weHelp: A Proposed Reference Architecture for Community Driven 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 proposed reference architecture for 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. Our architecture is designed to be application and domain agnostic. We describe the various components of our reference architecture and also show how various existing recommender system architectures map closely to weHelp. The main contribution of this paper is to present a possible reference architecture for community driven recommender systems. 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. We hope that our reference architecture will act as a useful starting point for people interested in building their own recommendation systems. Finally, we seek participation by the recommender system development community in discussions towards a future improved reference architecture, to enhance community communication and, eventually, component interoperability.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architectures—*Domain-specific architectures*; K.4.0 [**Computing Milieux**]: Computers and Society—*General*

General Terms

Human Factors, Design

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
RecSys '09 New York City, NY, USA
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

Keywords

Recommender Systems, Reference Architecture, Knowledge Sharing

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 [28], things we might want to buy [3], music we may like [20, 30], and people we may know [12], 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 competition [29]. In the academic community, there has been much research on recommender systems, mostly focused on common issues of recommender systems such as recommendation algorithms [16, 31, 40], implications of social networks in recommendation systems [14, 39] and security and privacy issues [5, 22]. We have found relatively little research to date, however, in the area of system architectures for recommendation systems outside specific domains; work such as [15, 32] 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 proposed reference architecture for community driven 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. It consists of several components, each dealing with a specific task needed for most recommendation systems. weHelp's three main components are called the *Watcher*, the *Learner*, and the *Advisor*. The Watcher observes the users' activities. The Learner uses this data to infer patterns and/or recommendations. These recommendations are then provided to the user via the Advisor.

Our proposed reference architecture is targeted towards community driven recommender systems in which the recommendations are based on the activities of an effective community of users. Commercial examples of community driven recommender systems include Amazon [3] and Netflix [28], 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 community driven recommendation systems with what we call "knowledge driven recommendation systems", such as ACE [24] or WebMD [37], 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* community driven 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 proposed reference architecture for community driven recommender systems. 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. In the longer term, we hope to encourage the community towards participating in collectively devising a future improved reference architecture that may, eventually, lead to standard interfaces enabling interoperability and reuse among recommender system components.

Section 2 discusses our background and motivation along with related work. Section 3 presents our reference architecture and explains its proposed components. In Section 4, we discuss how various existing recommender systems map to the weHelp reference architecture along with some limitations. Finally, in Section 5, we call for community participation in further development of a reference architecture for recommender systems.

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.

2.1 Background

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 proposed reference architecture.

The **genSpace** project [27] is being conducted in collaboration with Columbia University's Center for Computational Biology and Bioinformatics (C2B2) and its Multiscale Analysis of Genomics and cellular Networks center (MAGNet). Researchers at C2B2 have developed geWorkbench [7], 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, such a system can be very useful for experienced users as well, as it can provide certain 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 sends these logs back to a central server periodically. At the server side, the data is aggregated, patterns and suggestions are derived, and these are used to generate recommendations for the users. Example recommendations to a given user include suggesting in real-time which analysis most people perform next given that they had started with the analysis most recently completed by this user, as well as listing on demand 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 [26] 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, specifically how long they have worked on an assignment, the compilation errors that were reported, and the runtime errors (uncaught exceptions) that they have encountered. Once data has been collected and aggregated, Retina makes real-time recommendations to students as they are working on their assignments. In particular, Retina warns the given student if she is encountering an especially high number of errors per compilation, if she is spending too long on the assignment, or if the same error has occurred a high number of times; all of these depend on thresholds set by the instructor. Retina can also send expert (as opposed to community driven) recommendations about how to fix certain programming errors, based on observing the student's code and predefined rules.

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 (assuming homework is assigned from semester to semester). 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 compilation and runtime errors are they making?" so that upcoming lectures can be revised accordingly.

In our third recommender system project, where the initial design effort is still in progress, we are considering the problem that parallel hardware is becoming a ubiquitous component in computer processing technology but exploiting the full potential of such platforms has foxed programmers for a very long time. **COMPASS** (Community Driven Parallelization Advisor for Sequential Software) [33] proposes to leverage aspects of social networking to suggest multi-core (multi-threading) optimizations to both professional and student programmers.

The COMPASS design assumes a base of expert users who will parallelize existing sequential code using one or more of the numerous techniques for parallelization. It records their code changes, summarizes this information, and stores 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 (via profiling), and then prioritize 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. The suggested optimizations will then be presented in a stylized template so as to make it easier to understand.

2.2 Related Work

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 [11, 36] and using organizational memory for collaborative help [2]. The KnoSoS project [9] seeks to apply 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 (what they term "knowledge in common") to concept sharing, and eventually activity sharing and activity awareness [8]. 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.

There has been other work [15, 32] in which recommender system architectures are discussed. Most of this research, however, focuses only on the authors' own architectures and implementations without aiming to propose a generic reference architecture. Through weHelp, on the other hand, we define a general-purpose reference architecture that can be used in myriad domains. We also hope that the component structure of our proposed reference architecture will provide a template for designing modularized recommender systems, and that community participation in collectively designing a future improved reference architecture will lead to standard interfaces and interoperability among recommender system components.

3. PRELIMINARY WEHELP REFERENCE ARCHITECTURE

A Reference Architecture [17] 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 [21], multimedia services [35], web servers [17], and peer-to-peer overlay networks [1]. In this paper, we define a reference architecture for community driven recommender systems.

The weHelp Reference Architecture is shown in Figure 1. It consists of three main components: the *Watcher*, the *Learner*, and the *Advisor*. The Watcher is responsible for monitoring user activities. The Learner is responsible for deriving patterns and recommendations after looking at the user activities provided by the Watcher. The Advisor is responsible for providing recommendations and feedback to

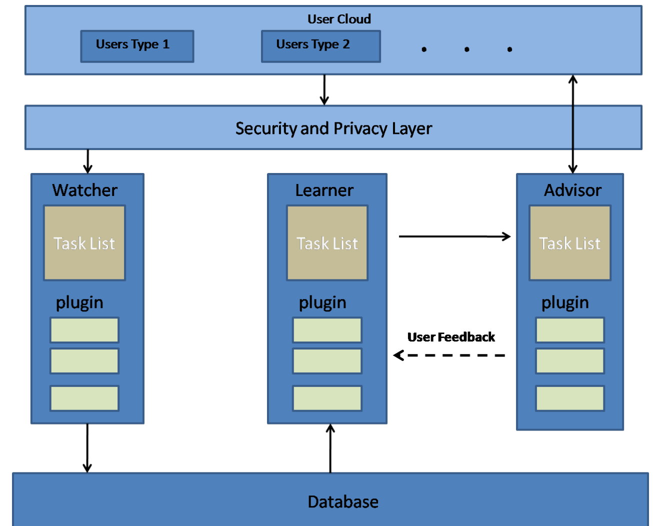


Figure 1: weHelp Reference Architecture

the users. All of these components have their own *task list*, which is a required set of tasks they must perform, and their own optional set of *plugins* for providing added functionality. 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. What we envision for a future improved reference architecture is a more sophisticated model for plugins that could lead to interoperability of plugins across different realizations of the architecture.

There are also two other components in the reference architecture: the database, and the security and privacy layer. Finally, there is the user cloud, which has the users of the system. We could distinguish between different types of users (*e.g.*, novices vs. experts) depending on the system domain.

The following subsections describe the various components of the weHelp reference architecture. We use Amazon [3] as a running example to present how its recommender system behavior maps to weHelp[1]. We also discuss how the architectures of our three systems - genSpace, Retina, and COMPASS - map closely to weHelp.

In Section 4, we describe how various other recommendation system architectures also 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, number of times a link was clicked, or number of times a query was given to a recommender system. Implicit observation is usually a better

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

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. geWorkbench has a core engine which provides basic functionality and allows developers to add new functionality via the use of plugins in a publish/subscribe framework. The Watcher module of genSpace is one such plugin and receives notifications from the core engine whenever any analysis is started.

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 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.

Just as genSpace transparently records the execution of analysis tools in geWorkbench, the Watcher module in Retina transparently records students' compilation attempts and compiler errors, either using a plugin for the Eclipse or BlueJ programming environment, or via a wrapper around the Java command-line compiler. When the student invokes the compiler, the student's name (or user ID) and the current date and time are recorded. 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.

To address privacy issues, the Watcher in Retina allows the student to opt out of the collection of data entirely. Alternatively, the student can specify that information be collected anonymously so that it contributes only to the overall data for the class, and cannot be tied back to the individual student.

Note that the Watcher module in Retina does not differentiate between "expert users" and "novices". In fact, *all* Retina users are considered to be novices; similar to genSpace, Retina is built on the notion that a majority of users represent what is common across all users and that recommendations should be based on general observations, without delineating who is an expert and who is not.

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 in an unique signature format. If later the Learner module encounters a similar input program, the signature is used for code matching.

The Watcher module will have an additional plugin that allows users to give domain information. This information pertains to knowledge of the target architecture for which the given parallel solution is suited. Furthermore, information regarding files that may be required to run the program and miscellaneous comments may also be added by the user.

3.2 Learner

In the previous subsection, we discussed the Watcher module and how it is used for monitoring the users' activities. We also discussed how our three systems implement the Watcher module. In this subsection, we describe the Learner module.

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. Also, depending on the type of recommender system, the Learner module can perform different activities. For example, in the case of collaborative filtering approaches [18], the Learner can have different rank based search algorithms that could be based on different features rated by the user. In the case of content-based approaches [4], the Learner can form personalized user profile models (using the feedback provided by the users via the Advisor) and use, *e.g.*, Bayesian or Decision-Tree approaches to infer rules.

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. Examples of this would be weighing data from experts more than the data from novice users, or weighing recent data more than older data. The Learner module may also generate overall statistics about system usage. Finally, the Learner module may include optimizations such as caching the results and data to improve the response time to user queries.

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 overall 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 [10] to weigh the data; user data that is recent is weighted more heavily. This allows us to address the problem of concept drift [38], *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. The instructor can also 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. It takes the sequential program given by the user as the input query and searches for a corresponding parallelization in the database. 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 are then parsed to generate a unique representation, which will act as the input query to the database. The Learner module will then use a code matching and ranking algorithm to extract various parallelizations for the user.

The Learner module also asks the user to give system information (*e.g.*, architectural information). The Learner can then use this system information to get the best possible match for the current user.

3.3 Advisor

In the previous subsection, we described the Learner module and how it can infer patterns and/or recommendations. We gave examples of how genSpace, Retina, and COMPASS implement the Learner module. In this subsection, we discuss how these patterns and recommendations are provided to the user via the Advisor module.

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 recommendations 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" [13], which would only serve to annoy the user.

Optional plugins may 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 inferred by the Learner module to the user. The user interface is through HTML web pages. For example, when a user is viewing the details of a particular product, the Advisor module provides information such as other similar products viewed by users who also viewed the current 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 common workflows that include a particular 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 and workflows, write comments, and read other users' comments.

Similar to genSpace, the Advisor module in Retina supports two different models for providing advice and suggestions based on what has been observed about the particular student, her classmates, and students who took the class in previous semesters. 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 by the Watcher module. Retina uses Instant Messaging (IM) applications as the user interface for its recommendations. Using the JClaim [19] API, these recommendations are sent via an IM server bot that works with the Yahoo! Messenger, Windows Live Messenger, and Google Talk chat networks. A student initiates a chat session with Retina, and identifies herself via a username. When Retina determines that a recommendation is in order, a message is sent to the student so that she can get a better understanding of what to do next.

The current implementation of the Advisor module in Retina is rule-based: when event logs are received, Retina checks whether any of the conditions (set by the instructor) have been met, and sends a message accordingly. These include spending too long on an assignment or making too many compiler errors. However, the module could easily be modified so as to take into account other students' activities, so that the real-time recommendations are triggered by deviations from what is observed to be "normal" student behavior.

With COMPASS, on the other hand, we realize that simply providing relevant advice to inexperienced users is not enough; this advice also needs to be coherent to the user. This is more of a problem with COMPASS than with genSpace or Retina. With minimal previous experience, it may be a tough task for a programmer to understand how to map the recommendations to her current problem. The proposed Advisor 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 suggestion as is, or alter them according to their requirements. As future work, COMPASS will allow the user to give feedback to the system regarding the usefulness of the suggestion.

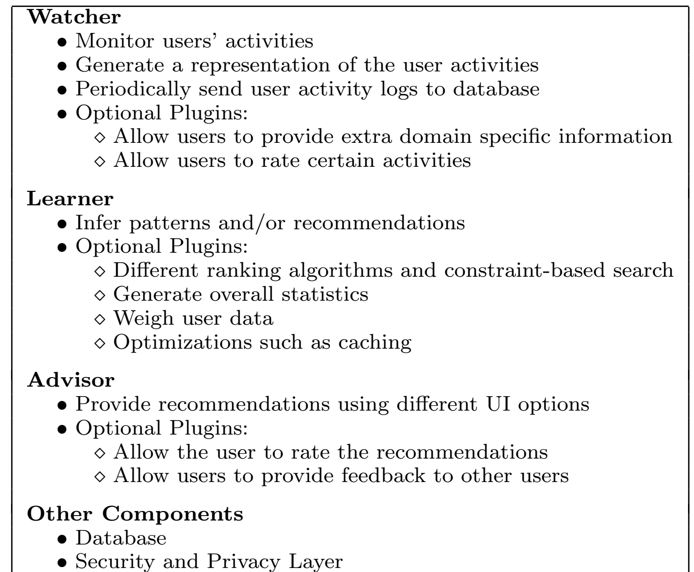
3.4 Discussion

Figure 2 presents a summary of the different modules of our reference architecture and the functionality of each module. In addition to the three major modules described above, the weHelp reference architecture also contains two other components described here.

The **Database** component is used as a data store for the user activities. The Watcher module sends the logs of user activities to the database. The Learner module uses this database to infer the recommendations, which are then sent to the user via the Advisor module.

Depending on the problem domain, security and privacy issues can be critical as the recommendations 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.



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

Figure 2: Summary of weHelp modules

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).

Retina presents an interesting case of user roles. In Retina, there are instructors and students; however, unlike in the case of COMPASS, Retina does not attempt to learn from the instructors. Rather, Retina learns from the students and then makes recommendations to other students. But the instructor can receive recommendations, too. Because Retina observes what the students have done, it can make recommendations to the instructor, for instance which topics to spend more time discussing, or whether the assignment is too difficult.

4. ANALYSIS

In this section, we discuss various types of recommender systems and see how their architectures map closely to the weHelp reference architecture.

4.1 Published Architectures

There has been a variety of research on recommender system architectures [23]. However, to the best of our knowledge there is no definitive work that describes a generic end-to-end architecture for recommender systems. This section briefly describes a few systems classified by different approaches for developing recommender systems, and compares them to our reference architecture.

The most common and well-known approach for implementing recommender systems is "collaborative filtering". Collaborative filtering [18] summarizes preferences of users in order to make suggestions to other users User preferences can be either explicit, wherein the user explicitly states

whether or not she likes a given item; or they can be implicit, whereby the usage data is analyzed to infer the popularity of the item based on statistics, such as the number of queries, number of sales, or links associated with the given item.

Initial work in collaborative filtering systems includes systems such as Tapestry [15] (filtering e-mails), GroupLens [32] (filtering news articles) and ringo [34] (recommending music CDs). The GroupLens system is used to suggest relevant news articles to users. In the Watcher module of the GroupLens system, we could observe users rating the articles. The Learner predicts how much the user will like an article based upon different scores (ratings) provided by previous users. The Advisor translates the predictions made by the learner into letter grades to give a prediction of how much the user would be interested in the article.

Another approach commonly used in recommender systems is a content-based approach [4], whereby systems recommend an item to a user based on its attributes and the user's profile. User profiles are derived from learning models such as decision trees or Naive Bayes and are used to support long-term models based on features (attributes) of objects rated by users. This approach has the advantage of being able to recommend previously unrated items to users. Take, for example, the book recommender by Mooney *et al.* [25]. The Watcher module in their system can be used to observe explicit ratings done by the user. The Learner module, on the other hand, could implement and maintain models based on user profiles. In addition to providing the suggestions to the user, the Advisor module is also used to gather feedback and assist in the learning phase of the Learner module.

4.2 Inferred architectures

We had used Amazon as a running example in section 3 and discussed how its architecture maps to weHelp.

Facebook [12] [2] is a well known social networking website, which uses social networking patterns to suggest friends, and groups that the user might be interested in. The Watcher module in Facebook can be imagined to keep a log of the users' friends and groups. The Learner module can then use a variety of aggregation and mining algorithms to infer the people that a user may know or groups that she may be interested in. The Advisor, through the HTML interface, shows these recommendations to the user.

4.3 Limitations

One of the approaches towards developing recommender systems is a knowledge-based approach. These systems are very similar to expert systems and may not utilize any aspects of social networking. 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. The Entree [6] restaurant recommender system uses a rule-based system to give a range of suggestions to its users. A limitation of the weHelp architecture is that systems that depend purely on domain knowledge and do not use any knowledge gained from usage data cannot be mapped to the weHelp reference architecture. In knowledge based recommender systems there will be no Watcher module that keeps track of user activities, the rule-based logic could be a part of the Learner, and the Advisor displays the suggestions to the user.

[2] The authors have no affiliation with Facebook and have inferred the architecture from usage.

5. CONCLUSION

We have described a reference architecture for community driven recommender systems. Our reference architecture, *weHelp*, is designed to be generic and domain agnostic. We have described the various components of our architecture and their functionality. We have also discussed how a variety of recommender systems' architectures map closely to weHelp. Some of the future challenges would involve expanding on and providing more concrete modules for the Security and Privacy layer.

We are seeking community involvement towards a future improved reference architecture, to further community research and development. We hope that the community can help in designing standard interfaces and developing components that adhere to these interfaces to enable interoperability among recommender system components. We have set up a mailing list for this purpose, `wehelp@lists.cs.columbia.edu`, and we invite everyone to join this mailing list and participate in the design.

6. 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-0717544, CNS-0627473 and CNS-0426623, and NIH 1 U54 CA121852-01A1.

7. REFERENCES

[1] K. Aberer, L. O. Alima, A. Ghodsi, S. Girdzijauskas, S. Haridi, and M. Hauswirth. The essence of p2p: A reference architecture for overlay networks. *IEEE International Conference on Peer-to-Peer Computing*, 0:11–20, 2005.

[2] 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.

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

[4] M. Balabanovic and Y. Shoham. Combining content-based and collaborative recommendation. *Communications of the ACM*, 40:66–72, 1997.

[5] 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.

[6] R. D. Burke, K. J. Hammond, and B. C. Young. The findme approach to assisted browsing. *IEEE Expert*, 12:32–40, 1997.

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

[8] 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.

[9] 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.
[10] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *Proc. of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems (PODS)*, pages 223–233, 2003.
[11] 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 2006 20th anniversary conference on computer supported cooperative work (CSCW)*, pages 219–228, 2006.
[12] Facebook. `http://www.facebook.com`.
[13] R. G. P. Galluccio. Humanizing CALL: The use of pedagogical agents as language tutors. New England Regional Association for Language Learning Technology, Oct. 2006.
[14] 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.
[15] 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.
[16] 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.
[17] A. Hassan and R. Holt. A reference architecture for web servers. *Proc. of Seventh Working Conference on Reverse Engineering*, pages 150–159, 2000.
[18] J. L. Herlocker, J. A. Konstan, L. G. Terveen, John, and T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53, 2004.
[19] Java Compliant Logging and Auditing Instant Messenger. `http://www.itbsllc.com/jclaim/`.
[20] Last.fm. `http://www.last.fm`.
[21] 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 Transactions on Services Computing*, 2(1):79–92, 2009.
[22] 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.
[23] D. W. Mcdonald and M. S. Ackerman. Expertise recommender: a flexible recommendation system and architecture. In *Proc. of the 2000 ACM conference on Computer supported cooperative work*, pages 231–240, 2000.
[24] 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.
[25] R. J. Mooney and L. Roy. Content-based book recommending using learning for text categorization. In *Proc. of the Fifth ACM Conference on Digital Libraries*, pages 195–204, 2000.
[26] 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.
[27] C. Murphy, S. Sheth, G. Kaiser, and L. Wilcox. genSpace: Exploring Social Networking Metaphors for Knowledge Sharing and Scientific Collaborative Work. In *1st International Workshop on Social Software Engineering and Applications*, pages 29–36, September 2008.
[28] Netflix. `http://www.netflix.com`.
[29] Netflix Prize. `http://www.netflixprize.com`.
[30] Pandora Radio. `http://www.pandora.com`.
[31] 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.
[32] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. 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.
[33] S. Sethumadhavan, N. Arora, R. B. Ganpathi, J. Demme, and G. E. Kaiser. COMPASS: Community Driven Parallelization Advisor for Sequential Software. In *Second International Workshop on Multicore Software Engineering*, 2009.
[34] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proc. of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 210–217, 1995.
[35] T. Sutinen and T. Frantti. Reference architecture for mobility enhanced multimedia services. In *MOBILWARE '08: Proc. of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–6, 2007.
[36] 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.
[37] WebMD Symptom Checker. `http://symptoms.webmd.com`.
[38] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.
[39] 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.
[40] 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.