Towards Next Generation CiteSeer: A Flexible Architecture for Digital Library Deployment

I.G. Councill[1], C.L. Giles[1]
E. Di Iorio[2], M. Gori[2], M. Maggini[2], and A. Pucci[2]

[1] School of Information Sciences and Technology, The Pennsylvania State University, 332 IST Building University Park, PA 16802
{icouncil, giles}@ist.psu.edu
[2] Dipartimento di Ingegneria dell'Informazione, University of Siena, Via Roma, 56. Siena, Italy
{diiorio, marco, maggini, augusto}@dii.unisi.it

Abstract. CiteSeer began as the first search engine for scientific literature to incorporate Autonomous Citation Indexing, and has since grown to be a well-used, open archive for computer and information science publications, currently indexing over 730,000 academic documents. However, CiteSeer currently faces significant challenges that must be overcome in order to improve the quality of the service and guarantee that CiteSeer will continue to be a valuable, up-to-date resource well into the foreseeable future. This paper describes a new architectural framework for CiteSeer system deployment, named CiteSeer Plus. The new framework supports distributed indexing and storage for load balancing and fault-tolerance as well as modular service deployment to increase system flexibility and reduce maintenance costs. In order to facilitate novel approaches to information extraction, a blackboard framework is built into the architecture.

1 Introduction

The World Wide Web has become a staple resource for locating and publishing scientific information. Several specialized search engines have been developed to increase access to scientific literature including publisher portals such as the ACM Portal[1] and IEEE Xplore[2] as well as other academic and commercial sites including the Google Scholar[3]. A key feature common to advanced scientific search applications is citation indexing [3]. Many popular commercial search services rely on manual information extraction in order to build citation indexes; however, the labor involved is costly. Autonomous citation indexing (ACI) [4] has emerged as an alternative to manual data extraction and has proven to

[1] http://portal.acm.org/portal.cfm
[2] http://ieeexplore.ieee.org
[3] http://scholar.google.com

J. Gonzalo et al. (Eds.): ECDL 2006, LNCS 4172, pp. 111–122, 2006.

be successful despite some loss of data accuracy. Additionally, the ACI model has traditionally been coupled with autonomous or semi-autonomous content acquisition. In this approach, focused crawlers are developed to harvest the web for specific types of documents, in this case academic research documents, in order to organize distributed web content within a single repository. Automatic content acquisition is particularly useful for organizing literature that would otherwise be difficult to locate via general search engines [8].

CiteSeer [4] emerged as one of the first focused search engines to freely provide academic papers, technical reports, and pre-prints, and is also the first example of a working ACI system. CiteSeer consists of three basic components: a focused crawler or harvester, the document archive and specialized index, and the query interface. The focused spider or harvester crawls the web for relevant documents in PDF and PostScript formats. After filtering crawled documents for academic documents, these are then indexed using autonomous citation indexing, which automatically links references in research articles to facilitate navigation and evaluation. Automatic extraction of the context of citations allows researchers to determine the contributions of a given research article quickly and easily; and several advanced methods are employed to locate related research based on citations, text, and usage information. Additional document metadata is extracted from each document including titles, author lists, abstracts and reference lists, as well as the more recent addition of author information such as affiliations and contact information [6] as well as acknowledgement information [5]. CiteSeer is a full text search engine with an interface that permits search by document or by numbers of citations or fielded searching, not currently possible on general-purpose web search engines.

CiteSeer has proven its usefulness to the computer and information science communities. The CiteSeer installation at Penn State University[4] currently receives over one million requests and serves over 25 GB of information daily. The CiteSeer service is currently being made more available to the world community through the advent of several mirrors. At the time of this writing there are CiteSeer mirrors hosted at MIT, Switzerland, Canada, England, Italy, and Singapore in various stages of completion. However, CiteSeer currently faces significant challenges of interoperability and scalability that must be overcome in order to improve the quality of the services provided and to guarantee that CiteSeer will continue to be a valuable, up-to-date resource well into the foreseeable future.

The current architecture of the CiteSeer application is monolithic, making system maintenance and extension costly. Internal system components are not based on any established standards, such that all interoperability features incorporated have necessarily been crafted as wrappers to exposed functionality. The resulting lack of integration reduces the potential of CiteSeer to serve the research community. Additionally, as the CiteSeer collection grows (to over 730,000 documents as of the time of this writing), query latencies are rising and document updates are becoming increasingly cumbersome as the system pushes the boundaries of its current architecture.

[4] http://citeseer.ist.psu.edu

Recently, other ACI-enabled search engines for scientific literature have been developed, including Google Scholar. Although Google Scholar indexes at least an order of magnitude more documents than CiteSeer, CiteSeer remains competitive as an open archive and offers more features. A separate effort that has shown much promise is OverCite, a re-implemenation of CiteSeer within a peer-to-peer architecture based on distributed hash tables [15].

In this paper we present our own re-invention of CiteSeer, currently named CiteSeer Plus. This work builds on a previous architectural proposal for digital libraries [13]. CiteSeer Plus is based upon a new architecture designed to be flexible, modular, and scalable. As CiteSeer is currently operated within an academic environment with a focus on research as well as production, we have developed a framework that allows scalable, distributed search and storage while easing deployment of novel and improved algorithms for information extraction as well as entirely new service features.

The resulting architecture is oriented toward a collection of deployed services instead of a traditional web search engine approach. Each service component can be treated as a stand-alone application or as part of a larger service context. Users and programs can interact directly with individual services or with the entire system through web-based service front-ends such as a traditional search engine interface, consistent with ideas emerging from *Web 2.0* [11].

2 Project Goals

Flexibility. CiteSeer's current monolithic architecture limits the extensibility of the system. Information extraction routines are difficult to upgrade or change since they are tightly coupled with other system components. Not only does this cause maintenance difficulty, but it also limits the potential scope of the CiteSeer system. Adopting a highly modular service-oriented architecture will make the system more easily extendable with new services and more adaptable to different content domains. This is a core requirement for a next-generation CiteSeer service. Although an API has been developed for the existing CiteSeer [13], the API does not expose internal system functionality that is needed for a powerful extension environment. To alleviate this problem, each service module should carry its own API. This will allow service extensions to combine components in a flexible manner without incurring the overhead of refactoring existing code, and will allow the system to be more easily extensible to novel content domains.

Performance. A next-generation CiteSeer system must show improvements over the current system in terms of both query processing and update performance. Due to the current indexing and database framework, CiteSeer shows significant performance degradation when handling more than five simultaneous queries. Traffic spikes often account for more than 30 simultaneous queries and as many as 130 simultaneous connections have been observed. The resulting performance drop often limits the query response times to well below acceptable standards, in many cases turning users away outright. The new system should be able to handle at least 30 simultaneous queries without significant performance degradation. In addition, CiteSeer currently indexes no more than 3-4 papers per

minute, resulting in poor speed for acquiring new content. The update processes are large batch operations that typically take three days for every two weeks of content acquisition. To improve the freshness of information in the repository, it is desirable for a next-generation CiteSeer architecture to handle content updates quickly in an iterative process, so new content can be made available immediately after acquisition.

Distributed Operation. Although CiteSeer is currently implemented as a collection of processes that interoperate over network sockets, the architecture does not currently support redundant service deployment. This situation is mitigated through the use of Linux Virtual Server for service load balancing and fail-over; however, this increases maintenance demands and does not support distributed operation in a WAN environment. There is no support for propagating updates to mirrors without large file copies containing much redundant information. The new system should be natively capable of distributed operation with no single point of failure and should be easily extendable to support incremental updates over a WAN deployment.

3 System Features and Architecture

This section details the features supported by the CiteSeer Plus framework as well as its architecture. CiteSeer Plus is designed to be a flexible platform for digital library development and deployment, supporting standard digital library features as well as plugins for advanced automation. In keeping with the goals presented in Section 2, the feature set is expandable based on application or domain requirements and the user interface to the application is arbitrary, to be built on top of a rich system API. An experimental prototype of a CiteSeer Plus deployment is publicly available[5].

The CiteSeer Plus system architecture is highly modular. In the following sections every module is presented and module interactions are discussed. The system architecture is organized in four logical levels as shown in Figure 1.

The *Source Level* contains document files and associated data. The *Core Level* contains the central part of the system in which document and query processing occurs. The *Interface Level* offers interface functions to allow the communication between the Core Level and services that can be developed using CiteSeer Plus (in the Service Level). This level is implemented as a collection of Web Services. Finally, the *Service Level* contains every service that is running on top of the CiteSeer Plus system.

Figure 2 maps the levels to the actual system architecture. At the Core Level are the sets of master and slave indexing nodes. These sets contain redundant indexing nodes tailored for specific tasks within the CiteSeer Plus system, and are the fundamental processing nodes. A single node is made of different subcomponents. Figure 3 shows the details of a master indexing node. We can describe these nodes by following a typical paper lifecycle through an indexing node.

[5] http://p2p.science.unitn.it/cse

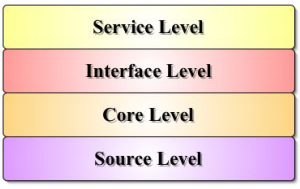

Fig. 1. Logical levels

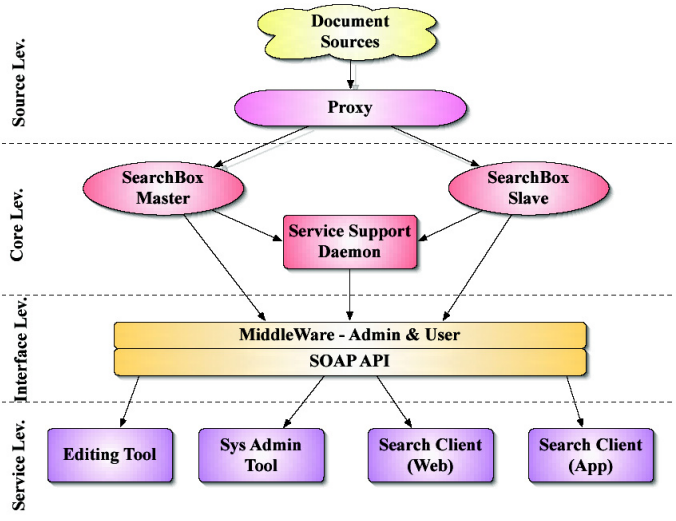

Fig. 2. System architecture overview

The system is agnostic regarding the method of content acquisition. New content may be harvested by a crawler, received from an external library, or submitted by users, so long as documents are posted to the system via a supported acquisition interface. Once a paper has been received it is stored in the *PDF cache* to guarantee persistence of a document in the original format, then submitted to a document processing workflow for integration into the system data. The paper encounters a *PDF parser* whose duty is to extract text from the original file and produce a new XML-based representation. This new document contains text and some layout information such as text alignment, size, style, etc. Next the raw XML file enters the metadata extraction subsystem. This subsystem is composed of several modules, including a *BlackBoard Engine* that is used to run a pool of experts (shown as *EXP 1*, *EXP 2*, ..., *EXP N* in Figure 3) that cooperate to extract information from the document. This process is presented in more detail in Section 5. This process outputs an XML document that contains all tagged metadata.

Finally the paper is ready to be indexed: the labeled XML is stored in the *XML cache* (to make it available for later retrieval) and passed to the *indexer*.

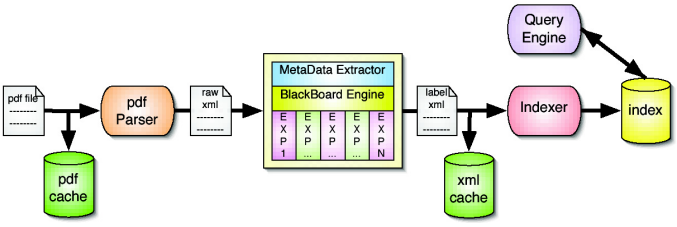

Fig. 3. Indexing node detailed structure

At this point the *Query Engine* will be able to provide responses to user or system queries involving the indexed document. Metadata elements are stored in separate index fields, enabling complex queries to be built according to various document elements. Every indexing node is able to communicate with the other system components by exposing a set of methods as a web service. The entire indexing process takes place in on-line mode, such that a paper entering the system will enter one or more indexing nodes for immediate consumption by the system.

In addition to normal indexing nodes (called *master nodes*) there are also *slave nodes*. Slave nodes are a lighter version of master nodes; their inner structure is just the same as seen in Figure 3, with the exception that slave nodes do not maintain any kind of cache (no PDF cache nor XML cache). Furthermore, their indexes contain only metadata slices (such as title, author, abstract and citation slices), but they do not contain generic text slices, which support full-text queries. Both master and slave nodes can be deployed redundantly for load balancing. During initial indexing, a paper can be routed to any number of slave nodes but must be routed to at least one master node, in order to allow the system to provide full-text indexing and caching. Slave nodes are provided in order to support frequent operations such as citation browsing, graph building, and paper header extraction (a header contains just title, author, abstract and references) since those operations do not require access to a full-text index. In this way, performance can be improved by adding new slave nodes that do not incur large additional storage requirements. Slave nodes can also be used to support system processing for graph analysis and the generation of statistics without affecting performance for user queries; however, only a single master node is needed to run a CiteSeer Plus system.

It is also possible to split the indexes among different machines (in this case the controller will send a query to all of them and then organize the different responses received). At the same time, indexes can be redundant; that is, the same indexes can be managed by different mirror nodes running on different computers in order to improve system performance through load balancing. In Figure 4 we show a typical system configuration.

In this deployment we have divided the index into two parts (A and B), so every time a document is accepted by the system, the controller decides which subindex will receive the document, such that indexes are balanced. Nodes in

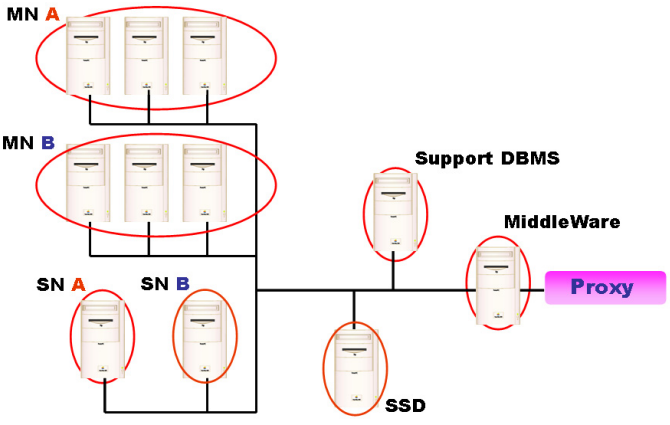

Fig. 4. Example of system deployment

the same node set have the same indexes to support index redundancy. In this example "MN A" (master node set of subindex A) contains three computers running three separated and identical master node instances, and "SN A" provides support to "MN A" nodes. In this case "SN A" contains only one slave node, but, in general, it can be a set of slave nodes. The same configuration is kept for the "B" (in this case we have "MN B" and "SN B"). In this scenario, if a user submits a full-text query the controller will route the query to a master node chosen from the "MN A" set and one from "MN B", so the system, in this sample configuration, is able to provide service for up to three concurrent users just the same as one by sharing the workload among redundant master node mirrors inside "MN A" and "MN B". The same situation happens when a query does not involve a full-text search, but is just referred to metadata indexes. The only difference in this case is the fact that slave nodes ("SN A" and "SN B") will respond to the query instead of master nodes.

At the Interface Level we find the *Middleware*, which is the active part of the external SOAP API. This component converts API methods into procedure calls on the services provided by the components in the Core Level. The Middleware contains methods to perform user authentication control in order to determine whether a system user is authorized to perform the requested operations. The Middleware also manages the controller threads and performs query and paper routing in order to maintain consistency in the distributed and redundant sets of Master and Slave Nodes. Every operation regarding resource distribution and redundancy is performed in this module.

Each system component exposes public methods through the *SOAP API*, allowing the development of discrete services using the CiteSeer Plus framework. The Service Level uses the API to define prescribed usage scenarios for the system and interfaces for user control. This level contains HTML forms and representations for user and administrative interaction. Some exemplar services that have been built include tools to add or remove documents and correct

document metadata, deployment configuration tools, and search interfaces for users (a web application) or programs (via SOAP).

4 Citation Graph Management

A document citation graph is a directed graph where the nodes correspond to the documents and edges correspond to citation relationships among the documents. A document citation graph is useful for deriving bibliometric analyses such as computing document authorities and author importance as well as to perform social network analysis. In order to construct a document citation graph all citations contained in each document must be identified and parsed, and then the citations must be matched to corresponding document records. CiteSeer Plus uses an approach that differs in many ways from the legacy CiteSeer.

CiteSeer's method could be defined as a *"hard approach"*. Each citation is parsed using heuristics to extract fields such as title, authors, year of publication, page numbers and the citation identifier (used to mark the citation in the body text). The fields of each citation are compared with one another based on a string distance threshold in order to cluster citations into groups representing a single document. Finally, the metadata from each citation group is compared to existing document records in order to match the citations to documents. Citations to a given paper may have widely varying formats; hence, developing rules for citation field identification can be very time consuming and error prone. CiteSeer's approach relies heavily on off-line computations in order to build the document citation graph. If no document is found to match a citation group, all citations in the group are *unsolved*, and cannot be solved until the next graph update, even if a matching document enters the system beforehand.

The CiteSeer Plus approach could be defined as a *soft approach*. Our method is less computationally costly and can be performed online, in an approach similar to the SFX system [16]. The process of building the citation graph in CiteSeer Plus is query-based; that is, the citations are solved using queries performed in the *query module*. The Indexer allows metadata to be stored in different sub-indexes (*slices*) and so a query can be performed on a specific slice of the main index. Subfields parsed from citations are used to perform complex document queries on appropriate index slices and the top document is found to match a citation if it's similarity to the query surpasses a given threshold. In the other direction, to find citations matching a new document, CiteSeer Plus makes a query using all the words of the document title and authors. This query is performed on the citation slice; thus the query results are all documents that have a citation containing some words of the query.

Master nodes do not cache the document citation graph since they have to provide query results that are as fresh as possible. However, slave nodes can use a query result caching mechanism in order to improve performance at the cost of reduced information freshness. Repository statistics are built using slave nodes, but user queries operate on the master node. When a user tries to follow a citation, this produces a corresponding query on the master node and the user

will obtain one or more documents that are likely to match the citation. This framework relieves workload on dynamic components that handle user queries while allowing detailed statistics and graph management activities to be handled online within separate components.

5 Metadata Extraction System

Metadata extraction is the most difficult task performed by an automated digital library system for research papers. In the literature, there are two main approaches to information extraction: knowledge engineering and machine learning. In the knowledge engineering approach, the extraction rules used by the system are constructed manually by using knowledge about the application domain. The skill of the knowledge engineer plays a large role in the level of system performance, but the best performing systems are often handcrafted. However, the development process can be very laborious and sometimes the required expertise may not be available. Additionally, handcrafted rules are typically brittle and do not perform well when faced with variation in the data or new content domains. CiteSeer uses this approach, employing information about the computer science document styles (or templates) to extract metadata.

In the machine learning approach, less human expertise regarding template styles is required when customizing the system for a new domain. Instead, someone with sufficient knowledge of the domain and the task manually labels a set of training documents and the labeled data is used to train a machine learning algorithm. This approach is more flexible than the knowledge engineering approach, but requires that a sufficient volume of training data is available. In the last decade, many techniques have been developed for metadata extraction from research papers. There are two major sets of machine learning techniques in the metadata extraction literature. Generative models such as Hidden Markov Models (HMM) (e.g. [14], [9]) learn a predictive model over labeled input sequences. Standard HMM models have difficulty modeling multiple non-independent features of the observation sequence, but more recently Conditional Random Fields (CRF) have been developed to relax independence assumptions [7]. The second set of techniques is based on discriminative classifiers such as Support Vector Machines (SVM) (e.g. [6]). SVM classifiers can handle large sets of non-independent features. For the sequence labeling problem, [6] work in a two stage process: first classifying each text line independently in order to assign it a label, then adjusting these labels based on an additional classifier that examines larger windows of labels. The best performance in metadata extraction from research papers has been reached by McCallum and Peng in [12] using CRFs. The CiteSeer Plus metadata extraction system has been built to maximize flexibility such that it is simple to add new extraction rules or extraction models into the document processing workflow. In our metadata extraction system, different kinds of models can be used which have been trained for different or the same extraction tasks using various techniques, including but not limited to HMM, CRF, regular expression, and SVM classifiers. The CiteSeer Plus metadata extraction system

is based on a blackboard architecture ([10], [1], [2]) such that extraction modules can be designed as standalone processes or within groups of modules with dependencies. A blackboard system consists of three main components:

Knowledge Sources (in our framework these are named Experts): independent modules that specialize in some part of the problem solving. These experts can be widely different in their inference techniques and in their knowledge representation.

BlackBoard: a global database containing the input data, partial solutions and many informational items produced by experts to support the problem solving.

Control component: a workflow controller that makes runtime decisions about the course of problem solving. In our framework, the control component consists of a set of special experts called *scheduling experts* that are able to schedule the knowledge sources registered in the framework. The scheduling expert is chosen by the controller components based on the problem solving strategy that is employed and the kinds of metadata that the system needs to progress. Using different scheduling experts, it is possible to change the problem solving strategy dynamically in order to experiment with various learning strategies.

Although an individual expert can be independent from all the other experts registered in the framework, each expert can declare its information dependences, that is, all the information that it needs to work. The *control component* activates the expert when all these dependences are satisfied. As such, experts can be activated when all the information required by the expert has been extracted and stored on the *BlackBoard module*. The experts declare their skills (the information they can extract) to the *Control component*, such that during the problem solving (metadata extraction), at the right moment the control component can activate the experts, and the controller can reason about which intermediary experts must be employed in order to reach a later result. The BlackBoard groups similar information and registers expert accuracies based on the *prior expertise*[6] declared by each expert. In this way, if more than one expert produces the same (or similar) kinds of information, the accuracy value of that information will be computed as the joint confidence among the experts.

An example configuration may group experts into three classes or *functional levels*, although the framework does not restrict the processing workflow. The first level is the *Entity Recognition* level. In this level are all the experts able to give words a specific semantic augmentation, including part-of-speech tagging and recognition of named entities such as first or last name, city, country, abbreviation, organization, etc. Experts at this level will be activated first for processing workflows. The second level is the *Row Labeling* level. At this level are all the experts able to classify a paper line with one or more defined labels such as author, title, affiliation, citation, section title and so on. The experts at this level classify the paper lines using a document representation supplied by the *Document module*, a framework object able to elaborate the document

[6] The prior expertise is a measure of expert ability (F score) on a standard dataset.

structure by supplying a representation based on many different features regarding line contents, layout and font styles. Row labeling can be an iterative process, reclassifying lines based on tagged context in subsequent passes. The last level is the *Metadata Construction* level. Using all the extracted information from the previous levels, the experts at this level can build the final metadata record for a document.

6 Summary

This paper has presented a new version of the CiteSeer system, showing significant design improvements over its predecessor. The new system reproduces every core feature of the previous version within a modular architecture that is easily expandable, configurable, and extensible to new content domains. Increased flexibility is obtained through a design based on customizable plug-in components (for the metadata extraction phase) and the extensive use of web service technology to provide an interface into every system component. CiteSeer Plus can also be a useful tool for researchers or other developers interested in information retrieval and information extraction, as CiteSeer Plus can be used as a powerful yet easy to use framework to test new ideas and technologies by developing third party applications that bind with specific components of the CiteSeer Plus framework.

Acknowledgments

We thank Nicola Baldini and Michele Bini (FocuSeek.com) for fruitful discussions, suggestions and support during the system design and development process. We also thank Fausto Giunchiglia and Maurizio Marchese (University of Trento, Italy) for fruitful discussions that have aided the evolution of the system.

References

1. B. L. Buteau. A generic framework for distributed, cooperating blackboard systems. *Proceedings of the 1990 ACM annual conference on Cooperation*, p.358-365, February 20-22, 1990.
2. H. Chen , V. Dhar. A knowledge-based approach to the design of document-based retrieval systems. *ACM SIGOIS Bulletin*, v.11 n.2-3, p.281-290, Apr. 1990.
3. E. Garfield. Science Citation Index - A new dimension in indexing. *Science*, 144, pp. 649-654, 1964.
4. C.L. Giles, K. Bollacker and S. Lawrence. *CiteSeer: An Automatic Citation Indexing System*, Digital Libraries 98: Third ACM Conf. on Digital Libraries, ACM Press. New York, 1998, pp. 89-98.
5. C.L. Giles and I.G. Councill. Who gets acknowledged: measuring scientific contributions through automatic acknowledgement indexing. *PNAS*, 101, Number 51, pp. 17599-17604, 2004.

6. H. Han, C. Lee Giles, E. Manavoglu, H. Zha, Z. Zhang, E. A. Fox. Automatic Document Metadata Extraction using Support Vector Machines. *Proceedings of the 2003 Joint Conference on Digital Libraries (JCDL03)*, 2003.
7. J. Lafferty, A. McCallum, and F. Pereira. Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. *In International Conference on Machine Learning*, 2001.
8. S. Lawrence, C. Lee Giles. Searching the World Wide Web. *Science*, 280, Number 5360, pp. 98-100, 1998.
9. T. R. Leek. Information extraction using hidden Markov models. *Masters thesis, UC San Diego*, 1997.
10. H. Penny Nii. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *The AI Magazine*, VII(2):38–53, Summer 1986.
11. T. O'Reilly. What Is Web 2.0 Design Patterns and Business Models for the Next Generation of Software. *http://www.oreillynet.com/pub/a/oreilly/tim/news /2005/09/30/what-is-web-20.html*
12. F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. *Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics(HLT-NAACL)*, pages 329336 (2004).
13. Y. Petinot, C. Lee Giles, V. Bhatnagar, P. B. Teregowda, H. Han, I. Councill. A Service-Oriented Architecture for Digital Libraries. *ICSOC04*, November 15-19, 2004.
14. K. Seymore, A. McCallum and R. Rosenfeld. Learning hidden Markov model structure for information extraction. *In Papers from the AAAI-99 Workshop on Machine Learning for Information Extration*, pages 3742, July 1999.
15. J. Stribling, I.G. Councill, M.F. Kaashoek, R. Morris, and S. Shenker. Overcite: A cooperative digital research library. In *Proceedings of The International Workshop on Peer-To-Peer Systems (IPTPS 05)*, Ithaca, NY, 2005 .
16. H. Van de Sompel, P. Hochstenbach. Reference linking in a hybrid library environment. Part 1: Frameworks for linking. *D-Lib Magazine*, v.5 n.4, 1999.