The weHelp Reference Architecture for Community-Driven Recommender Systems — Short Position Paper

Swapneel Sheth, Nipun Arora, Christian Murphy, Gail Kaiser
Department of Computer Science, Columbia University, New York, NY 10027
{swapneel, nipun, cmurphy, kaiser}@cs.columbia.edu

ABSTRACT

Recommender systems have become increasingly popular. Most research on recommender systems has focused on recommendation algorithms. There has been relatively little research, however, in the area of generalized system architectures for recommendation systems. In this paper, we introduce *weHelp* - a reference architecture for social recommender systems. Our architecture is designed to be application and domain agnostic, but we briefly discuss here how it applies to recommender systems for software engineering.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architectures—*Domain-specific architectures*

General Terms

Human Factors, Design

Keywords

Recommender Systems, Reference Architecture

1. INTRODUCTION

Social recommender systems have become increasingly popular and ubiquitous, being used in a variety of different domains such as suggesting movies we might enjoy watching (e.g., Netflix), and things we might want to buy (e.g., Amazon), often based on community-driven "people like you..." paradigms. In this paper, we introduce *weHelp*: a reference architecture for social recommender systems - systems where recommendations are derived automatically from the aggregate of the activities of the system's users and thus reflect community interests and behaviors.

Our reference architecture is derived from three of our otherwise unrelated research projects that share a similar theme: providing suggestions aimed to help users employ particular software tools in some better way. **Retina** [6], a system targeted towards CS1 (intro to programming) courses, provides recommendations on how to fix compiler or runtime errors. The Retina system is directly relevant to software engineering (SE), albeit focused towards novice programmers. **genSpace** [5] provides recommendations to biomedical researchers on workflows that are executed within the geWorkbench [1] integrated genomics platform. Although genomics workflows are not directly germane to SE, they are similar to workflows used in IDEs. **COMPASS** [8] provides recommendations to programmers on how to parallelize code, based on past parallelizations of similar code. Though COMPASS is currently targeted to a niche community of parallel programmers, similar systems can be envisioned to apply more widely to other SE optimizations.

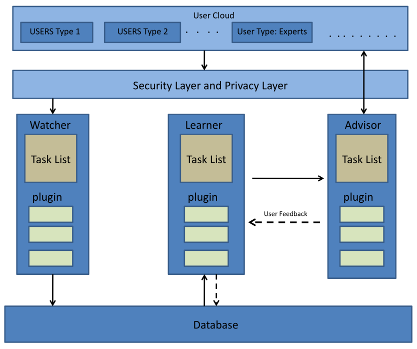


Figure 1: weHelp Reference Architecture Diagram

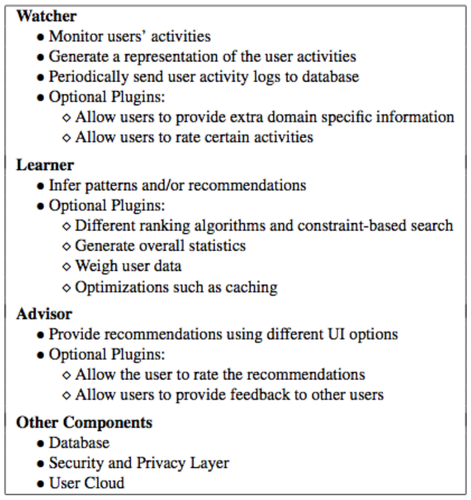
2. WEHELP REFERENCE ARCHITECTURE

A Reference Architecture is a collection of best practices for a certain domain. It is a design template and acts as a blueprint for building applications. Reference Architectures also define a common vocabulary with the goal of standardizing different independent implementations.

The weHelp Reference Architecture is shown in Figure 1. Each component has its own *task list*, which is a required set of tasks it must perform, and its own optional set of *plugins* for providing added functionality, shown in Figure 2.

The **Watcher** module is an observer of user activities. It "watches" what a user does with the tools or software in question and logs this information. These observations may be transparent to the user or, alternatively, may include allowing the user to provide extra domain specific information and/or to rate their experience.

The **Learner** module is responsible for inferring patterns



Watcher
- Monitor users' activities
- Generate a representation of the user activities
- Periodically send user activity logs to database
- Optional Plugins:
  - ◇ Allow users to provide extra domain specific information
  - ◇ Allow users to rate certain activities

Learner
- Infer patterns and/or recommendations
- Optional Plugins:
  - ◇ Different ranking algorithms and constraint-based search
  - ◇ Generate overall statistics
  - ◇ Weigh user data
  - ◇ Optimizations such as caching

Advisor
- Provide recommendations using different UI options
- Optional Plugins:
  - ◇ Allow the user to rate the recommendations
  - ◇ Allow users to provide feedback to other users

Other Components
- Database
- Security and Privacy Layer
- User Cloud

Figure 2: Task list for weHelp modules

and recommendations. Depending on the problem domain, the Learner module can either use a rule-based system, data aggregation, or complex data mining algorithms. The Learner module may weigh the user data in many different ways such as weighting recent data more than older data.

The **Advisor** module is responsible for providing the actual recommendations to the users. It can be implemented using a variety of different user interface options based on the problem domain. Furthermore, the recommendations can either be pushed to the user or pulled by the user. Users may also provide feedback, *e.g.*, through comments and ratings, to other users and to the recommender system. We differentiate this feedback received via the Advisor from the activities observed by the Watcher. The latter is restricted to observing how a user uses the system in question whereas the former is used for providing feedback to the users and the system about the efficacy of the recommendations.

3. ANALYSIS

Systems such as [7] have a similar architecture to weHelp. This validates our architecture as being a practical one as there are successful systems, other than ours, that have a similar architecture.

On the other hand, systems such as Rascal [4] are incompatible with the weHelp Reference Architecture. The Rascal architecture is not as well-modularized as the weHelp architecture as there is essentially one main component that does most of the recommendation functionality. Hence, it does not create the potential for the interoperability of components, which would have been possible if it had followed the weHelp Reference Architecture.

4. RELATED WORK

Fink and Kobsa in their work on "User Modeling Systems" [2] discuss some recommender systems but only focus on a generalized case study of these systems and compare each of their architectures. There has been other work [7] in which recommender system architectures are discussed. Most of the research has focused only on existing architectures and implementations without aiming to propose a generic reference architecture.

5. LIMITATIONS AND FUTURE WORK

Knowledge-based recommendation systems, such as [3], are similar to expert systems and use domain knowledge to provide recommendations. A limitation of the weHelp architecture is that systems that do not use any knowledge gained from usage data cannot be mapped to it.

Some of the future challenges involve expanding on and providing more concrete modules for what we call "Other Components" in Figure 2, specifically the Security and Privacy layer.

6. CONCLUSION

We have outlined a reference architecture for social recommender systems applicable to SE problems as well as other domains. It was derived from three ongoing projects in recommender systems for specific SE subtasks or analogous to SE task workflows. The component structure of our reference architecture provides a template for designing modularized recommender systems and could lead to standard interfaces and interoperability among recommender system components.

7. ACKNOWLEDGEMENTS

The authors are members of the Programming Systems Lab, funded in part by NSF CNS-0905246, CNS-0717544, CNS-0627473 and CNS-0426623, and NIH 1 U54 CA121852-01A1.

8. REFERENCES

[1] A. Califano, A. Floratos, M. Kustagi, and J. Watkinson. geWorkbench: An Open-Source Platform for Integrated Genomics. `http://www.geworkbench.org`.

[2] J. Fink and A. Kobsa. A review and analysis of commercial user modeling servers for personalization on the world wide web. *User Modeling and User-Adapted Interaction*, 10(2-3):209–249, 2000.

[3] R. Holmes, T. Ratchford, M. P. Robillard, and R. J. Walker. Automatically recommending triage decisions for pragmatic reuse tasks. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*, pages 397–408, 2009.

[4] F. McCarey, M. Cinnéide, and N. Kushmerick. Rascal: A recommender agent for agile reuse. *Artificial Intelligence Review*, 24(3):253–276, 2005.

[5] C. Murphy et al. genSpace: Exploring Social Networking Metaphors for Knowledge Sharing and Scientific Collaborative Work. In *1st Intl. Workshop on Social Software Engg. and Applications*, pages 29–36, September 2008.

[6] C. Murphy et al. Retina: Helping Students and Instructors Based on Observed Programming Activities. In *Proc. of the 40th ACM SIGCSE Techn. Symp. on CS Education*, pages 178–182, March 2009.

[7] P. Resnick et al. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proc. of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, 1994.

[8] S. Sethumadhavan et al. COMPASS: Community Driven Parallelization Advisor for Sequential Software. In *2nd Intl. Workshop on Multicore Software Engg.*, 2009.