them (e.g., if predicted score is below a threshold). Finally, recommendations can be used to *generate* personalized links on the page, referring to items the user might be interested in, such as in a list of drink recipe recommendations.

## 4 An approach based on end-user development

Taking our previously proposed architecture to a more generic level, in this section we propose a model for configuration and control of recommender systems in general as well as for web augmentation. Furthermore, to reach a wider audience, we present an EUD approach to enable end-users without programming skills to create recommender user scripts.

**A meta-model for configuration and control of recommender systems** To capture the parameters of recommender systems on a generic level, taking into consideration our previously proposed architecture, and studying existing systems as well as literature, we have developed a conceptual meta-model for configuration and control of recommender systems, as presented in Fig. 3. The root class at the bottom comprises six main areas:

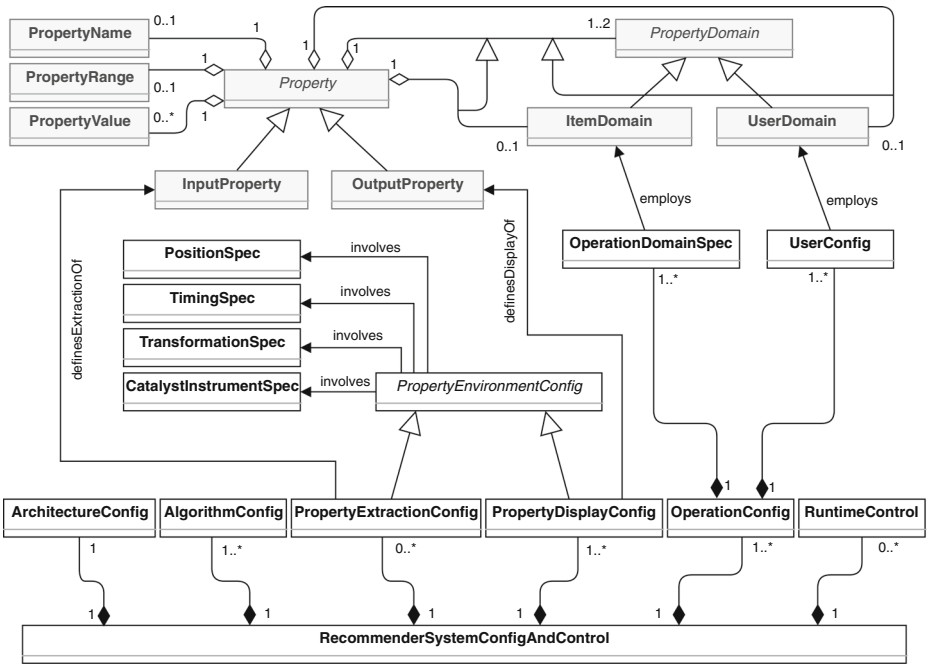


**Fig. 3** Proposed meta-model in UML for configuration and control of recommender systems in general as well as for web augmentation. All the model classes in the context of Property can be associated directly with web page DOM elements (grey fill, top)

1. ArchitectureConfig: defines architecture as graph, i. e., clients, servers, intermediaries as nodes, including their responsibilities, as well as communication directives as edges;
2. AlgorithmConfig: defines the type of recommender engine (collaborative filtering, content-based, etc.) and algorithms (scoring, prediction, fallback mechanisms, similarity computation, feature extraction and combination, etc.);
3. PropertyExtractionConfig configures the extraction of InputPropertys for the recommender engine;
4. PropertyDisplayConfig: configures the display of OutputPropertys from the recommender engine;
5. OperationConfig: defines the domains for operation of the system, i. e., the class of items to be recommended, and the class of users to recommended to (optional: distinction between source and target domain for items, assigning levels of rights for users/groups);
6. RuntimeControl: Control and reporting of the system at runtime;

Extraction and display of content rely on Propertys (with name, range, value) describing items and/or users (i.e., the domain of a property). Further it must be specified where (PositionSpec), when (TimingSpec), and how (TransformationSpec) properties are extracted/displayed, as well as whether some additional instrument (CatalystInstrumentSpec) is employed to facilitate extraction/display.

Basing on this meta-model, we propose an EUD approach to enable users to instantiate the comprised meta-classes for arbitrary websites. Thereby, in principle, all parameters can be configured by end-users. However, instantiating this meta-model for recommender systems based on web augmentation, and employing an architecture such as the one presented

in the previous section, we refrain from allowing users to customize ArchitectureConfig and AlgorithmConfig. Instead we focus on configuring the user interface and interactions at runtime. We present a study about an interaction mode, in an end-user development setup, to let end-users define the configuration aspects without requiring any programming skills. This, as we will show, involves interactions to define extraction templates for items, visual DOM manipulation to arrange layout and some configuration through user-friendly forms. The development process is supported by examples, giving end-users a final view of their artifacts while they are being defined.

Web augmentation with recommendation features entails the modification of target pages' DOMs, and the implementations of several classes from the meta-model are tied to DOM elements (see Fig. 3), because they represent perceivable aspects of the recommender system (from the end-user's perspective), and, therefore, they must be managed at the augmentation layer.

Focusing on link generation based on numeric ratings given by the users, we have instantiated that meta-model for recommender systems based on web augmentation, basically by capturing those elements related to user interaction. Most commonly, what a user perceives of recommender applications is items, their ratings (maybe a widget to rate), and, finally, recommendations (also possible through a widget). Thus, the required parameters are, firstly, the *positions* of information to be extracted *from an item's web page* DOM: user, rated item, rating value (PropertyExtractionConfig). Secondly, it must be defined how to present recommendations to the user. For embedding links in the page, we need a *position in the* DOM to render and place items (PropertyDisplayConfig). Thirdly, it must be specified on *which sites* and for *which users* to run the script in the first place (OperationConfig). Finally, the creators of augmentation artifacts may customize control options for the final script users (RuntimeControl).

**Acquiring recommender specifications from users** In the following we discuss how these parameters can be acquired from end-users. Thereby, the various positions of elements for gathering inputs and positioning output widgets are selected and specified using visual selection of DOM element positions. In parallel we present our corresponding RecSys-Creator wizard implementation, which is deployed as a browser side-bar plug-in. In our implementation we base on XPath expressions for positions, which we obtain using example item pages (assuming that web pages representing items adhere to a static structure). In the example we create a user script for recommending articles on Wikipedia (applicable directly to any wiki site using MediaWiki as front end). In general we tried to support alternative kinds of uses, depending on the available features of the target web site.

**1. PropertyExtractionConfig—user properties** For user identification, a unique name or ID is required, such as an email address (globally unique) or a site login name (site- or script-specific). Web sites with login functionality, including Wikipedia, typically show a user ID on the page, which can be extracted from the matching DOM element. After pressing the corresponding button in the side-bar of our wizard, the login name can be selected with the mouse pointer. Thereupon, its XPath expression is saved as one of the parameters in the wizard and the HTML element is being highlighted (as shown in Fig. 4).

Alternatively, authentication may be performed directly against the PAA server through a JavaScript widget, when sites do not have a login mechanism, when unique user IDs are not rendered within pages, to relieve script end-users from having to register at the target

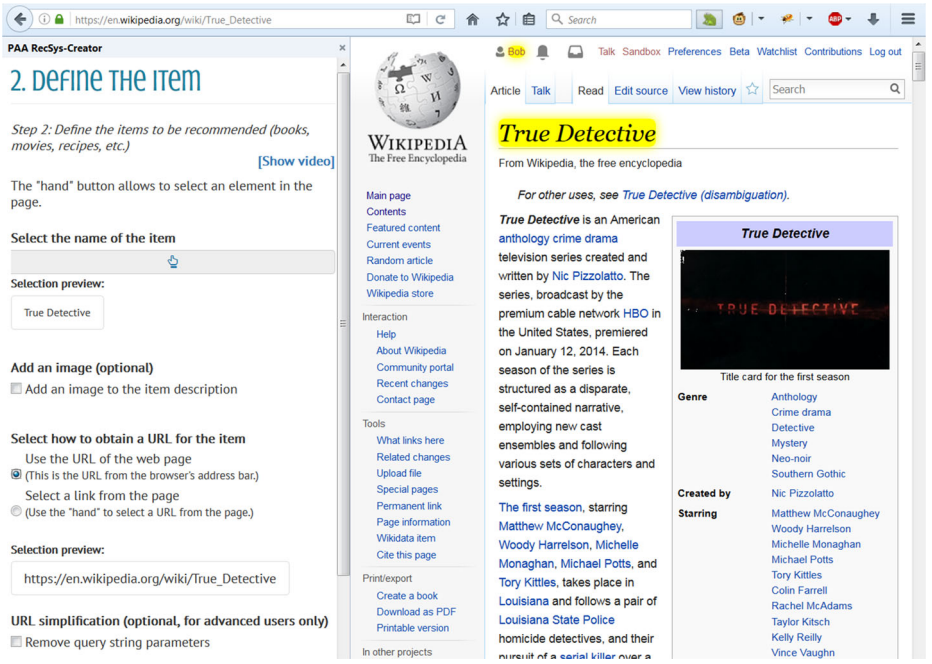


**Fig. 4** RecSys-Creator plug-in wizard after definition of user ID (highlighted user name "Bob" at the very top of the page), item name (page title "True Detective"), and URL (from the browser's address bar) for a Wikipedia article recommender user script

website, or to increase security. In this manner, third party identity providers may be used, such as OpenID[9] (supported by Google accounts) or Facebook Login.[10]

**2. PropertyExtractionConfig—Item Properties** Next, for the items (articles) to be recommended, again unique IDs as well as some additional properties for item display are required. Under the assumption that in the context of the Web every item to be considered by the recommender system can be identified with a URL, we propose to represent every item with its page (URL, e. g., taken from browser's address bar or extracted from page DOM). For this our wizard allows to remove URL query string parameters which may be shown in the browser but are not actually part of the ID (e.g., `?source=ttep_ep1`, specifying from where the link was clicked), either completely or keeping selected ones in case they are actually required.

To allow reasonable augmentation of recommendation links on the site, the approach further requires a human readable name for each item, which could be extracted from the page title, or from within the page DOM (e. g., a heading). Optionally a URL for an item image can be extracted in the same manner.

**3. PropertyExtractionConfig—ratings from explicit and implicit data** When sites have a rating mechanism and user ratings are shown within the item page, explicit ratings can be

[9] OpenID: http://openid.net/
[10] 'Facebook Login' (or 'Facebook Connect'): https://facebook.com/help/?page=730

extracted directly (and the user script can send them to the server immediately after page load).

In the case of Wikipedia no rating mechanism to measure the user's interest is present. A straightforward solution is to implement one, for instance, in form of a standardized ratings-widget. Our wizard allows to augment such a widget into pages, positioned relative to (i.e., before or after) another element in the page DOM. The position can again be specified via graphical selection of an element to obtain an XPath expression, as shown in Fig. 5. The widget then collects ratings (numerical, from $\frac{1}{2}$ to 5 stars) and sends them to the PAA server whenever a rating is given by a user. Alternative rating scales could be, e.g., unary (like), binary (like vs. dislike), or a 100-point slider (see [36]). Finally, instead of asking the user for explicit ratings, the user's interest can also be estimated from implicitly recorded inputs or signals (e.g., page visit count and duration monitored in the browser, or number of discussion posts and reported plays from a gaming community site) including context (e.g., current location, time, brightness). Such additional information could further be employed for filtering or re-ordering of recommended items. For textual comments or posts, analyzing whether they are positive or negative through natural language processing could also yield further insights. Other examples would be querying the user's current geographic location or local time for recommender systems with location or temporal awareness. Using augmentation engines and extensions in the browser, monitoring user behaviour a wide range of parameters can be collected, and, if necessary, on the server they can be aggregated to derive rating values (i. e., the server has a memory of past parameter recordings, also avoiding any storage requirements on clients). However, whereas it would be possible to build comprehensive user profiles from such recorded inputs, a discussion of which parameters to use or, thereupon, the configuration of a recommender engine would go beyond the scope of this paper.

**4. PropertyDisplayConfig—recommendations** In principle recommended items could be provided to the user in any kind of separate UI element, however, an obvious solution is to augment item links directly into all shown item pages (also allowing customization

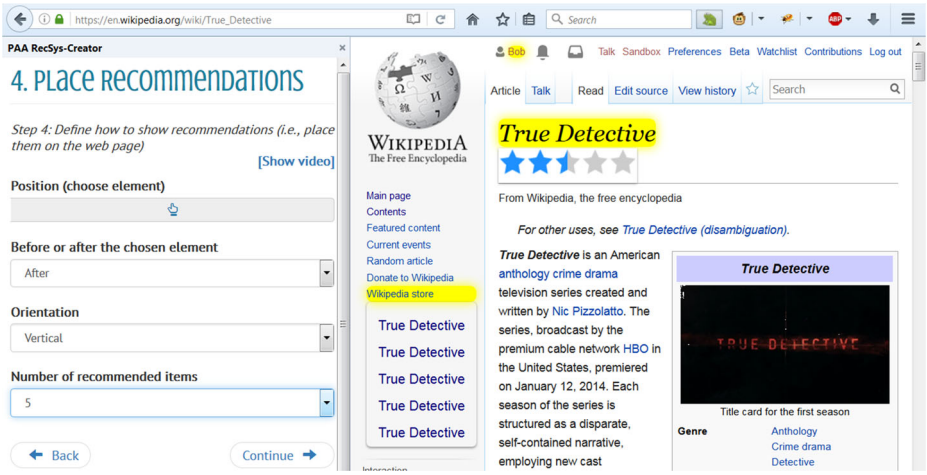


**Fig. 5** RecSys-Creator plug-in wizard after placing the widgets for rating (center/top; stars below page title "True Detective") and for recommendations (page's left menu; with orientation "vertical" and number of recommended items "5")

of number and layout of shown recommendations). In our wizard, like with the ratings-widget above, the user has to configure the relative position and parameters for this recommendations-widget, which retrieves recommendations from the PAA server (in JSON format) and shows them according to its configuration.

**5. OperationConfig—script execution parameters** Beyond extraction and display of properties, a URL *pattern* defines on which pages the recommender script should be executed (i.e., the item domain). Additionally, script *name* and *namespace* (e.g., the creator's email address) define a unique operation domain for the recommender engine. This means that, multiple user scripts can share items and ratings between each other, enabling the versioning of scripts, and, additionally, by specifying the same operation domain for scripts running on different web sites, simple cross-domain recommender systems can be realized in a straightforward manner.

**6. RuntimeControl—system control at runtime** As discussed in the related works section, controllability is an important aspect to achieve user satisfaction. Therefore, it is fundamental to inform the end-user about the system's actions, as it is currently possible through monitoring customizable information messages in the browser console. To increase user control, implicit or explicit feedback on recommendations could be collected through the recommendations-widget, i.e., implicitly tracking item clicks, or explicitly providing options for "don't show this again" or "more like this". To take this further, widgets or dedicated UIs (e.g., on the PAA server) could also provide options to select and customize algorithms and architectures. For this, however, the script creator would have to specify alternate options in all the corresponding configuration classes.

**Wizard implementation details** From a technical point of view, our RecSys-Creator plug-in implementation is a browser extension developed as a bootstrapped Firefox extension (executable on Firefox up to version 56). That is to say, we provide a special script with four functions (install, uninstall, start up, shut down) that the browser can call, and these are in charge of loading/unloading special behaviour and UI elements into the browser. We used the high-level API of the Firefox add-on SDK, which allowed us to interact with the browser tabs, sidebar and add a toolbar toggle button, to make CORS requests (cross-origin resource sharing), to use L10n content localization and to attach new behaviour and style to the visited web pages.

A web extension of this kind has components at two levels: browser level and page level. Those at the browser's side have a higher level of permissions regarding requests, file storage, and browser UI modification, but those components cannot directly manipulate the DOM of a page. For this it was required to load some components at page-side and implement data exchange between page, browser, and sidebar. There, in the sidebar, our wizard can be loaded for any page the user wants to work with. We used page workers to load JQuery (for pages that do not have it) and our own JavaScript- and page-side artifacts that make it possible for our extension to interact with the page content, for instance, to enable users to pick a DOM element and retrieve its XPath for further processing.

The sidebar (built using SDK UI components) creates a new context for any web content to be loaded independently from the current web page. This means that we did not tarnish the current page style, behaviour or content to implement our wizard. In this manner, the browser-level components constitute a bridge between page components and sidebar components, and they are responsible for retrieving external content as well as for saving the recommender user script specification. Employing a set of hard-coded JavaScript blocks

and libraries, this specification allows to generate a user script and offer it for download and installation. After the installation in an augmentation engine (e.g., Greasemonkey), the user script automatically modifies loaded pages and augments recommendations as it was shown in Fig. 1. The Appendix provides further examples of running scripts.

Not least, it is important to mention that for practical reasons, in this paper, we use our previous client-server architecture for adding recommendations through web augmentation. However, the (EUD) approach presented in this section is at a different level of abstraction, focusing on an EUD environment that generates a user script using a particular code generator strategy. In this regard the concepts and techniques are reusable even when it would be required to generate augmentation artifacts for another architecture, such as, for instance, for generating peer-to-peer augmentation artifacts [20].

## 5 Evaluation, results & discussion

The overall goal of the evaluation were, firstly, to find out whether end-users understand the general idea of the approach and if they are able to follow the wizard's instructions, also assessing the usefulness of the provided documentation with tutorial videos. Secondly, we investigate if they are successful in creating working recommender user scripts. Thirdly, we question participants to acquire their opinion on the RecSys-Creator's usability and versatility for different people, websites and situations. Concluding we discuss threats to validity and limitations of the approach along with some potential remedies.

### 5.1 Evaluation task and survey results

The participants' task was to use the plug-in wizard for a website of choice to create a recommender system user script, to run that script (giving ratings and receiving recommendations, but not necessarily sharing the script with other users), and, finally, to fill in an online survey with predefined options as well as free-text form fields.

**Participants & skills** The survey had 30 participants,[11] who all stated that they use browsers daily. More than half also use browser plug-ins at least almost daily. In contrast, only 4 participants employ user scripts on a regular basis.[12] Further, 6 users have already developed browser extensions or scripts.[13] The participants mostly have some sort of technical background and during self-assessment 20 stated to have professional or advanced programming skills,[14] however, surprisingly, only 3 have such knowledge of recommender systems.

**Goal 1: understanding and documentation** Our provided documentation (information on the website, instructions in the sidebar plug-in, tutorial videos) for using the RecSys-Creator was perceived to be mostly useful.[15] For 83% of participants using the RecSys-Creator was

[11] 12 of them were engaged in the context of a practical course within the computer science master curriculum at Johannes Kepler University Linz, Austria, whereas the remaining 18 were voluntary participants from Austria, Argentina, France, and Spain. The participants were from 19 to 40 years of age.

[12] Chose between 'some times per month' and 'daily'.

[13] User scripts shared with friends or an online community: 5 participants;
Browser plug-ins: 2 participants.

[14] Only 7 have professional or advanced skills in JavaScript.

[15] For website, sidebar, and videos: 26, 29, and 25 out of 30 participants chose 'mostly' or 'very much' useful. 6 participants would have wished for more documentation.

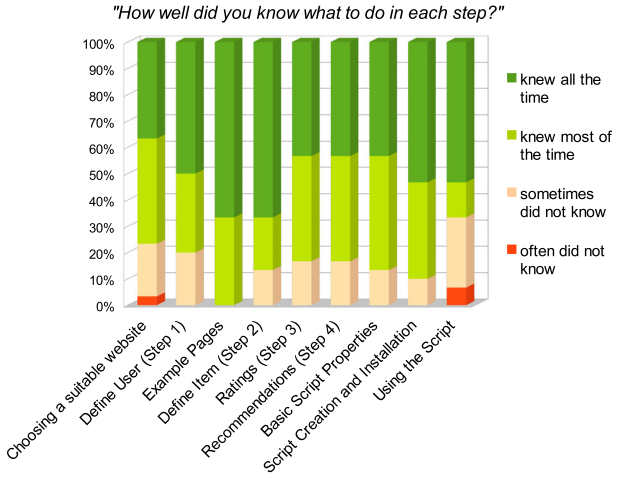


**Fig. 6** RecSys-Creator evaluation survey responses regarding experienced difficulties throughout the various steps of the given tasks

easy[16] and, as shown in the diagram in Fig. 6, for the individual practical tasks they largely knew what to do in each step. In particular, those steps that were guided by the wizard caused only a few problems.

In this context, we did also observe tendencies that skillful programmers as well as participants with better knowledge of recommender systems had fewer problems.[17]

In spite of these positive responses, difficulties arose regarding installation and handling of browser plug-ins and created user scripts, as well as during the selection of suitable websites. In this context, through the free-text response questions we learned that several participants in fact did not comprehend the concept of collaborative filtering (in particular the fact that ratings by (multiple) other users are required to retrieve meaningful recommendations). However, since dealing with plug-ins and scripts is inevitable in the world of web augmentation, better instructions and integration in the RecSys-Creator may improve user experience in these regards. As for the selection of sites and pages, users who voluntarily augment personalized recommendations should have chosen a suitable target site before even starting the RecSys-Creator. Nevertheless, additional guidance or a verification mechanism would be favorable, especially to explicate the requirements for user ratings, and script usage statistics could be reported to script creators as well as users.

**Goal 2: functionality of scripts** Each of the participants created at least one user script, with a variety of target websites and items.[18] 47% of participants managed to create a script that

[16] Chose 'easy' or 'mostly easy'.

[17] 87% of participants think that technical or programming skills are 'not at all' or 'hardly' necessary to use the RecSys-Creator.

[18] Ranging from books, food recipes, beers, movies, computer games, comics, shopping portals, wikis, blogs, news, to university courses.

works 'very much' as they had expected.[19] 10% could 'not at all' create such a script.[20] The remaining majority of scripts did in fact work fine in terms of functionality, but problems arose during the positioning of widgets, as we explain in the following.

**Goal 3: usability and versatility** As mentioned earlier, augmented widgets for rating and recommendation are positioned relative to existing elements in the page DOM tree, which in turn are picked by clicking with the cursor. Whereas several survey participants particularly liked this way of interacting with the page, it was irritating to some users that it has to be done in a trial-and-error procedure (click and retry if not satisfied). Neglecting the given instructions, several participants did not do this.[21]

To alleviate such problems, the wizard should provide better explanations and a more interactive guidance. Additionally, further options for widget positioning and style could be implemented (e.g., positioning as sub-elements or absolute positioning, or applying custom CSS styles), and the augmentation could be verified as a separate step during script creation (e.g., by forcing users to interact with the positioned widgets). Finally, when errors occur at script execution time, for instance when a site's individual pages do not adhere to a coherent structure or due to modifications made by the provider, appropriate reports should be provided to the user or script creator.[22] Finally, in this context, Fig. 7 exhibits some survey responses regarding the plug-in wizard's versatility according to the survey participants.

**Further observations** Measuring the time it took the users to create their first script, for the vast majority it took between 5 and 18 min to finish (from starting the RecSys-Creator wizard to creating the `.user.js` user script file).[23]

### 5.2 Threats to validity

The prime target audience for using the RecSys-Creator comprises end-users who are members of some community or website users who want to introduce a recommender system for personalization on their own initiative. In contrast, our sample of 30 survey participants were asked to select some site on their own. This does not perfectly resemble the actual target audience of end-users, and the survey results have to be interpreted considering this aspect.

Regarding the necessity for programming skills, only 5 of 30 participants stated to have

[19] Scale ranging from '1 (not at all)' to '4 (yes very much)'; 27% answered '3'.

[20] One of them simply misunderstood the concept of individual items and pages. Several others failed due to (now fixed) bugs in our prototypical implementations. Two scripts did not work because users positioned the rating and recommendation widgets relative to each other.

[21] Resulting in widgets being augmented such that they are difficult to be seen, e.g., overlapping with other content or positioned at the very end of the page.

[22] Instead of logging to the browser console only.

[23] For a few outliers it took below 3 or beyond 20 min. Interestingly, there also is a gap from 10 to 13 min. Thus, in fact there seem to be two groups (5–9 and 14–18 min), possibly due to some users having watched the tutorial videos beforehand or not at all, while others may have watched them from within the wizard.

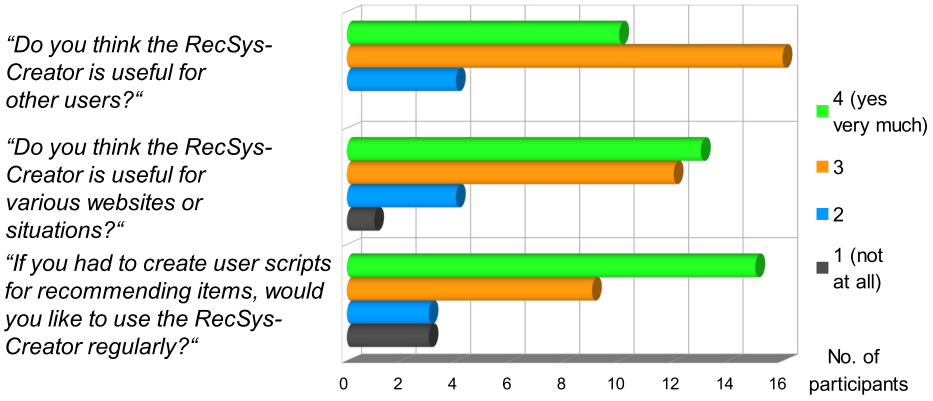


**Fig. 7** RecSys-Creator evaluation survey responses regarding versatility

basic or no programming skills. Nevertheless, 3 of these 5 managed to create a user script that functions perfectly according to our judgement (all parameters defined and widgets positioned properly). Thus, we conclude that programming skills are indeed not a requirement for using our RecSys-Creator, but there is definitely a necessity for some technical and browser skills.

### 5.3 Limitations of the approach

In addition to the issues that were identified through the evaluation, there are some limitations of our approach. In the following, also comparing our approach with others, we discuss these issues along with potential remedies.

**Item removal** So far our approach does not foresee the removal of items, and, thus, invalidated items may still be recommended by the PAA server. The RecSys-Creator documentation therefore discourages its use for items or pages with a limited lifetime, such as online auctions. Other approaches do not have this problem: oftentimes recommender systems (be it traditional or distributed ones) do not separate data collection from model creation; and systems for personalization as a service are usually well integrated with the site's database of items (instead of collecting data on-the-fly during web browsing). Potential remedies could be setting a generic time-to-live to remove items after they have not been rated for a certain time. Another possibility would be checking for page availability or specific contents of pages (e.g., "bidding has ended"), either regularly from the server or when pages are loaded on a client. Finally, items could be removed manually (e. g., through a widget button), either globally for all users (optionally using a voting mechanism), or for individual users only (LensKit supports setting such filtering restrictions; cf. Recommendation Query Language [1]).

**Uniqueness of item IDs** Currently the approach bases on web addresses as they are used for browsing the web, i.e., URLs, to identify items. However, these are per definition only

*locators* of resources, consequently they are not necessarily unique (i.e., multiple URLs may represent the same item). This is a challenge faced by many approaches which crawl sites on the Web to retrieve unique items or pages. For our generic approach aiming to work with arbitrary websites, however, this is rather difficult to automate. To alleviate this situation, besides removing query string parameters from URLs and unifying protocol prefixes (`https` vs. `http`), a straightforward solution could be to create unique IDs from URLs, item names, specific sections in the page DOM, or combinations of the former.

**Recommendation quality** Our server bases on an extensible open-source recommender system framework, providing a variety of configuration and customization options[24] to deal with the issue of quality of recommendations, as this would have clearly gone beyond the scope of our paper. During development and as long as there are not enough collected ratings, random item recommendations are given. Nevertheless, with only a few users sharing a specific recommender script, the cold start problem may be particularly grave in comparison to other solutions. As a remedy, publicly available ratings and items could be used to populate the server database beforehand (e. g., using a separate wizard). Furthermore, feedback on recommendations may be exploited to improve recommendations or to tackle the cold start problem, as previously proposed by Xie et al. [39], who further incorporate tag information for that purpose.

**Website modifications** As it is the case for any web augmentation user script, the presented approach is rather sensitive to website layout and structure modifications. Traditional recommender systems, in contrast, usually have direct access to the underlying model, and are therefore invariant to view changes. Consequently, error handling and reporting from scripts is crucial and, to facilitate script maintenance, the RecSys-Creator should provide functions for versioning and for notification of creators upon site changes, as well as for propagating changes to script end-users (e. g., using the functionality for automatic updates in Greasemonkey). Not least, if page styles are modified, and the content can still be matched, new XPath expressions may be acquired and user scripts be updated automatically.

**Communication and execution times** Regarding script execution and traffic between client and PAA server, there are several issues that we did not investigate in detail, including performance and scalability of server components. Since scripts are not aware whether items were rated already (by any user), currently all item information is being extracted and sent to the server redundantly with every rating, and, in addition, ratings are sent every time the page is loaded in the user's browser. This causes a significant overhead on the server, therefore, optimizations would be required for real applications. On one hand, in this manner page load times are not affected in the browser, but on the other hand the asynchronous DOM modifications can cause visible delays in page assembly. Unfortunately, there is no obvious solution to this problem specific to our approach, since only after page load from the web server (1. view generation), augmentation data can be requested from the PAA server (2. query recommender service) to later modify, once again, the shown page (3. view modification). To alleviate this problem, a possible optimization in terms of performance would be to pre-build recommender models and to cache models as well as item catalogs on clients.

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

## 6 Conclusions and outlook

In this paper we have presented an approach to combine web augmentation techniques with recommender systems, allowing end-users without programming skills to embed collaborative filtering functionality into almost arbitrary websites. For this we have established a list of required configuration parameters to automatically create collaborative filtering JavaScript user scripts, basing on our previously proposed PAA architecture [38]. Thereupon, we have developed an EUD environment to capture these parameters based on a generic meta-model for configuration and control of recommender systems, independent of the system's architecture. It was then implemented as a RecSys-Creator browser plug-in wizard employing a simple-to-use graphical interface as well as code generation components. The resulting scripts are executed on client-side through complementary browser extensions, such as the popular Greasemonkey. Many of these augmentation engines also exist for mobile browsers, making the artifacts generated in our EUD environment also executable on mobile devices. Recommender system functionality is consumed from a third-party server, to augment rating functionality and recommended items into web pages. In an evaluation survey with 30 end-users with varying technical backgrounds the idea of the approach was well received. Whereas widget positioning should be improved technically as well as with respect to the user interface, the approach has proven to work and the created scripts function correctly for a variety of different websites, without requiring any programming.

**Potential extensions and future work** For extending the presented approach several ideas come to mind. To begin with, due to changes in browser APIs and the rise of a standard for web extensions,[25] we are currently planning to migrate our wizard to use the new stable WebExtensions API. This way, it will not be limited to Firefox, but it should run on all current browsers implementing this future standard (currently also Chrome, Edge, Opera).

The asynchronous workflow of web augmentation recommender scripts (1. view generation, 2. querying recommender service, 3. view modification) offers certain distinctive possibilities. For instance, it could be investigated how the user model can be enriched with other information that can be extracted from the page, including recommendations or personalization artifacts provided by the site (i. e., learning about the user through items recommended to him on sites that already provide personalization). Regarding recommendation quality, whereas it has been suggested that it is difficult to measure the user's perceived quality with traditional metrics [10], the resulting systems could be evaluated performing a direct comparison between original and additionally augmented recommendations (e. g., performing a systematic user study). Related to this, besides reordering or replacing, original and additionally computed recommendations could also be mixed using different strategies.

Also, as the sheer amount of available user scripts on the Web indicates, plentiful of options exist to adapt web pages and applications, which implies that there also could be many more options for extracting user profiles and injecting recommendations within the browsing process.

Concerning the operation domain of created recommenders we could differentiate between the configuration of a view for creating user profiles (i.e., calculating user similarities) and the range of items for giving recommendations (i. e., calculating item

[25] https://browserext.github.io/browserext/

recommendations). Thereupon, tracking of user behaviour across multiple websites may yield comprehensive profile information for improving recommendations on individual sites (cf. [30]). For instance, a user reading about a historic topic on an encyclopedia site could be provided with recommendations for related board games on a gaming site, further giving options to obtain matching games in libraries and shops near his/her current location. However, for sharing such comprehensive profiles an appropriate control mechanism for script creators as well as users is essential to maintain the user's trust in the system. In this context, providing provenance information to explain given recommendations to interested users might also be beneficial.

Regarding algorithm configuration, the variety of setting options of the employed Lenskit framework[24] could be exploited to offer further customization options to the script creator (potentially set from clients via an API using Lenskit's Groovy-based DSL). In this manner users could specify preferences regarding recommender persistence, serendipity, or privacy (however, potentially impacting accuracy). In a related matter, an interesting outlook is that the used recommender algorithms, based on collaborative filtering, could be improved, changed or combined with others independently of the EUD approach proposed in this paper. In this work we focus on the end-user programming techniques required to enable end-users to define the recommender system's aspects they perceive, i.e., mostly related to how user interface components are defined and woven in relation to a particular web site's DOM. Other works have analyzed the problem of how end-users control recommender algorithms [14]. In this regard, we believe that our approach can easily be extended with new features. For instance, in other works we have also proposed an end-user mobile web augmentation approach [7], that enables end-users to define context-aware augmentations from their mobile web browsers. This approach could be easily integrated with the approach presented in this paper by defining an extension of the proposed meta-model according to these new features.

Beyond the configuration of recommender algorithms, extensions could allow to create completely different architectures of recommender systems, allowing the script creator to choose, for instance, a peer-to-peer architecture with models and recommendations computed entirely on client-side, and adding specified noise before data exchange with neighbors to increase privacy. Further, to broaden its applicability, it could be investigated how the proposed meta-model can be extended from recommender systems to specify configuration and control to cover all kinds of personalization and profiling architectures, for instance, for decentralized learning of models [37].

On top of that, a generic DSL, maybe a "recommender systems description language (RSDL)", could be developed to configure all kinds of recommender system architectures, based on a refined version of our meta-model, and using established reference models from literature as extensions (e.g., for modeling users and items). Means for serialization of system configurations could then also aid the versioning and modification of created scripts. In this context, required server components (depending on the system architecture) could be hosted by a *generic* platform-as-a-service *provider* of cloud computing *services*, possibly also offering to utilize Lenskit. Apart from parallelization this could also alleviate potential scalability issues that arise from communication overhead in large-scale application scenarios with our current implementation. Such an RSDL could then serve as a standardized reference model for providers of mobile and web applications to describe their integrated recommender and personalization systems, to further give users an opportunity to explicitly configure their personalization artifacts based on their privacy preferences (similar to the "Do Not Track" function of modern browsers). Ultimately, the rise of such architectures

may be only a small step for application providers, but one giant leap for the empowerment of the crowds longing for customization on the Web.

**Funding** Open access funding provided by Johannes Kepler University Linz.



## Appendix : Running RecSys-Creator recommender scripts

This appendix provides several screen-shots for running user scripts, utilizing the alternative capabilities of our RecSys-Creator wizard, as explained in Section 4.


**Fig. 8** Example: adapted CocktailScout website (cocktailscout.de) with augmented recipe recommendations (right menu bar; box "Recommendations"). In contrast to Fig. 1 in the introduction, where the augmentation user script was implemented manually, for this example it was created with the RecSys-Creator wizard in just a few minutes


**Fig. 9** Example: adapted MediaWiki wiki site (www.c64-wiki.de), running a user script created according to the demonstration from Section 4. The script augments a rating widget (below the title heading) as well as recommendations (in the left menu)


**Fig. 10** Example: adapted bookstore page of the Argentinian retailer Cúspide (cuspide.com) with an augmented rating widget (center/top, directly below the book title "Notas de Viaje") as well as augmented recommendations (further below, heading "Recommendations"). The bottom of the figure shows the login form to authenticate against the PAA server directly, which is displayed instead of item recommendations as long as the user is not logged in

## References


1. Adomavicius G, Tuzhilin A (2005) Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans Knowl Data Eng 17(6):734–749
2. Aghaee S, Pautasso C (2014) End-user development of mashups with naturalmash. Vis Lang Comput 25(4):414–432
3. Ankolekar A, Vrandecic D (2008) Kalpana—enabling client-side web personalization. In: Duval E (ed) Proceedings of hypertext 2008. HT '08, ACM, Pittsburgh
4. Barbosa ADM (2014) Privacy-enabled scalable recommender systems. Ph.D. thesis, University of Nice Sophia Antipolis, France. https://tel.archives-ouvertes.fr/tel-01135312
5. Barrett R, Maglio PP (1998) Intermediaries: new places for producing and manipulating web content. Comput Netw ISDN Syst 30(1):509–518
6. Bobadilla J, Ortega F, Hernando A, Gutiérrez A (2013) Recommender systems survey. Knowl-Based Syst 46:109–132
7. Bosetti G, Firmenich S, Gordillo SE, Rossi G, Winckler M (2017) An end user development approach for mobile web augmentation. Mob Inf Syst 2525367:1–2525367:28. https://doi.org/10.1155/2017/2525367
8. Brusilovsky P (2007) Adaptive navigation support. In: Brusilovsky P, Kobsa A, Nejdl W (eds) The adaptive web, LNCS, vol 4321. Springer, Berlin, pp 263–290
9. Cantador I, Fernández-Tobías I, Berkovsky S, Cremonesi P (2015) Cross-domain recommender systems. Springer US, Boston, pp 919–959. https://doi.org/10.1007/978-1-4899-7637-6_27
10. Cremonesi P, Garzotto F, Negro S, Papadopoulos A, Turrin R (2011) Comparative evaluation of recommender system quality. In: CHI '11 extended abstracts on human factors in computing systems. ACM, New York, pp 1927–1932
11. Della Penna G, Magazzeni D, Orefice S (2010) Visual extraction of information from web pages. Vis Lang Comput 21(1):23–32
12. Díaz O, Arellano C (2015) The augmented web: rationales, opportunities, and challenges on browser-side transcoding. ACM Trans Web 9(2):8:1–8:30
13. Díaz O, Arellano C, Aldalur I, Medina H, Firmenich S (2014) Towards the personal web: empowering people to customize web content. In: Web information systems engineering (WISE) 2014, Lecture Notes in Computer Science, vol 8786. Springer International Publishing
14. Ekstrand MD, Kluver D, Harper FM, Konstan JA (2015) Letting users choose recommender algorithms: an experimental study. In: Proceedings of the 9th ACM conference on recommender systems. RecSys '15. Association for Computing Machinery, New York, pp 11–18. https://doi.org/10.1145/2792838.2800195
15. Elmisery AM, Rho S, Sertovic M, Boudaoud K, Seo S (2017) Privacy aware group based recommender system in multimedia services. Multimed Tools Appl 76:26103
16. Eynard D (2008) Using semantics and user participation to customize personalization. Tech. rep., HP Labs. http://www.hpl.hp.com/techreports/2008/HPL-2008-197.html
17. Firmenich D, Firmenich S, Rivero J, Antonelli L (2014) A platform for web augmentation requirements specification. In: Web engineering, lecture notes in computer science, vol 8541. Springer International Publishing, pp 1–20
18. Fu X, Budzik J, Hammond KJ (2000) Mining navigation history for recommendation. In: Proceedings of the 5th international conference on intelligent user interfaces. IUI '00. ACM, New York, pp 106–112
19. Garrido A, Firmenich S, Rossi G, Grigera J, Medina-Medina N, Harari I (2013) Personalized web accessibility using client-side refactoring. Internet Computing, IEEE 17(4):58–66
20. Gonzalez R, Firmenich S, Fernández A, Rossi G, Velez D (2020) An approach to build P2P web extensions. In: Bieliková M, Mikkonen T, Pautasso C (eds) Web Engineering—20th international conference, ICWE 2020, Helsinki, Finland, June 9–12, 2020, Proceedings. Lecture Notes in Computer Science, vol 12128. Springer, pp 467–474. https://doi.org/10.1007/978-3-030-50578-3_31
21. Guo H, Chen J, Wu W, Wang W (2009) Personalization as a service: the architecture and a case study. In: Proceedings of the first international workshop on cloud data management. CloudDB '09. ACM, New York, pp 1–8
22. Harper FM, Xu F, Kaur H, Condiff K, Chang S, Terveen L (2015) Putting users in control of their recommendations. In: Proceedings of the 9th ACM conference on recommender systems. RecSys '15. Association for Computing Machinery, New York, pp 3–10. https://doi.org/10.1145/2792838.2800179
23. Hendry, Pramadharma H, Chen RC (2015) Building browser extension to develop website personalization based on adaptive hypermedia system. In: Current approaches in applied artificial

intelligence. Lecture Notes in Computer Science, vol 9101. Springer International Publishing, pp 316–325
24. Hijikata Y, Kai Y, Nishida S (2014) A study of user intervention and user satisfaction in recommender systems. J Inf Process 22(4):669–678
25. Jawaheer G, Weller P, Kostkova P (2014) Modeling user preferences in recommender systems: a classification framework for explicit and implicit user feedback. ACM Trans Interact Intell Syst 4(2):8:1–8:26
26. Kleek MV, Smith DA, Shadbolt NR, Schraefel M (2012) A decentralized architecture for consolidating personal information ecosystems: The webbox. In: Personal information management workshop, ACM conference on computer supported cooperative work
27. Knijnenburg B, Bostandjiev S, O'Donovan J, Kobsa A (2012) Inspectability and control in social recommenders. In: Proceedings of the 6th ACM conference on recommender systems
28. Kobsa A, Knijnenburg BP, Livshits B (2014) 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 '14, pp 81–90, Toronto, Canada
29. Kolias C, Kolias V, Kambourakis G, Kayafas E (2013) A client-side privacy framework for web personalization. Springer, Berlin, pp 297–316
30. Kotkov D, Wang S, Veijalainen J (2017) Improving serendipity and accuracy in cross-domain recommender systems. In: Monfort V, Krempels KH, Majchrzak TA, Traverso P (eds) Web information systems and technologies. Springer International Publishing, Cham, pp 105–119
31. Malle B, Giuliani N, Kieseberg P, Holzinger A (2017) The more the merrier—federated learning from local sphere recommendations. In: Holzinger A, Kieseberg P, Tjoa AM, Weippl E (eds) Machine learning and knowledge extraction. Springer International Publishing, Cham, pp 367–373
32. Newell C, Miller L (2013) Design and evaluation of a client-side recommender system. In: Proceedings of ACM conference on recommender systems. ACM, New York, pp 473–474
33. Ricci F, Rokach L, Shapira B (eds) (2015) Social recommender systems. Springer US, Boston
34. Simpson JE (2002) XPath and XPointer—locating content in XML documents. O'Reilly Media ISBN: 978-0-596-00291-6
35. Son LH (2016) Dealing with the new user cold-start problem in recommender systems: a comparative review. Inf Syst 58:87–104
36. Sparling EI, Sen S (2011) Rating: how difficult is it? In: Proceedings of the fifth ACM conference on recommender systems. RecSys '11. ACM, New York, pp 149–156
37. Vanhaesebrouck P, Bellet A, Tommasi M (2017) Decentralized collaborative learning of personalized models over networks. In: International conference on artificial intelligence and statistics (AISTATS). Fort Lauderdale. https://hal.inria.fr/hal-01533182
38. Wischenbart M, Firmenich S, Rossi G, Wimmer M (2015) Recommender systems for the people—enhancing personalization in web augmentation. In: Proceedings of the IntRS workshop, ACM conference on recommender systems, Vienna, Austria, September 19, 2015, pp 53–60
39. Xie Q, Xiong F, Han T, Liu Y, Li L, Bao Z (2018) Interactive resource recommendation algorithm based on tag information. World Wide Web
40. Zheng X, Luo Y, Sun L, Ding X, Zhang J (2017) A novel social network hybrid recommender system based on hypergraph topologic structure. World Wide Web

## Affiliations

**Martin Wischenbart[1] · Sergio Firmenich[2,3] · Gustavo Rossi[2,3] · Gabriela Bosetti[2] · Elisabeth Kapsammer[4]**

Sergio Firmenich
sergio.firmenich@lifia.info.unlp.edu.ar

Gustavo Rossi
gustavo.rossi@lifia.info.unlp.edu.ar

Gabriela Bosetti
gabriela.bosetti@lifia.info.unlp.edu.ar

Elisabeth Kapsammer
elisabeth.kapsammer@jku.at

1 Johannes Kepler University Linz, Linz, Austria

2 Laboratorio de Investigación y Formación en Informática Avanzada (LIFIA), Universidad Nacional de La Plata, La Plata, Argentina

3 CONICET, La Plata, Argentina

4 Department of Cooperative Information Systems (CIS), Johannes Kepler University Linz, Linz, Austria