weHelp: A Reference Architecture for Social Recommender Systems

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 of the 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 - systems where recommendations are derived automatically from the aggregate of logged activities conducted by the system's users. Our architecture is designed to be application and domain agnostic. We feel that a good reference architecture will make designing a recommendation system easier; in particular, weHelp aims to provide a practical design template to help developers design their own well-modularized systems.

1 Introduction

In the past few years, recommender systems have become increasingly popular and ubiquitous. Recommendation systems are being used in a variety of different domains such as suggesting movies we might enjoy watching [Neta], things we might want to buy [Ama], music we may like [Pan], and people we may know [Fac], often based on community-driven "people like you ..." paradigms. There has been keen public interest in improving such systems' recommendations, such as the Netflix Prize [Netb]. In the academic community, there has been much research on recommender systems, mostly focused on common issues of recommender systems such as recommendation algorithms [ZP07, GM08, PT08], implications of social networks in recommendation systems [ZC08, GDM+08] and security and privacy issues [BEKR07, MA07]. We have found relatively little research to date, however, in the area of system architectures for recommendation systems outside specific domains; work such as [GNOT92, RIS+94] has focused on their own architectures and implementations, without aiming to propose a general-purpose reference architecture, as we do here.

In this paper, we introduce *weHelp* - a reference architecture for social recommender systems. A reference architecture here is a collection of best practices and acts as a blueprint for building applications. Our architecture is designed to be independent of any specific application or domain. Our reference architecture is targeted towards community-driven recommender systems. weHelp systems are community-driven in the sense that the recommendations are derived automatically from the aggregate of logged activities conducted by the system's users and thus reflect community interests and/or behaviors. Commer-

cial examples of such recommender systems include Amazon [Ama] and Netflix [Neta], which use recorded user activities such as buying goods and renting movies, respectively, to recommend what might be a useful product to buy or an interesting movie to watch. We contrast social recommendation systems with what we call "knowledge driven recommendation systems", such as ACE [Mei88] or [MFS09], which are essentially rule-based systems and do not take into account the activities of the users. Such systems – which might be referred to collectively by a term like "iHelp" – do not learn by observing the activities of the users, but instead rely on a hard-coded set of rules and are, hence, relatively static. *weHelp* social recommender systems, in contrast, are dynamic as they learn (and generally over time refine) their recommendations by observing their users.

The main contribution of this paper is our weHelp reference architecture. A good reference architecture should make designing a new recommendation system simpler. weHelp is intended to provide a template to help developers design a well-modularized system.

2 Background and Motivation

The authors are collectively working on three otherwise unrelated research projects that share a similar theme: providing suggestions aimed to help users employ particular software tools in some better way. From that ongoing body of work we reverse engineered the common components and structure that led to weHelp. Our goal is to present a generic reference architecture for recommender systems that will provide a good design template and act as a blueprint for building recommender system applications. The following paragraphs give some background on the three systems prototyped as part of our projects: genSpace, Retina, and COMPASS. These three systems were designed independently, and were not intended *a priori* to exhibit a common architecture. But we then realized that the architectures for these systems are very similar, and we have distilled a common architecture and the best practices that we discovered along the way into our reference architecture.

The **genSpace** project [MSKW08] is being conducted in collaboration with Columbia University's Center for Computational Biology and Bioinformatics (C2B2). Researchers at C2B2 have developed geWorkbench [CFKW], which is an open-source Java-based system for integrated genomics targeted to biomedical researchers. As geWorkbench includes more than 50 plugin tools for genomics data analysis and visualization, it can be very daunting for a new user who does not know which tools to use with which data set, the order in which to use these tools, *etc*. A recommender system can be particularly beneficial for such users. At the same time, genSpace can also be very useful for experienced users, as it can provide insights about their peers' geWorkbench usage that they may not be aware of. The main goal of genSpace is to provide collaborative filtering and knowledge sharing features to geWorkbench users through recommendations presented via social networking metaphors such as "people like you . . . ". The genSpace module of geWorkbench records certain aspects of user activities and these are used to generate recommendations. Example recommendations include suggesting which analysis most people perform next given that they had started with the analysis most recently completed by this user, and listing the most commonly used workflows (series of analysis tools) and most popular tools.

We are separately investigating community-driven recommendation in the domain of computer science education. Our **Retina** system [MKLH09] is targeted towards CS1 (intro to programming) courses. Because students in these classes have little prior programming experience on which to draw, they may be unable to accurately estimate how long a programming assignment should take to complete, or how to address the often cryptic compiler and runtime error messages that they encounter. Thus, a recommendation system that is targeted to their individual needs could be very effective in helping them to be more efficient, and spend more time focused on algorithmic thinking and problem solving than on syntax and exceptions. Retina collects objective observational data about students' programming activities. Once data has been collected and aggregated, Retina makes real-time recommendations to students as they are working on their assignments. Example recommendations include warning the given student if she is encountering an especially high number of errors per compilation or if she is spending too long on the assignment. Retina also supports longer term "organizational memory", enabling it to make suggestions to students about what to expect from the assignment, for instance how much time to expect to spend on it or what errors to look out for. Last, Retina provides informative reports based on the aggregation of that data. These reports allow instructors to answer such critical questions as "how long are students taking to complete the programming assignments?", or "what sorts of errors are they making?" so that upcoming lectures can be revised accordingly.

Our third recommender system project, **COMPASS** (Community Driven Parallelization Advisor for Sequential Software) [SAG+09] proposes to leverage aspects of social networking to suggest multi-core (multi-threading) optimizations to programmers. COMPASS assumes a base of expert users who will parallelize existing sequential code using one or more of the numerous techniques for parallelization. The code changes are recorded, summarized, and stored it in a central database. When an inexperienced user wishes to parallelize her code, COMPASS will first identify the regions of code most warranting performance improvement, and then show a list of potential optimizations mined from previous parallelizations, *i.e.*, matching previously observed "before" serial code to retrieve the corresponding "after" parallel code - thus propagating community knowledge.

3 weHelp Reference Architecture

A Reference Architecture [HH00] 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. Reference Architectures usually define different modules and specify the functionality of each module along with the interaction and interfaces between the different modules. There have been many examples of reference architectures in varying domains such as scientific workflow management [LLF+09], and web servers [HH00]. In this paper, we define a reference architecture for social recommender systems.

The weHelp Reference Architecture is shown in Figure 1. It consists of three main components: the *Watcher*, the *Learner*, and the *Advisor*. All of these components have their

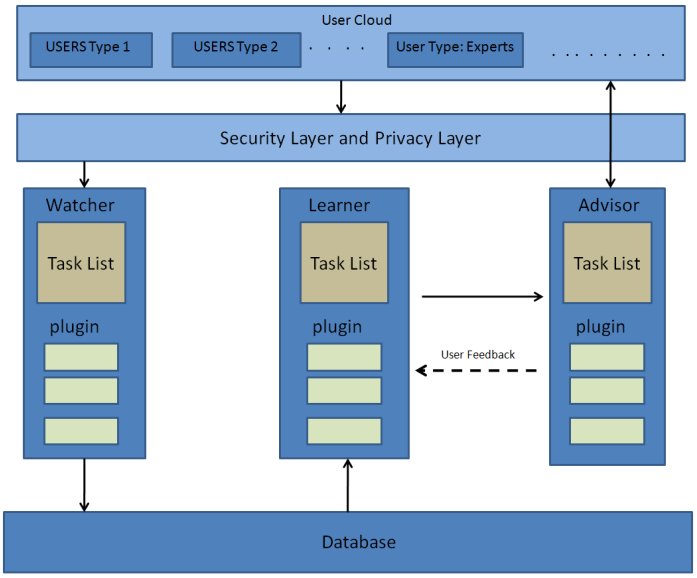

Figure 1: weHelp Reference Architecture

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

own *task list*, which is a required set of tasks they must perform, and their own optional set of *plugins* for providing added functionality, shown in Figure 2. These plugins are conceptual, and in a given implementation might be hard-wired into the design/code or indeed be supplied in a replaceable plugin form. The following subsections describe the various components of the weHelp reference architecture. We use Amazon [Ama] as a running example to present how its recommender system behavior maps to weHelp[1]. We discuss how the architectures of our three systems map closely to weHelp.

3.1 Watcher

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 explicit or implicit as far as the user is concerned. Explicit observations enable the user to annotate or tag the data and/or her activities. Other forms of explicit feedback include asking the user to fill out a form or answer a survey. Implicit observations, on the other hand, work in the background and monitor what the user is doing. Examples of implicit observations include counting the number of times a particular item was bought, or the number of times a link was clicked. Implicit observation is usually a better option, where possible, as it does not require any extra effort by the user and is also unobtrusive. The Watcher needs to generate an easily parseable representation of the user activities. Periodically, a log of users' activities needs to be stored in a database or web repository. Optional plugin functionality provided by the Watcher may include allowing the user to provide extra domain specific information and allowing users to rate certain activities.

In the case of Amazon, we can imagine that the Watcher module keeps track of user activities by observing the web pages visited. Consider an example where a user wishes to buy an MP3 player. The Watcher module can, implicitly, keep track of information such as the different kinds of MP3 players looked at. If the user decides to buy a particular MP3 player, the Watcher can also keep track of details such as the color, the capacity, and the vendor. The user can also provide explicit feedback to the Watcher module by rating the product he bought and writing a review for it. Amazon's Watcher module would conceivably log this information and it will later be used by the Learner module to infer recommendations for other users.

The following paragraphs describe how our three systems implement the Watcher module.

The Watcher module in genSpace transparently monitors the users' activities as they use geWorkbench. Whenever any data analysis is executed, different kinds of information are captured such as the name of the analysis, the type of the data set, and the current time. The Watcher module also allows users to choose how their data is logged: with their user names, anonymously, or not at all. Finally, in genSpace, all users are treated equally; there is no differentiation between "novice users" and "expert users". Thus, all the user logs are weighted equally when being considered as source data for the recommendations. This

[1]The authors have no affiliation with Amazon except as customers. The Amazon architecture has been inferred from the functionality provided by the website and may bear little or no relationship to the true architecture.

allows genSpace to work on the principle that if a particular tool is used in a certain way by a majority of the users, that is likely the "right" way to use that tool.

Similar to genSpace, the Watcher module in Retina transparently records students' compilation attempts and compiler errors. Additionally, any compilation errors are reported to the student as normal, but for each error, the type of error, the file name and line number, and the associated error message are all recorded as well. This information is accumulated completely transparently from the students' perspective without any manual intervention. Retina allows users to choose how their data is logged: with their user names, anonymously, or not at all. Finally, similar to genSpace, Retina treats all users equally and does not differentiate between "expert users" and "novices".

On the other hand, COMPASS will distinguish between "novice users" and "expert users". The Watcher module of COMPASS is meant specifically for experienced programmers, also known as "gurus". COMPASS assumes that a guru is an expert at writing optimizations for the corresponding input program. In contrast to Retina and genSpace, recording observations is not transparent to the user in COMPASS. The COMPASS IDE will allow the user to select portions of the input program (hot-spots) for parallelization. The expert user will then write a corresponding parallel program in the IDE. The Watcher module parses and stores the input program and the corresponding parallel solution. The Watcher module will have an additional plugin that allows users to give additional domain information such as target architecture and miscellaneous comments.

3.2 Learner

The Learner module is responsible for inferring patterns and/or recommendations. It will use the data that the Watcher module stores in the database. Depending on the problem domain, the Learner module can either use a simple rule-based system, data aggregation, or complex data mining algorithms. Optional plugins to the Learner module include the ability to have different ranking algorithms. The Learner module may also weigh the user data in many different ways such as weighing data from experts more than the data from novice users, or weighing recent data more than older data. Finally, the Learner module may also generate overall statistics about system usage and include optimizations such as caching the results and data to improve the response time to user queries. Note that the name "Learner" is intended to refer to *any* means for distilling knowledge from the data accumulated by the Watcher, not just those using machine learning algorithms.

Continuing our Amazon example, we can imagine that the Learner module uses the user information recorded by the Watcher module to infer patterns and recommendations. An example of these recommendations is suggesting alternate products to buy after considering other similar products viewed by the user. The Learner module also generates statistics such as the highest rated products and the most popular products in different categories.

The following paragraphs describe how our three systems implement the Learner module.

In genSpace, the Learner module uses data aggregation to provide the recommendations. For example, one of the suggestions provided by genSpace is 'what is the next tool to use'.

The Learner module uses the accumulated user data to find the most common workflows that are supersets of the user's current workflow. Looking at the most common workflows, it can suggest the best tool to use next. The genSpace Learner module uses an exponential time-decay formula [CS03] to weigh recent user data more heavily. This allows us to address the problem of concept drift [WK96], *i.e.*, workflows performed by users a long time ago may not be relevant today. The Learner module in genSpace also generates overall system statistics such as the most popular tools and most popular workflows.

In the case of Retina, although many of the recommendations are rule-based, the Learner module does perform some analysis of the data to determine the suggestions to make to individual students. For instance, one of Retina's suggestions is the amount of time that the student can expect to spend on the next programming assignment. This is done by considering the student's past performance on previous assignments with respect to the class average of time spent, and then finding the time that it took similarly-ranked students to complete the assignment in previous semesters. Another suggestion made by Retina involves the types of compiler errors that it feels the student is likely to make. This is achieved by noting any errors that the student has frequently made on previous assignments, especially those that fall outside the list of most common errors across all students in the class. The Learner module in Retina also has plugins for generating statistics that can be used in reports for the course instructor. The instructor can get an understanding of an individual student's efforts on a particular assignment, by seeing a list of all of the student's compilation and runtime errors, as well as aggregate data about the total number of compilation errors, the most common compilation error, and an approximation of how much time was spent on the assignment. For example, the instructor can select a single assignment and see an overview of how the class has performed as a whole, *e.g.*, the most common compilation and runtime errors. The instructor can also get a report of how much time each student has spent on the assignment, and the average time spent for all students in the class. This lets the instructor gauge the difficulty of a particular assignment.

In COMPASS, the Learner module will search for parallelizations performed by gurus so as to help inexperienced users. The workflow of the Learner module is as follows: the sequential code input by the user is first instrumented and its coverage data is analyzed to extract hot-spots (frequently executed parts of code). These hot-spots will now act as the input query to the database. The Learner module will use a code matching and ranking algorithm to extract various parallelizations for the user. The Learner module will also ask the user for some domain information, which it uses to get the best possible match.

3.3 Advisor

The Advisor module is responsible for providing recommendations to the users. These recommendations are given using the data inferred by the Learner module. The Advisor module can be implemented using a variety of different user interface options, depending on how exactly these recommendations should be provided, based on the problem domain. Furthermore, the recommendations can either be pushed to the user or pulled by the user. Pulled recommendations are very beneficial as users can ask for particular recommenda-

tions when they need them the most. Pushed recommendations can also be useful as they can be dynamic and take into account the user's current activities. Care needs to be taken, however, to avoid the "Clippy Effect" [Gal06], which would only serve to annoy the user. Optional plugins include allowing the user to rate the suggestions. Users may also provide feedback, *e.g.*, through comments, to the other users as well as to the recommender system. This form of feedback can be extremely useful as some recommendations might work better than others.

In the example of Amazon, the Advisor provides the recommendations to the user using HTML web pages as a user interface. *E.g.*, when a user is viewing a product, the Advisor module provides information such as other similar products viewed by users who also viewed this product, or other products purchased by users who also purchased this product.

The following paragraphs describe how our three systems implement the Advisor module.

In genSpace, the Advisor module is implemented as another component for geWorkbench. The user interface is Java Swing. The Advisor provides both pushed and pulled recommendations. Users can ask for recommendations such as finding workflows that include a certain tool or workflows that begin with a certain tool. Users can also view overall system usage statistics such as the top three most popular workflows, and the most popular tools. Users can also get pushed recommendations using "Real-Time Workflow Suggestions". As a user interacts with geWorkbench and runs different analyses, genSpace can provide real-time suggestions. Examples of the feedback provided include suggesting the most popular tool to use next and common superflows including the user's current workflow. Users can also rate the tools, write comments, and read other users' comments.

Similar to genSpace, the Advisor module in Retina supports two different models for providing advice and suggestions. Students can request suggestions by accessing a web application and getting reports about their own behavior, and what to expect from upcoming programming assignments. These suggestions include the amount of time that the student will likely spend on the assignment, and different errors to look out for. Additionally, Retina can produce immediate, real-time recommendations that are proactively sent to students based on their observed programming activities. These recommendations are sent to the students as they are programming and as their event logs are being captured. Retina uses Instant Messaging (IM) applications as the user interface for its recommendations.

The Advisor module in COMPASS will aid the user by providing suggestions in stylized templates know as 'sketches'. These sketches will be shown in the form of graphical overlays on the existing source code, which will be easily understood by the user. Users can accept the suggestions as is, or alter them according to their requirements. COMPASS will also allow users to give feedback to the system regarding the usefulness of the suggestion.

3.4 Other Components

The **Database** component is used as a data store for the user activities.

Depending on the problem domain, security and privacy issues can be critical as the rec-

ommendations provided by the system may be used to identify individual users or invade the privacy of users by inferring how a particular user uses the system. The **Security and Privacy Layer** can be used to provide the necessary functionality to ensure that the users' privacy is maintained and that security threats are mitigated.

Of course, a recommendation system is not useful without users in the so-called "user cloud". Broadly speaking, users can be separated into two types: those from whom the recommendation system learns, and those who benefit from the aggregated knowledge. Sometimes those user groups can overlap. For instance, in genSpace all users are considered the same: anyone can contribute to the store of knowledge, and anyone can benefit from it. On the other hand, sometimes these user roles can be explicit. For instance, in COMPASS, users can either be "gurus" (those whose parallelization activities have been monitored) or "novices" (those who are trying to parallelize code and need assistance).

4 Related Work

There have been many examples of reference architectures in varying domains such as scientific workflow management [LLF+09], and web servers [HH00].

Previous research into community-driven recommender systems has investigated the rationale behind such tools, *e.g.*, the psychology of collaborative technical help and help-giving [COT+06, TR04] and using organizational memory for collaborative help [AM96]. The KnoSoS project [CKVDM06] applies social networking concepts to knowledge sharing by investigating how to create group boundaries and track content. Carroll *et al.* have looked at extending the idea of knowledge sharing to concept sharing, and eventually activity sharing and activity awareness [CRCG06]. We build on these important works by considering the challenges of designing a reference architecture that supports such techniques, and hope that our work facilitates the building of such systems in the future.

Fink and Kobsa in their work on "User Modeling Systems" [FK00] 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 [GNOT92, RIS+94] in which recommender system architectures are discussed. Most of the research has focused only on the existing architectures and implementations without aiming to propose a generic reference architecture. To the best of our knowledge, there is no definitive work that describes a generic end-to-end architecture for such systems.

Our reference architecture can be viewed as a design pattern. The weHelp reference architecture does not map to any of the well-known design patterns, to the best of our knowledge. The design pattern that is the closest to weHelp is the MVC design pattern [Ree79]. The Model in MVC corresponds to the Database and the Learner components in weHelp. The Controller in MVC is similar to the Learner in weHelp. The View in MVC corresponds to the Watcher and the Advisor components in weHelp. We did not use the MVC design pattern as is, as the functionality and the division of responsibility need to be different for social recommender systems.

5 Limitations and Future Work

Knowledge-based recommendation systems use domain knowledge to gather inferences about the requirements of the user and to understand how a particular item meets a user's need. These systems are very similar to expert systems and may not utilize any aspects of social networking. Examples of such systems include ACE [Mei88] or [MFS09]. A limitation of the weHelp architecture is that systems that do not use any knowledge gained from usage data cannot be mapped to the weHelp reference architecture.

Some of the future challenges would involve expanding on and providing more concrete modules for the Security and Privacy layer.

6 Conclusion

We have described a reference architecture for social recommender systems. Our reference architecture, *weHelp*, is designed to be generic and domain agnostic. The component structure of our reference architecture will provide a template for designing modularized recommender systems and could lead to standard interfaces and interoperability among recommender system components.

7 Acknowledgements

The authors would like to thank Aris Floratos and Kiran Keshav for their guidance and assistance with genSpace. We would also like to thank Simha Sethumadhavan for his direction on the COMPASS project, and Sahar Hasan for her work on Retina. We also thank Lauren Wilcox for sharing her insights into recommendation systems. 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.

References

[AM96] M. Ackerman and D. McDonald. Answer Garden 2: merging organizational memory with collaborative help. In *Proc. of the 1996 ACM conference on computer supported cooperative work (CSCW)*, pages 97–105, 1996.

[Ama] Amazon.com. `http://www.amazon.com`.

[BEKR07] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *RecSys '07: Proc. of the 2007 ACM conference on Recommender systems*, pages 9–16, 2007.

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

[CKVDM06] T. Coenen, D. Kenis, C. Van Damme, and E. Matthys. Knowledge Sharing over Social Networking Systems: Architecture, Usage Patterns and Their Application. *LNCS 4277: On the Move to Meaningful Internet Systems 2006*, pages 189–198, 2006.

[COT+06] A. Crabtree, J. O'Neill, P. Tolmie, S. Castellani, T. Colombino, and A. Grasso. The practical indispensability of articulation work to immediate and remote help-giving. In *Proc. of the 20th anniversary conference on computer supported cooperative work (CSCW)*, pages 219–228, 2006.

[CRCG06] J. M. Carroll, M. B. Rosson, G. Convertino, and C. H. Ganoe. Awareness and teamwork in computer-supported collaborations. *Interacting with Computers*, 18(1):21–46, January 2006.

[CS03] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS)*, pages 223–233, 2003.

[Fac] Facebook. `http://www.facebook.com`.

[FK00] 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.

[Gal06] R. G. P. Galluccio. Humanizing CALL: The use of pedagogical agents as language tutors. New England Regional Association for Language Learning Technology, Oct. 2006.

[GDM+08] W. Geyer, C. Dugan, D. R. Millen, M. Muller, and J. Freyne. Recommending topics for self-descriptions in online user profiles. In *RecSys '08: Proc. of the 2008 ACM conference on Recommender systems*, pages 59–66, 2008.

[GM08] A. Gunawardana and C. Meek. Tied boltzmann machines for cold start recommendations. In *RecSys '08: Proc. of the 2008 ACM conference on Recommender systems*, pages 19–26, 2008.

[GNOT92] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[HH00] A.E. Hassan and R.C. Holt. A reference architecture for Web servers. *Proc. of Seventh Working Conference on Reverse Engineering*, pages 150–159, 2000.

[LLF+09] C. Lin, S. Lu, X. Fei, A. Chebotko, D. Pai, Z. Lai, F. Fotouhi, and J. Hua. A Reference Architecture for Scientific Workflow Management Systems and the VIEW SOA Solution. *IEEE Transaction on Services Computing*, 2(1):79–92, 2009.

[MA07] P. Massa and P. Avesani. Trust-aware recommender systems. In *RecSys '07: Proc. of the 2007 ACM conference on Recommender systems*, pages 17–24, 2007.

[Mei88] B. J. Meier. ACE: a color expert system for user interface design. In *UIST '88: Proc. of the 1st annual ACM SIGGRAPH symposium on User Interface Software*, pages 117–128, 1988.

[MFS09] M. Mandl, A. Felfernig, and M. Schubert. Consumer Decision Making in Knowledge-Based Recommendation. *IEEE International Conference on Active Media Technology and Brain Informatics, Peking, China, Springer Lecture Notes on Computer Science, LNCS 5820*, pages 69–80, 2009.

[MKLH09] C. Murphy, G. Kaiser, K. Loveland, and S. Hasan. Retina: Helping Students and Instructors Based on Observed Programming Activities. In *Proc. of the 40th ACM SIGCSE Technical Symposium on Computer Science Education*, pages 178–182, March 2009.

[MSKW08] C. Murphy, S. Sheth, G. Kaiser, and L. Wilcox. 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.

[Neta] Netflix. `http://www.netflix.com`.

[Netb] Netflix Prize. `http://www.netflixprize.com`.

[Pan] Pandora Radio. `http://www.pandora.com`.

[PT08] Y.-J. Park and A. Tuzhilin. The long tail of recommender systems and how to leverage it. In *RecSys '08: Proc. of the 2008 ACM Conference on Recommender systems*, pages 11–18, 2008.

[Ree79] T. M. H. Reenskaug. Models-Views-Controllers. *Xerox PARC technical note*, 1979.

[RIS+94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 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.

[SAG+09] S. Sethumadhavan, N. Arora, R. B. Ganpathi, J. Demme, and G. Kaiser. COMPASS: Community Driven Parallelization Advisor for Sequential Software. In *2nd Intl. Workshop on Multicore Software Engg.*, 2009.

[TR04] M. Twidale and K. Ruhleder. Where am I and who am I?: issues in collaborative technical help. In *Proc. of the 2004 ACM conference on computer supported cooperative work (CSCW)*, pages 378–387, 2004.

[WK96] G. Widmer and M. Kubat. Learning in the Presence of Concept Drift and Hidden Contexts. *Machine Learning*, 23(1):69–101, 1996.

[ZC08] V. Zanardi and L. Capra. Social ranking: uncovering relevant content using tag-based recommender systems. In *RecSys '08: Proc. of the 2008 ACM conference on Recommender systems*, pages 51–58, 2008.

[ZP07] J. Zhang and P. Pu. A recursive prediction algorithm for collaborative filtering recommender systems. In *RecSys '07: Proc. of the 2007 ACM conference on Recommender systems*, pages 57–64, 2007.