RecTurk: Constraint-based Recommendation based on Human Computation*

Alexander Felfernig
Institute for Software Technology
A-8010 Graz, Austria
afelfern@ist.tugraz.at

Sarah Haas
Institute for Software Technology
A-8010 Graz, Austria
sarah.haas@student.tugraz.at

Gerald Ninaus
Institute for Software Technology
A-8010 Graz, Austria
gninaus@ist.tugraz.at

Michael Schwarz
Institute for Software Technology
A-8010 Graz, Austria
michael.schwarz@student.tugraz.at

Thomas Ulz
Institute for Software Technology
A-8010 Graz, Austria
thomas.ulz@student.tugraz.at

Martin Stettinger
Institute for Software Technology
A-8010 Graz, Austria
mstettinger@ist.tugraz.at

ABSTRACT

The development of constraint-based recommender applications is still challenged by the knowledge acquisition bottleneck. The management of the underlying constraint sets is often accompanied by additional efforts related to the information exchange between knowledge engineers and domain experts. In this paper we introduce the RecTurk research prototype which supports the development of constraint-based recommender applications on the basis of Human Computation concepts. Thus we substitute complex knowledge engineering tasks with simple micro-tasks that can be performed by persons without experiences in constraint-based recommender application development. We introduce the basic concepts currently integrated in RecTurk and report the results of a first user study that evaluated the applicability of RecTurk in three item domains.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.2 [**Design Tools and Techniques**]: [User interfaces]

General Terms

Human Factors, Algorithms

Keywords

Constraint-based Recommendation, Human Computation

1. INTRODUCTION

In this paper we present approaches that support an easier development of *constraint-based recommenders* [3, 5]. These systems are – beside *critiquing-based recommenders* [1, 2] – a major representative of knowledge-based recommenders [1]. Critiquing-based recommenders focus on a navigation-based paradigm where users are navigating in the item (product) space by articulating critiques, for example, *a cheaper camera* or *a more fancy car* and the recommender retrieves new alternatives on the basis of similarity metrics. Constraint-based recommenders focus on a search-based user support where users articulate requirements and the recommender system searches for a solution.

In the following we focus on constraint-based recommenders [3] that exploit constraints (rules) in order to determine recommendations. Such systems are especially useful in scenarios related to *high-involvement items* where consumers typically invest more time and efforts (e.g., evaluation of prices, quality, and services) to assure they make the best choice. They are prepared to invest additional efforts since the purchase decision is often associated with high investment costs and/or is related to increases in or losses of prestige. The underlying items are typically complex which requires constraint-based representations. Examples of items for which constraint-based recommenders have already been developed, are digital cameras, financial services, and software engineering methods.

The major *advantages* of constraint-based recommenders are (1) *deep knowledge about the item domain* which is exploited for taking into account existing domain restrictions (e.g., in the automotive domain certain car bodies must not be combined with certain engines), (2) *deep explanations* for recommendation results (e.g., the user requirements *high return rate* and *low risk* are directly related to the item property *longterm investment period*), and (3) automated diagnosis and repair of inconsistent requirements (e.g., a user has to change his/her *willingness to take risks* due to the fact that a *short investment period* and *high return rates* are preferred).

One major drawback of constraint-based recommendation technologies are time-intensive knowledge acquisition processes that are needed to identify the item domain knowledge (properties and the corresponding constraints). Efforts related to the communication overhead between item domain experts and knowledge engineers are often summarized (denoted as) *knowledge acquisition bottleneck*. This phenomenon is one of the major impediments for a more wide-spread application of constraint-based recommenders.

*This work presented in this paper has been conducted in the project PeopleViews funded by the Austrian Research Promotion Agency (843492).

CrowdRec 2014 6 Oct. 2014, Foster City, CA, USA

Intuitive knowledge acquisition for constraint-based recommenders can have an enormous impact on user acceptance. Our major goal is to provide engineering techniques that allow persons without technical expertise in the development of constraint-based recommenders to easily develop their own recommenders or contribute to recommenders of interest. In this context we introduce the RecTurk research prototype[1] that helps to easily define and execute *constraint-based recommenders* on the basis of micro-contributions [8] of a group of users. RecTurk asks simple questions (in the form of micro-tasks [8]) and then (from the given answers) derives a recommendation knowledge base. As a working example in this paper, we will use the domain of *digital cameras*. However, RecTurk users are free to choose a (reasonable) application domain on their own.

The name RecTurk was inspired by the amazon.com platform *amazon mechanical turk* which hosts a community that completes so-called Human Computation tasks (see [10]).[2] Typically, such tasks are easy to complete by humans but often infeasible for computers. Examples of such tasks are the quality comparison of pictures or the manual identification of item properties from a web page. Another example of the application of Human Computation is CAPTCHA (Completely Automated Public Turing Test To Tell Computers and Humans Apart – www.captcha.net) which is exploited, for example, to protect websites against bots.

Human Computation style approaches have already been applied in the context of recommender systems. For example, WikiLens [8] is a recommendation environment that provides user communities the possibility to cooperatively develop recommenders in an open environment where users are able to introduce new items, comment on items, and search for items. The basic principles of community-maintained recommenders [8] are taken into account in RecTurk. Another application of Human Computation in the recommendation context is the work of [9] (*Matchin*). *Matchin* is based on the idea of eliciting preferences for large datasets by asking users what a "random" person would prefer when having to choose between alternatives. The difference between the work of [8, 9] and RecTurk is that RecTurk-collected and -compiled recommendation knowledge can be exploited for constraint-based recommenders [3] that include functionalities such as deep explanations and intelligent diagnosis and repair for inconsistent requirements.

The remainder of this paper is organized as follows. In Section 2 we sketch the application of *Human Computation* in constraint-based recommendation. The RecTurk user interface is presented in Section 3. The results of a first empirical study are presented in Section 4. The paper is concluded with Section 5.

2. RECTURK APPROACH

On the basis of a working example from the domain of *digital cameras* we show how RecTurk can be exploited for the development and application of constraint-based recommenders. RecTurk can be used in two basic modes.

A. Modeling Mode. Recommender applications (items, properties, and constraints) can be defined and managed in the *modeling mode*. Users engaged in the modeling mode can contribute to the construction of constraint-based recommenders by simply completing a set of so-called *micro-tasks* [8]. Micro-tasks currently offered by the RecTurk environment are summarized in Table 1.

micro-task	description
define recommender	create a new recommender
add item	add a new item to recommender
add item property	add a new item property
describe item property	characterize selected property(ies)
rate item	provide a new rating for an item
rate recommender	evaluate a recommender

Table 1: Types of RecTurk *micro-tasks*.

In RecTurk, items (see, e.g., Table 2) can be added to a recommender and characterized with regard to their basic properties (e.g., whether a digital camera is well-suited for *sports* photography). Product (item) information is entered directly by users (the *add-principle* [8]). Each user is allowed to add new items but also to adapt the information of existing ones. Only administrators are allowed to insert, update, and delete item properties. For our example knowledge base we are interested in the following item properties (see Table 3): portrait (support of good portrait pictures), sports (good support of sports photography), macro (good support of macro photography), and price (price segment). Item properties describe the way in which users can *evaluate an item* and how they can *specify their requirements* in the *recommendation mode*. For each item property we have to specify the question posed to the user (see Table 3).

name	description	link	user
300D	300D is ...	...	Joe
7D	7D is ...	...	Ed
Mark III	Mark III is ...	...	Mary

Table 2: Example items (CANON EOS cameras).

property	domain	question to user	user
portrait	{yes, no}	portrait shots?	Mary
sports	{yes, no}	sports shots?	Mary
macro	{yes, no}	macro shots?	Mary
price	{0–300, ... , ≥1500}	accepted price?	Mary

Table 3: Example item properties.

B. Recommendation Mode. Recommender applications (developed in the modeling mode) are available for public use in the *recommendation mode*. As mentioned, we use the domain of *digital cameras* as a simple working example – however, in RecTurk users are free to choose their own item domain. For the purposes of our working example we assume the existence of users (see Table 4). These users contribute to the construction of the recommender knowledge base and apply the resulting RecTurk recommender (in our case: a simple digital camera recommender).

Recommendation Task. In RecTurk, a *recommendation task* (P, A, F, R) consists of an item table P, a set A of item

[1] recturk.ist.tugraz.at.

[2] *Amazon mechanical turk* was inspired by *The Turk* (18^{th} century) which was assumed to be a chess playing automaton but then revealed of being a chess master hidden inside.

user	name	r	p	s	m	p
Joe	300D	3.0	yes	no	yes	0–300, 300–600
Ed	7D	4.0	yes	yes	yes	600–1500, ≥1500
Peter	7D	4.0	yes	yes	yes	600–1500, ≥1500
Ann	7D	4.0	yes	yes	yes	600–1500, ≥1500
Ed	Mark III	5.0	yes	no	no	≥1500
Alex	300D	3.0	yes	no	no	0–300, 300–600
Leo	300D	3.0	yes	no	no	0–300
Mary	Mark III	5.0	yes	yes	-	≥1500
Alex	Mark III	5.0	yes	no	no	≥1500
John	Mark III	5.0	yes	no	no	600–1500, ≥1500

Table 4: Example: user-specific filter constraints (r = rating, p = portrait, s = sports, m = macro).

properties (attributes of P), a set F of filter constraints that select items from P depending on the requirements in R, and a set of customer requirements (R). In this context, P, A, F are denoted as *recommender knowledge base*. The identification of recommendation candidates can be described as a conjunctive query $\sigma_{R \cup F}[P]$ where σ represents the selection operator of a conjunctive query that is executed on the item table P taking into account the selection criteria in $R \cup F$. The basic steps to derive and apply a RecTurk recommender knowledge base will be discussed in an example.

Before we are able to apply a RecTurk recommender, we have to define the relevant properties of this recommender, i.e., the basic elements needed for the definition of a recommendation task. Let us assume that the RecTurk recommender DigiCams has already been defined (micro-task *define recommender*) and that this task has been performed by the user *Mary*. Thereafter, users start entering items that should be integrated in DigiCams from their point of view. In our working example the users *Joe*, *Ed*, and *Mary* add the items (Canon cameras) *300D*, *7D*, and *Mark III* (related micro-task: *add item* – see Table 2). After having created the RecTurk recommender DigiCams, *Mary* also added a set of item properties (see Table 3). An example of an item property is *portrait* which entails the information whether a camera provides a good support for portrait photography – the corresponding micro-task is *add item property*.

Since only creators of recommenders are currently allowed to define item properties, *Mary* is the only user who is allowed to do that in the DigiCams recommender (see Table 3 for an overview of item properties used in our working example). After item properties have been defined, other users are allowed to characterize items with regard to the defined item properties, for example, user *Peter* who evaluates the Canon EOS *7D* camera, concludes that the camera provides a good *macro* support *macro=yes*.[3] For our example we assume that other users are specifying item properties as well (micro-task: *describe item property*). An overview of these property specifications is given in Table 4.

User-specific Filter Constraints. Each user is allowed to rate each RecTurk item; a user-specific rating provided for an item is denoted as *user-specific filter constraint*. Such constraints reflect item evaluations of a specific user and thus form the basis for defining the relationship between user requirements (instantiations of item properties) and items from the viewpoint of a user. For example, user *Mary* (see Table 4) recommends the camera *Mark III* to users interested in good portrait and sports photos. Her rating for this camera is 5.0 (on a scale of 1.0, 2.0, .., 5.0), she specifies the price category with ≥1500, and does not provide an evaluation for the item property *macro* since she has no further technical information. Users are also allowed to adapt and delete their own user-specific filter constraints. Note that when defining a user-specific filter constraint, there is no need that a user specifies values for all item properties. For example, Mary did not specify a value for the item property *macro* of the *EOS Mark III*. It is also allowed that one user specifies more than one value per item property. For example, the *7D* can be recommended **to users who accept a price of 600-1500 or a price ≥1500**. On a more formal level, the user-specific filter constraint introduced by *Mary* would look like as follows (*noreq* indicates that no value was specified for the property by the recommender user): *if (portrait = yes or noreq) and (sports = yes or noreq) and (price ≥ 1500 or noreq) then include (Mark III).*

Recommendation-relevant Filter Constraints. On the basis of *user-specific filter constraints* (see, e.g., Table 4), RecTurk applies the following strategy for aggregating these filter constraints to *recommendation-relevant filter constraints*. For each item, the user-specific filter constraints are aggregated *per item property*. An example of such an aggregation is shown in Table 5 (aggregation of the user-specific filter constraints related to the item Mark III). On a formal level, the recommendation-relevant filter constraint for the item Mark III looks like as follows (*noreq* indicates that no value was specified for the item property by the recommender user): *if (portrait = yes or noreq) and (sports = yes or noreq) and (price between 600-1500 or price ≥ 1500 or noreq) then include (Mark III).*

The complete set of recommendation-relevant filter constraints is shown in Table 6. These constraints are activated in the RecTurk recommendation mode. In RecTurk, filter constraints can automatically be derived from micro-contributions. This way the system is able to automatically derive constraints which represent the knowledge base of a constraint-based recommender. As an alternative to time-consuming communication processes between item domain experts and knowledge engineers, RecTurk provides a reasonable alternative which has the potential to reduce overheads related to the knowledge acquisition bottleneck and – due to a simple way of entering recommendation knowledge – makes constraint-based recommendation technologies accessible for a broader user group.

For each combination of *name* × *property value* (see Table 4), RecTurk determines a support value (see Formula 1). For example, *support*(300D, portrait=yes)=1.0 means that the users completely agree on the fact that the Canon EOS 300D camera is a good recommendation when a user wants to take portrait pictures. The complete set of *support* values for item property values derived from the user-specific filter constraints (Table 4) is included in Table 7. In Formula 1, *co-occurrence(name, value)* indicates how often the combination of item *name* and property *value* can be found. For example, *co-occurrence(300D, portrait=yes)* = 3.

[3] In the current RecTurk version it is not possible to define the degree of support, however, this is already planned for the next system version.

user	name	rating (r)	portrait	sports	macro	price
Ed	Mark III	5.0	yes	no	no	$\geq$1500
Mary	Mark III	5.0	yes	yes	-	$\geq$1500
Alex	Mark III	5.0	yes	no	no	$\geq$1500
John	Mark III	5.0	yes	no	no	600–1500, $\geq$1500
Aggregate	Mark III	5.0 (rating)	yes	no, yes	no	600–1500, $\geq$1500

Table 5: Example: Aggregation of user-specific filter constraints related to item *Mark III*. The result is a recommendation-relevant filter constraint for the Mark III.

name	rating (r)	portrait	sports	macro	price
300D	3.0	yes	no	no,yes	0–300, 300–600
7D	4.0	yes	yes	yes	600–1500, $\geq$1500
Mark III	5.0	yes	no, yes	no	600-1500, $\geq$1500

Table 6: Complete set of *recommendation-relevant filter constraints* derived from the *user-specific filter constraints* depicted in Table 4.

$$support(name, value) = \frac{\#co - occurrence(name, value)}{\#occurrence(name)} \quad (1)$$

Recommendation-relevant filter constraints can be applied for determining recommendations for users. For each item (a camera in our case) there exists exactly one aggregate (=recommendation-relevant filter constraint).[4] If less than 3 users have entered a user-specific filter constraint for that item, the corresponding recommendation is combined with a hint that the support for this item is below the threshold (see also [8]). In addition to *support* values, each item has an average rating which is derived from the information in Table 4. The average rating can be determined on the basis of Formula 2, for example, the average rating of the Canon EOS 300D camera is 3.0 (see Table 6).

$$rating(name) = AVG(name, userrating) \quad (2)$$

Determining Recommendations. Finally, we are interested in determining a recommendation for a concrete set of user requirements (*reqs*). Let us assume that the requirements of the current user (interacting with DigiCams) are R = {portrait=yes, sports=no}[5] which completes the recommendation task (P, A, F, R). Executing the query $\sigma_{R \cup F}[P]$ results in a pre-selection of the cameras *300D* and *Mark III*.

The ordering of alternative recommendations can be determined on the basis of the *utility* function (see Formula 3) where the *rating* (*r*) and the *support* values are taken from the information in Tables 6 and 7. For example, *utility*(300D,[portrait=yes, sports=no])=6.0 and *utility*(Mark III, [portrait=yes, sports=no])=8.75. Consequently, the cameras will be presented to the user in the following order: *Mark III* $\rightarrow$ *300D*. The detailed utility calculation is depicted in Table 8.

$$utility(name, reqs) = r(name) \times \sum_{i=1..n} support(name, reqs[i]) \quad (3)$$

Diagnosis & Repair. Furthermore, recommendation relevant filter constraints can be inconsistent with a set of requirements (R). For example, a digital camera in the price segment of $0 - 300$ that provides a good support for *sports* photography does not exist (Table 6). In such a situation,

[4]Redundancy elimination is planned for future versions.

[5]A user does not have to specify all properties.

name	portrait		sports		macro		price	
300D	yes	1.0	no	1.0	no	.66	0–300	1.0
					yes	.34	300–600	.66
7D	yes	1.0	yes	1.0	yes	1.0	600–1500	1.0
							$\geq$1500	1.0
Mark III	yes	1.0	no	.75	no	.75	600-1500	.25
			yes	.25			$\geq$1500	1.0

Table 7: Support values for item property values derived from *user-specific filter constraints* (Table 4).

RecTurk libraries include functionalities of model-based diagnosis – for related details we refer the reader to [4].

name	support portrait =yes	support sports =no	total	rating	utility
300D	1.0	1.0	2.0	3.0	6.0
Mark III	1.0	0.75	1.75	5.0	8.75

Table 8: Calculation of the *item utilities* (Formula 3) of *300D* and *Mark III* for the user requirements [portrait=yes, sports=no]. The *EOS 7D* is not included since there is no support for the individual user requirement *sports=no*.

3. RECTURK USER INTERFACE

In the following paragraphs we provide an overview of the basic functionalities provided by the RecTurk user interface. RecTurk can be operated in two different modes (see Figure 1).[6] *First*, a recommender is developed (modeling mode) by exploiting the micro-contributions of users to derive a set of recommendation-relevant filter constraints. Users can provide micro-contributions in different ways, for example, by entering new items into the system or by evaluating items that have already been entered by other users. For example, the user selects the item *EOS Mark III* and specifies values for *portrait, sports, macro*, and *price* (a user-specific filter constraint). RecTurk stores the new (user-specific) filter constraint (see, e.g., Table 4) and recalculates the *support* values of the individual item property values (see Formula 1) and the item-specific average ratings (see Formula 2). In RecTurk, only logged-in users

[6]RecTurk is currently available only in German.

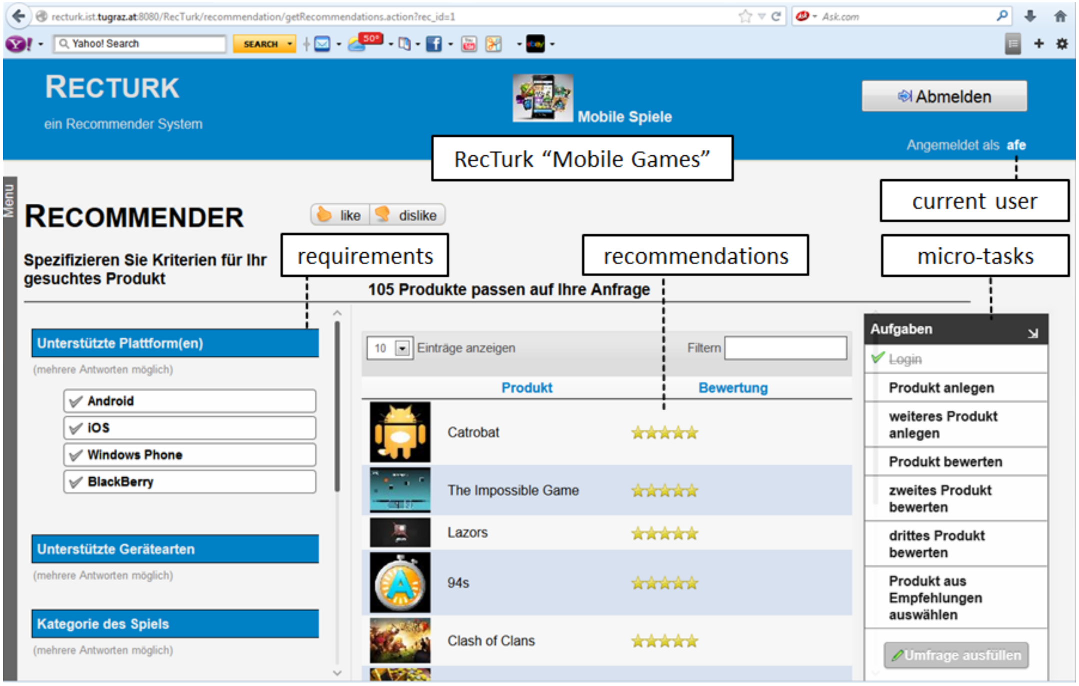

Figure 1: RecTurk MobileGames in *recommendation mode* – micro-tasks are displayed on the right hand side.

are allowed to provide micro-contributions. *Second*, RecTurk recommenders (automatically generated from micro-contributions) are open for use, i.e., no login is needed for applying RecTurk recommenders. An example of a RecTurk user interface for the Mobile Games recommender is depicted in Figure 1. This recommender has been developed by the participants of the user study reported in Section 4. In our working example (recommendation of digital cameras), the user could specify his/her requirements (portrait = yes, sports = no). RecTurk selects relevant items using the table of recommendation-relevant filter constraints (e.g., Table 6) and ranks the results on the basis of Formula 3.

4. EVALUATION

In this section we will first discuss the benefits of Human Computation based approaches compared to centralized ones where one or few knowledge engineers are in charge of developing and maintaining constraint-based recommenders. Thereafter, we report the results of a study that has been conducted at the Graz University of Technology.

Benefits of Human Computation based Approaches. The development of a constraint-based recommender application can cause development efforts in the range of a couple of man months [7]. In contrast, the basic RecTurk applications of our user study have been developed within the scope of two days. The major reason for this increase in efficiency is the provision of easy micro-tasks that are accessible to a larger group of users. From our commercial recommender projects (see, e.g., [7]) we know that *scalability* is one of the major challenges of constraint-based recommender development (see also [8]). In many scenarios, one or few knowledge engineers are in charge of developing and maintaining knowledge bases. Due to the increasing number of items offered via recommender applications, development and maintenance operations become infeasible for one or few persons. This situation then triggers a demand for keeping recommendation knowledge up-to-date in a more sophisticated fashion. RecTurk is a first step towards more scalable development techniques which will increase the popularity of constraint-based recommenders.

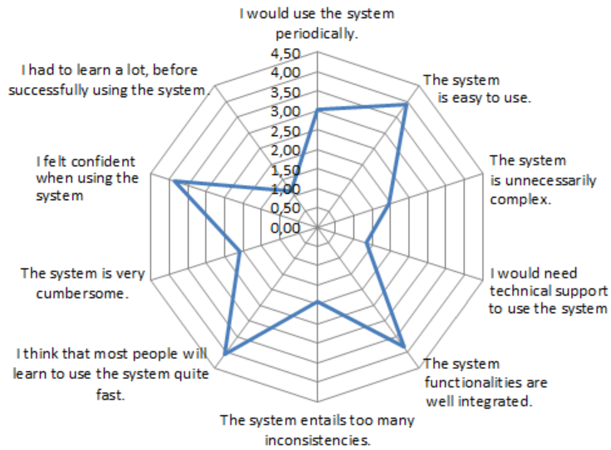

Figure 2: Results of the SUS evaluation.

RecTurk *usability.* This user study has been conducted at the Graz University of Technology. N=161 (15% female, 85%male) interacted with the RecTurk environment and developed three different constraint-based recommender applications (MobileGames, WorldCities, and SkiResorts). The distribution of expertises in the application of recommendation technologies is depicted in Figure 3.

Recommenders and item properties have been created before the study started. The micro-tasks (see Figure 1) of the study participants were to enter new items, to describe

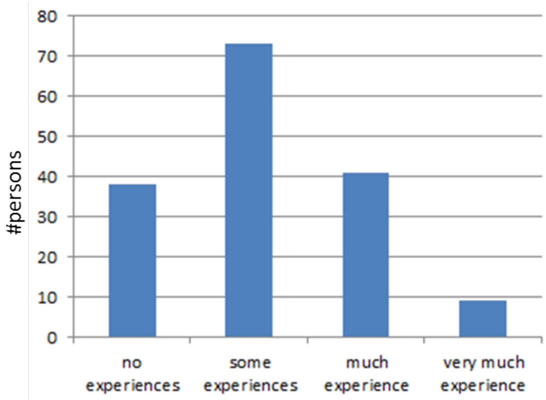

Figure 3: Experiences with recommenders.

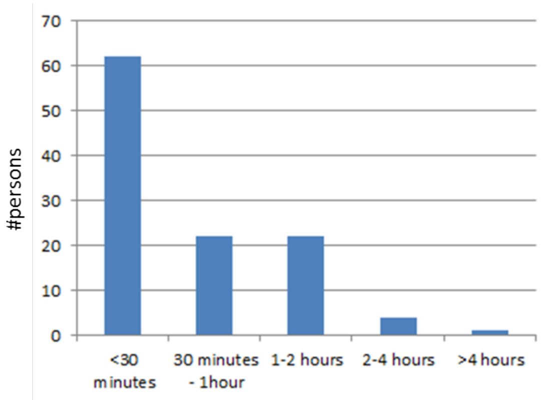

Figure 4: Willingsness to contribute (mins/week).

and evaluate items, and to apply one of the three RecTurk recommender applications where the task was to find and select a preferred item. After interacting with the RecTurk environment, users had to fill out a questionnaire based on the system usability scale (SUS) and to answer further questions. The results of the system usability scale (SUS) evaluation of RecTurk are depicted in Figure 2.

Willingness to contribute. Besides analyzing usability aspects, we were interested in whether users can imagine to contribute to the development of RecTurk recommenders, and – if yes – to which extent they are willing to support recommender development (in terms of the micro-tasks they had to complete within the scope of the user study). Figure 4 summarizes the user feedback regarding this question of available time for recommender development. The majority of those users who would like to contribute to RecTurk recommender development (69% of all users) would contribute within a time frame of less than 30 minutes per week (56% out of those who want to contribute).

Further application domains. Finally, we wanted to know for which item domains study participants have an interest in developing their own RecTurk recommender applications (N = 80). The most prominent item domains were electronic equipment, restaurants, bars, holiday trips, hotels, games, films, computers, and smartphones.

5. CONCLUSIONS

Constraint-based recommendation technologies still suffer from the knowledge acquisition bottleneck: knowledge engineers still have to invest huge efforts to be able to guarantee that the developed constraint set is up-to-date. RecTurk technologies help to tackle this challenge by simplifying knowledge engineering tasks. This simplification is achieved by confronting users with simple micro-tasks such as entering an item and evaluating an item with regard to a list of item characteristics. This form of user input is exploited by the RecTurk environment for automatically deriving constraints which are the basis for the application of knowledge-based techniques such as knowledge-based reasoning, intelligent (deep) explanations, and automated diagnosis & repair. Our *future work* will include the development and comparison of further item ranking mechanisms, the development of additional micro-task patterns (for constraints, conflict resolution, and redundancy elimination [6]), the development of quality assurance and manipulation detection mechanisms, and the design of "games with a purpose" for knowledge base construction in RecTurk.

6. ADDITIONAL AUTHORS

Additional authors: Klaus Isak (SelectionArts Ltd., email: `k.isak@selectionarts.com`), Michael Jeran (Institute for Software Technology, email: `mjeran@ist.tugraz.at`), and Stefan Reiterer (Institute for Software Technology, email: `stefan.reiterer@ist.tugraz.at`).

7. REFERENCES

[1] R. Burke. Knowledge-based recommender systems. *Encyclopedia of Library and Information Systems*, 69(32):180–200, 2000.

[2] L. Chen and P. Pu. Critiquing-based recommenders: survey and emerging trends. *UMUAI*, 22:125–150, 2011.

[3] A. Felfernig and R. Burke. Constraint-based Recommender Systems: Technologies and Research Issues. In *ICEC'08*, pages 17–26, 2008.

[4] A. Felfernig, G. Friedrich, M. Schubert, M. Mandl, M. Mairitsch, and E. Teppan. Plausible Repairs for Inconsistent Requirements. In *IJCAI'09*, pages 791–796, Pasadena, California, USA, 2009. AAAI.

[5] A. Felfernig, B. Gula, and E. Teppan. Knowledge-based Recommender Technologies for Marketing and Sales. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 21(2):1–22, 2006.

[6] A. Felfernig, L. Hotz, C. Bagley, and J. Tiihonen. *Knowledge-based Configuration: From Research to Business Cases*. Elsevier/Morgan Kaufmann, 2014.

[7] A. Felfernig, K. Isak, K. Szabo, and P. Zachar. The VITA Financial Services Sales Support Environment. In *AAAI/IAAI 2007*, pages 1692–1699, Vancouver, Canada, 2007. AAAI.

[8] D. Frankowski, S. Lam, S. Sen, F. Harper, S. Yilek, M. Cassano, and J. Riedl. Recommenders Everywhere: the WikiLens Community-Maintained Recommender System. In *WikiSym'07*, pages 47–60. ACM, 2007.

[9] S. Hacker and L. vonAhn. Matchin: eliciting user preferences with an online game. In *CHI09*, pages 1207–1206. ACM, 2009.

[10] L. VonAhn. Human computation. *Technical Report*, CMU-CS-05-193, 2005.