User-Centric Research Challenges in Community Information Management Systems

AnHai Doan[1], Philip Bohannon[2], Raghu Ramakrishnan[2]
Xiaoyong Chai[1], Pedro DeRose[1], Byron J. Gao[1], Warren Shen[1]
[1] University of Wisconsin-Madison, [2] Yahoo! Research

Abstract

In Cimple, a joint project between Wisconsin and Yahoo! Research, we are building systems that manage information for online communities. In this paper we discuss the fundamental roles users play in such systems, then the difficult user-centric research challenges raised by these roles, with respect to contributing to the system, accessing and using it, and leveraging the social interaction of users.

1 Introduction

In numerous online communities (e.g., those of database researchers, movie fans, and biologists) members often want to discover, query, and monitor relevant community information. *Community information management systems* (or *CIM systems* for short) aim to address such information needs [13]. First-generation CIM systems fall roughly into two classes: message boards and structured portals. In message-board systems (e.g., Usenet, Yahoo! Groups, DBworld), users exchange messages on active topics and the history of these messages provides a searchable repository of community knowledge. In contrast, portal systems include most enthusiast Web sites (e.g., *shakespeare-online.com*) and provide structured contents. While some portals (e.g. Citeseer [16]) have successfully presented automatically crawled content to users, most portal sites are maintained by a few system builders.

In Cimple, a joint project between Wisconsin and Yahoo! Research, we are developing techniques to build next-generation CIM systems [13]. Our first goal is to support *collaborative contribution and management* of a wide range of content (e.g., text, structured data, images). Our second goal is to minimize the information gathering load on community members by integrating *crawled Web content*. For example, in the DBLife prototype (see [12] and http://dblife.cs.wisc.edu), built as a part of the Cimple project, information of use to the database research community is crawled on a nightly basis. The challenge then is to integrate this data with the community-contributed text and structured data, while keeping quality high.

Several current projects are similar to Cimple in spirit, or share many of the goals. Examples include Impliance, MAFIA, and Avatar projects at IBM Almaden [8, 15, 23], BlogScope at the University of Toronto [7], BlogoCenter at UCLA [1], Dataspaces and PayGo at Google [19, 27], SHARQ and ORCHESTRA at the University of Pennsylvania [32, 9], Libra at MSR-Asia [28], related efforts at the University of Washington, MSR-Redmond [11, 17], Siemens Research [33], and many others (e.g., [6, 25], see also [14]). A key commonality underlying many of these projects is the *active and diverse roles* users play in building and using the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

systems. Consequently, we believe that for these emerging "Web 2.0" projects, it is important to discuss which fundamental roles users play, and what user-centric challenges these roles entail.

In this paper we contribute to this broader discussion, drawing from our initial experience in Cimple. We begin by observing that CIM users often play three fundamental roles: *active contributors*, *information explorers*, and *social players*. First, CIM users often act as active contributors, editing and supplying the system with data, code, and domain knowledge. Second, CIM users often have ill-defined information needs (e.g., find interesting relationships between X and Y), or have precise information needs but do not know how to express them in structured query formats, or are too "lazy" to express them. Consequently, they often behave as information explorers. Finally, CIM users operate in a social context, in that they often interact with other users in the same community and that the CIM data captures many of such interactions.

We then discuss the user-centric challenges raised by the above observations. We consider in particular three key challenges: (1) how to make it easy for users to contribute data, code, and knowledge to the system, (2) how can users easily access and query the system, and move seamlessly from one query mode to another, and (3) how to motivate users to interact more, then capture and exploit such interactions. Finally, we discuss *reputation management*, *explanation*, and *undo*, capabilities that we believe are critical to address the above challenges.

2 The Fundamental Roles of CIM Users

We now briefly describe CIM systems, then the roles their users play. To build a CIM system, such as the one for the database research community, a builder (who is a community expert) deploys an initial generic system and supplies it with a set of relevant data sources (e.g., researcher homepages, DBworld mailing list, conference pages, etc.). The system then proceeds in three main steps [13]:

- **Crawl, extract, and integrate the data:** The system crawls the sources at regular intervals to obtain data pages, then extracts mentions of relevant entities from the pages. Example mentions include people names (e.g., "Jim Gray" and "J. N. Gray"), conference names, and paper titles. Next, it integrates these mentions into entities, and discovers relationships among them (e.g., "Jim Gray gave a talk at SIGMOD-04"), thus transforming the raw data into an entity-relationship (ER) data graph.

- **Provide user services over the data:** Next, the system provides a variety of user services over the ER data graph. For example, the system may create for each user entity X a *superhomepage* which contains *all* information about X that the system finds from the raw community data. Other example services include browsing, keyword and structured querying, and monitoring.

- **Mass collaboration:** Finally, the system solicits and leverages the feedback of community users to further improve and evolve. For example, the system may publish each user superhomepage (as described earlier) in wiki format, then allow users to correct and add information. A user may also suggest a new data source for the system to crawl. As yet another example, if the system infers both X and Y to be PC chairs of SIGMOD-04, a user may flag these inferences as incorrect, and supply the domain constraint that each SIGMOD conference has just one PC chair.

The Cimple project [13] (see also http://www.cs.wisc.edu/~anhai/projects/cimple) describes CIMS in more details. Within this project, to validate and drive CIMS research, we have also been building DBLife, a prototype system that manages the data of the database research community (see [12] and http://dblife.cs.wisc.edu). For CIM systems, we observe the following user roles.

Active Contributors: CIM users often want to contribute data, code, and knowledge to the system. In DBLife, for example, users sent us URLs of new data sources, voted on whether a picture claimed to represent a person

X is truly X, and inquired about supplying new codes for keyword search, mention disambiguation, among others.

User willingness to contribute of course has been observed in numerous Web 2.0 efforts. The amount of contribution has also been observed to follow a Zipfian distribution: a relatively small percentage of users contribute very actively, followed by a long tail of users who contribute little or nothing (e.g., see [5]). Our initial experience suggests that this will also hold for CIMS. Consequently, we roughly divide human participants of a CIM system into three categories: (a) *builders:* a small, perhaps 1-3 person, team which deploys and maintains the hardware and software (analogous to the DBA of an RDBMS), (b) *editors:* a core of perhaps 10-20 highly motivated persons who actively contribute to the system, and (c) *users:* the rest of the community. When there is no ambiguity, we use "users" to refer to all three categories.

While users are willing to contribute in many Web 2.0 efforts, as noted above, *in CIM contexts it is particularly important that they do so*. This is because, by nature, CIM data comes from multiple heterogeneous sources. They are often incomplete, only partially correct, and semantically ambiguous. Hence, it is vital that users contribute so that the data can be gradually cleaned, disambiguated, and augmented, especially in cases where it is very difficult for systems, but relatively easy for human users to make a decision. For example, it is very difficult for DBLife to decide that a picture of X is indeed X, whereas it would be easy for users who know X. As another example, a user can quickly tell the system that "Alon Halevy" and "Alon Levy" are the same person, saving it much effort in attempting to determine so. Note that this is in sharp contrast to RDBMS settings, where the data often has a closed-world well-defined semantics. Many data management settings outside RDBMS however have semantic problems (e.g., CIM, but also schema matching, data integration, data cleaning, dataspaces, and model management), and thus can significantly benefit from user participations.

Information Explorers: Recent work has addressed the needs of users who approach structured data sources with vague queries, by supporting keyword queries over structured data (e.g., [4, 21, 20, 18, 31]). Similarly, CIM users often have diverse, ill-defined information needs. Many times a CIM user does not yet know exactly what he or she wants (e.g., knowing only that he or she wants to find something interesting on topic X). Hence, the user will start with keyword search and browsing, in an exploratory fashion. This is especially true in scientific data management. Eventually the user may "zoom in" on a precise information need (e.g., find all papers on topic X that Y and Z wrote in 2004), at which point he or she may want to switch to a structured query interface. So a major problem is how to ensure a smooth transition across heterogeneous query and browsing interfaces, with minimal user effort.

Even if a CIM user starts with a precise information need, he or she often is too "lazy" to compose a structured (e.g., SQL) query, or simply does not know how to do it. In DBLife, for example, few users appear to be willing to take the effort to compose a structured query, or know how to compose a *syntactically correct* one. This is an acute problem, because it severely limits the utility of all the structured data that DBLife has extracted and integrated. Consequently, finding a way to allow lay or "lazy" users to ask structured queries in CIM contexts is very important, if we want to maximize the full utility of structured CIM data.

Social Players: CIM users operate within a community. They are often aware of and interact with other users, and such interactions are often captured in the data managed by a CIM system. Exploiting such data on social interaction can often significantly improve the quality of CIMS. For example, in DBLife, interaction in form of citations, paper review, tagging, etc. can help identify topic experts, and help improve ranking the results of keyword searches. Hence, a key challenge is how to encourage such social interactions, and how to capture and exploit them.

Finally, as we have alluded to several times, CIM users often vary significantly in their degree of motivation and technical expertise. While we expect that a relatively small core of users (e.g., the builders and editors, as described earlier) are highly motivated and technically literate, the vast majority of users will just want to use the

system quickly if the need arises, then "move on with their lives". This exacerbates the user-centric challenges facing CIM systems, as we discuss next.

3 User-Centric Research Challenges

We now discuss the user-centric challenges, focusing in particular on user contribution, user services, and social interaction. Then we touch on reputation management, explanation, and undo, capabilities that are central to address the above challenges.

3.1 Effective User Contribution

Since user contribution is important for CIM, but the vast majority of users are reluctant to contribute, we must make it very easy for users to provide or modify system components. We focus on three main components: data, code, and domain knowledge.

Data: A user should be able to supply or edit any kind of data, using whichever user interface that he or she finds most convenient. The system then processes the data to its best ability. Example data include URL for a new data source, raw data pages (e.g., a page listing accepted SIGMOD papers), structured data, natural text, and tags, among others. Example user interfaces include form, GUI and wiki. Our Cimple experience suggests that wiki pages can provide a good baseline user interface, in that anything can be posted in wiki pages and can be easily edited. For instance, if DBLife displays user superhomepages in wiki format, then it is relatively easy for a user to correct and add information (especially natural text). Other interfaces can excel in certain cases. For example, a form interface is especially well suited for tagging data pieces with small text fragments.

In the above context, a major challenge is to translate user actions in an interface into actions over the underlying data. For example, conceptually a DBLife superhomepage describes a portion of the underlying ER data graph. Now suppose a user has revised a superhomepage (in wiki format). Then we must infer from the revised wiki page the exact sequence of actions the user intended to do over the ER data graph (e.g., remove a node, rename an edge, etc.). This inference is non-trivial because user edits often are ambiguous: the same edit can be mapped into multiple possible sequences of actions over the underlying data. Another challenge is that users often want to enter the data *together with some context information*. For example, when a user enters a page that contains a list of names, he or she may also want to say that these are the names of persons who are on the PC of SIGMOD-04.

Code: In practice, the code of a CIM system must often be tweaked to fine-tune the system performance. Today such tweaking is typically done by a small team of developers, incorporating suggestions from the members at large, in a slow and tedious process. This process can be improved markedly if we can open up certain parts of the code base for the multitude of members to edit.

To illustrate, consider extracting person names from the raw data pages. A common method is to start with a dictionary of names (e.g., "David Smith", "Michael Jones", etc.), perturb each name to generate variations (e.g., "D. Smith", "Smith, D"), then find occurrences of the variations in the raw pages. The method perturbs each name using the *same set of generic perturbation rules*. This often turns out to be insufficient. We found that when deployed in DBLife the method often had to be tweaked. It missed for example cases where a person X has an unusual nickname Y. Whenever this was pointed out to us by X or someone who knows X, someone on our development team would have to tweak the code, to add the nickname Y for X.

Clearly, allowing users to edit the code in such cases can drastically reduce the workload of the development team. Toward this goal, first we must make it very easy for users to edit the code. But it is unlikely that we can allow any user to edit code *directly*, as this can quickly result in corrupted code. A possible initial solution then

is to (a) decompose the code into a sequence of tasks, (b) materialize the *output* of each task, then (c) allow users to edit only these outputs. For example, the name extractor described above can be decomposed into a sequence of two tasks: generating variations for each name, then finding occurrences of the variations. Thus, the name extractor should *materialize* the set of variations it generates for each name, and expose these materialized sets to the users, so that they can edit (e.g., add the nickname Y to the set for X). In general, we can identify certain "edit points" in the code, make sure that the code "materializes" these edit points, then expose them (e.g., via a wiki interface) to allow users to edit.

Another possible solution (to make it easy to edit code indirectly) is to define *multiple choices* at certain points in the code. The default code always takes the default choices. But users can select other choices, thereby changing the execution flow of the code. For example, consider a module that matches person names, e.g., deciding if "D. Smith" and "David Smith" refer to the same person. This module may use the default choice of always applying the *same* matching method m to *all* superhomepages. But it should also offer several other matching methods, and allow users to choose a particular matching method for a particular superhomepage, if the user so desires. Thus, while examining a superhomepage H, a user may decide to examine the code that matches names within H, then decide that a matching method m' (offered in the code) is actually more accurate for H. Consequently, the user tells the system (perhaps via a radio-button interface) that, whenever matching names within H, it should use the matching method m' instead of the default method m.

This last example illustrates the power of collaborative code editing in CIM settings. In such settings, the small team that writes the initial code simply cannot examine *all* superhomepages to write appropriate code for each superhomepage. But they can write the code in a way that makes it easy later for community users to adapt the code to the peculiarities of each superhomepage.

To address malicious code editing, an initial solution is to limit code editing to only "trusted" users (e.g., editors). Even in this case, distributed code editing is already very useful, as it spreads the workload over multiple people. It is also very important to develop an undo capability, so that undesired changes to the code can be undone easily. We discuss this capability in more details in Section 3.4.

Domain Knowledge: When a CIM user finds something incorrect, he or she often knows some domain knowledge that can be used to flag it as incorrect or to fix it. For example, when a user sees that the system claims both A and B chair SIGMOD-04, he or she may be able to supply the knowledge that "only one person chairs a SIGMOD conference". We found such cases commonly occur in DBLife. Thus, just as domain knowledge (e.g., integrity constraints) plays an important role in RDBMS, it also plays an important role in CIMS. Consequently, it is important to find ways to allow users to express a broad variety of domain knowledge. The key challenge is to make it very easy for lay users to do this.

A possible solution is to cast each piece of domain knowledge as a constraint $Q\ op\ v$, where Q is a query template formulated in a structured language (e.g, SQL), op refers to a predefined operator (e.g., =, $<$, etc.), and v is a value. The user then interacts with the system to construct Q, then select op and v. For example, to express the constraint "only one person chairs a SIGMOD conference", the user constructs a template Q that finds the number of chairs of any given SIGMOD conference, then sets op to be $=$, and v to be 1. Another solution is for the system to solicit domain knowledge from the user. For example, while constructing a profile of a typical database researcher, a system may infer a constraint such as "no database researcher has published four or more SIGMOD papers in a year". It can then ask users to verify this constraint with answer "yes" or "no".

3.2 Effective User Services

As discussed earlier, CIM users often have ill-defined information needs, or do not know how to formulate the need in a structured query, or are too "lazy" to do so. Within this context, we must make it very easy for users to access and utilize the system. We now discuss the challenges in doing so, focusing on querying, context-sensitive services, and system access.

Querying: A user should be able to query the system using whichever query mode he or she finds most convenient, and should be able to switch seamlessly among them, with minimal effort. Example query interfaces include keyword search, GUI search, and structured querying. How to query effectively in each of these modes remains a major challenge. For example, while much work has addressed "plain-vanilla" keyword search (which returns a ranked list of data pages), no satisfactory solution exists today that can be adapted to work effectively, with minimal tuning, in a CIM domain. Similarly, much work has addressed keyword search over structured data, but no consensus has emerged on the most effective solution. Furthermore, how to execute structured queries over extracted structured data has received relatively little attention (with some exceptions [11, 22]). This last problem is difficult because the extracted structured data is often incomplete and imprecise.

Another major challenge is how to make smooth transition from one query mode to another. To move from a less structured query mode to a more structured one, a common solution is to interact with the user to refine the query [23, 26]. In the Avatar project [23], for example, when a user asks a keyword query "tom phone" over a corpus of emails, the system returns a ranked list of emails that contain these words. But it also provides an opportunity for the user to move to more structured querying, by asking if the user means to find emails that contain the phone number of Tom, or to find emails that come from Tom and contain the word "phone". There are often numerous possible structured-query interpretations for a keyword query. Hence a key difficulty facing this solution is how to select only the most likely interpretations, to show the user. User modeling (e.g., [3]) may help facilitate this selection. To move from a more structured query mode to a less structured one (e.g., when the more structured query does not produce any result and hence must be "relaxed", or when it cannot be executed over a text corpus), a common solution is to "collapse" the structured query, for example, into a set of keywords [30, 24]. The key issue is then how to select a good set of keywords.

Yet another major challenge is that once a CIM system has compiled a structured database, how can it enable users to easily pose structured queries over the database? For example, a user may want to know the average number of citations per paper for a particular researcher X. Clearly the system cannot expect that most users will be able to write a structured query (e.g., in SQL) expressing this information need. A possible solution is then for the system to interact with the user in a GUI fashion to construct a structured query.

Another possible solution is to generate form interfaces that capture the types of structured queries that we expect users will commonly ask. This is also the preferred approach for today RDBMS applications (e.g., $amazon.com$ provides a small set of form interfaces for users to query about books). CIM users however often have ill-defined and exploratory information needs (as discussed in Section 2). Consequently they often want to ask a far wider and more unpredictable range of structured queries. Thus, the CIM system may have to generate a very large number of form interfaces. Hence, for this approach to work, the system must be able to index these interfaces, and then return the most relevant ones, given a user's keyword query.

Context-Sensitive Services: To minimize user efforts and maximize their utilization of a CIM system, the system should provide context-sensitive services. For example, when the user accesses a page that contains publications, the system can consider all actions (querying, monitoring, etc.) that a user may want to do with those publications, then offer to execute those actions. These offers can be listed, e.g., on the right side of the page, similar to the way advertisements are displayed in search engine result pages. The key challenge here is to decide on which services to offer that would maximize users' utilization of the system, a challenge that is akin to deciding which advertisements to display in a search result page.

Easy Access to the System: Finally, we cannot just rely on users going to the system frontpage to ask queries or to browse. Most users today suffer from information overload. It is likely that they will just use a major search engine (e.g., Google, Yahoo) most of the time to search for information, an observation also made by [26]. Hence, it is very important that we "open up" a CIM system for major search engines to crawl and index, so that when a user asks a keyword query that can potentially be answered by the system, then the search engine

will return a page of the system in the top few results. The key challenges then are (a) how to maximize the chance that search engines will place a CIM system page high in the ranked list, if by accessing that page, the user can fulfill his or her information need, and (b) once the user has accessed the page, how to enable the user to quickly express his or her information need, then answer it.

3.3 Encouraging, Capturing, and Exploiting Social Interactions

So far we have discussed CIM users in isolation. But a distinguishing characteristic of CIM settings is that the users form a community: they often interact with one another, and such interactions are often captured in the data. Hence, we should design CIM systems such that they encourage such social interactions, capture them, and exploit them.

To encourage social interactions, CIM systems can employ a plethora of social tools such as those that allow users to tag, blog, comment, bookmark, form mailing lists, etc. And indeed many current social networking systems deploy such tools. The main problem is that we simply do not know when a particular tool will work (in that many users will use it). Hence, we foresee two major challenges. The first challenge is to develop more social tools, on the ground that expanding the tool collection makes it more likely that users will find something they like, and thus initiating more social interaction. The second challenge is to develop a mechanism to systematically deploy combinations of social tools in a CIM setting, evaluate their effectiveness in encouraging user participation, and then retain and improve the best ones.

Many CIM users also interact *outside* the system, but traces of such interactions are often captured in the raw data. For example, if X appears on the PC of a workshop organized by Y, then it is likely that X and Y have exchanged emails and are sharing some common interests. Hence, another challenge is to mine such social interactions from the raw community data. While mining social interactions is not a new topic, a distinguishing aspect of CIM settings is the abundance of *temporal data*. CIM systems crawl and archive community data over time (e.g., DBLife has crawled and archived the data of the database research community over the past 2.5 years). Exploiting the temporal aspect of this data may allow us to infer social interactions and their strengths more accurately.

Once social interactions have been captured or inferred, they can be exploited for many purposes, such as enhancing keyword search, identifying experts, finding emerging hot trends, viral marketing of ideas and services, among others. This has been a very active area of research (e.g., see the proceedings of recent WWW, KDD, database, and AI conferences). In CIM contexts, since feeding data into the system and querying it pose major difficulties (as discussed in Sections 3.1 and 3.2), an important challenge is to find out how to exploit social interactions to address these difficulties.

3.4 The Enablers: Reputation Management, Explanation Generation, and Undo

We have discussed user contribution, user services, and social interaction. These challenges share a set of core problems, and hence it is important that we develop effective solutions to these problems. We consider in particular reputation management, explanation generation, and undo.

Reputation management means knowing how much to trust any user X and to manage X's contributions to the CIM system. Much work has addressed reputation management (e.g., [2, 29]), but no consensus has emerged on the best method, and it is unlikely that a single silver bullet exists. Hence, like the case for social tools, an important challenge is to develop solutions that deploy reputation management tools, evaluate them, and retain and improve the best ones.

Explanation generation means that the system can explain to a user why a particular inference is made (e.g., why X is a PC member of conference Y) or *not* made (e.g., why didn't the system infer that Z is also a PC member of Y). We found that users asked many such questions in the DBLife context, either because they simply wanted to know, or because they used the explanations to decide on how much to trust the inference made by the

system. We ourselves also often asked such questions for debugging purposes. Hence, providing explanations is important for the effective development and utilization of CIM systems. Further, showing explanations also often allows better user corrections. For example, if a user only says "this output is wrong", the system has to infer which operator or datum involved in producing that output is the culprit. However, if the user can see an explanation, he or she may be able to pinpoint the error for the system.

Providing explanations on why a particular inference is made can utilize lineage (a.k.a. provenance [10, 34]) maintained by the system. The problem of providing explanations on why a particular inference is *not* made appears to be far harder, and has received little attention.

Finally, the undo capability allows users to roll the system back to a previous state. This capability is absolutely critical. As one user explained to us "without knowing that I can undo, I will not be willing to experiment with the features that the system provides". As Wikipedia demonstrates, undo is also important for managing malicious users. To enable this capability, a CIM system must log *everything*, including all user interactions. Then, the system must decide how much to allow users to undo. The problem is that if the system allows users to undo deep into the "past", it must limit concurrent editing of users, or risks losing user edits that build on a "transaction" that is later undone. How to strike the right balance here is a difficult question.

4 Concluding Remarks

As our field expands beyond managing structured data, to consider unstructured data in "Web 2.0" contexts, it is important that we discuss how the role of users has fundamentally changed in the new contexts, and what user-centric challenges those changes entail.

In this paper we have contributed to this broader discussion, drawing from our initial experience in the Cimple project on community information management systems. We described how users of such systems often act as active contributors, information explorers, and social players. For the role of active contributors, the key challenge is to enable users to supply or edit any kind of data, code, and domain knowledge, using whichever user interfaces they find most convenient. For the role of information explorers, the key challenge is to enable users to query the system using whichever query mode they find most convenient, and to switch seamlessly between the query modes with minimal effort. For the role of social players, the key challenge is to develop a broad range of social tools and mechanisms to select the most effective tools. Finally, we made the case that reputation management, explanation generation, and undo are critical in addressing the above challenges.

References

[1] http://oak.cs.ucla.edu/blogocenter.

[2] B. Adler and L. Alfaro. A content-driven reputation system for Wikipedia. In *Proc. of WWW-07*, 2007.

[3] E. Agichtein. Web information extraction and user modeling: towards closing the gap. *IEEE Data Engineering Bulletin*, 28(4), 2005.

[4] S. Agrawal, S. Chaudhuri, and G. Das. DBexplorer: A system for keyword search over relational databases. In *Proc. of ICDE-02*, 2002.

[5] R. Almeida, B. Mozafari, and J. Cho. On the evolution of Wikipedia. In *Proc. of the Int. Conf. on Weblogs and Social Media*, 2007.

[6] S. Amer-Yahia. A database solution to search 2.0 (keynote talk). In *Proc. of WebDB-07*, 2007.

[7] N. Bansal and N. Koudas. Blogscope: Spatio-temporal analysis of the blogosphere. In *Proc. of WWW-07*, 2007.

[8] B. Bhattacharjee, J. Glider, R. Golding, G. Lohman, V. Markl, H. Pirahesh, J. Rao, R. Rees, and G. Swart. Impliance: A next generation information management appliance. In *CIDR*, 2007.

[9] S. Boulakia, O. Biton, S. Davidson, and C. Froidevaux. Bioguidesrs: Querying multiple sources with a user-centric perspective. In *Bioinformatics*, 2007.

[10] P. Buneman and W. Tan. Provenance in databases (tutorial). In *Proc. of SIGMOD-07*, 2007.

[11] M. Cafarella, C. Re, D. Suciu, O. Etzioni, and M. Banko. Structured querying of Web text data: A technical challenge. In *Proc. of CIDR-07*, 2007.

[12] P. DeRose, W. Shen, F. Chen, Y. Lee, and D. Burdick. DBLife: A community information management platform for the database research community (demo). In *Proc. of CIDR-07*, 2007.

[13] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases*, 29(1), 2006.

[14] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing information extraction (tutorial). In *Proc. of SIGMOD-06*, 2006.

[15] Mehmet Altinel et. al. Mafia: A mashup fabric for intranet applications (demo). In *Proc. of VLDB-07*, 2007.

[16] C. Giles, K. Bollacker, and S. Lawrence. Citeseer: an automatic citation indexing system. In *Proc. of DL-98*, 1998.

[17] M. Gubanov and P. Bernstein. Structural text search and comparison using automatically extracted schema. In *Proc. of WebDB-06*, 2006.

[18] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRank: Ranked keyword search over xml documents. In *Proc. of SIGMOD-03*, 2003.

[19] A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems (invited paper). In *Proc. of PODS-06*, 2006.

[20] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proc. of VLDB-02*, 2002.

[21] A. Hulgeri and C. Nakhe. Keyword searching and browsing in databases using BANKS. In *Proc. of ICDE-02*, 2002.

[22] A. Jain, A. Doan, and L. Gravano. SQL queries over unstructured text databases. In *Proc. of ICDE-07 (poster)*, 2007.

[23] R. Krishnamurthy, S. Raghavan, J. Thathachar, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Engineering Bulletin, Special Issue on Probabilistic Databases*, 29(1), 2006.

[24] J. Liu, X. Dong, and A. Halevy. Answering structured queries on unstructured data. In *Proc. of WebDB-06*, 2006.

[25] J. Luxenburger and G. Weikum. Exploiting community behavior for enhanced link analysis and web search. In *Proc. of WebDB-06*, 2006.

[26] J. Madhavan, A. Halevy, S. Cohen, X. Dong, S. Jeffery, D. Ko, and C. Yu. Structured data meets the Web: A few observations. *IEEE Data Engineering Bulletin*, 29(4), 2006.

[27] J. Madhavan, S. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale data integration: You can only afford to pay as you go. In *Proc. of CIDR-07*, 2007.

[28] Z. Nie, J. Wen, and W. Ma. Object-level vertical search. In *Proc. of CIDR-07*, 2007.

[29] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. *Communications of the ACM*, 43(12):45–48, 2000.

[30] P. Roy, M. Mohania, B. Bamba, and S. Raman. Toward automatic association of relevant unstructured content with structured query results. In *Proc. of CIKM-05*, 2005.

[31] M. Sayyadian, A. Doan, and L. Gravano. Efficient keyword search over heterogeneous relational databases. In *Proc. of ICDE*, 2007.

[32] N. Taylor and Z. Ives. Reconciling changes while tolerating disagreement in collaborative data sharing. In *Proc. of SIGMOD-06*, 2006.

[33] F. Wang, C. Rabsch, P. Kling, P. Liu, and P. John. Web-based collaborative information integration for scientific research. In *Proc. of ICDE-07*, 2007.

[34] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. of CIDR-05*, 2005.