Herwig Unger
Thomas Böhme
Armin Mikler (Eds.)

LNCS 2346

Innovative Internet Computing Systems

Second International Workshop, IICS 2002
Kühlungsborn, Germany, June 2002
Proceedings

Springer

Herwig Unger Thomas Böhme
Armin Mikler (Eds.)

Innovative Internet Computing Systems

Second International Workshop, IICS 2002
Kühlungsborn, Germany, June 20-22, 2002
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Herwig Unger
Universität Rostock, FB Informatik
18051 Rostock, Germany
E-mail: hunger@informatik.uni-rostock.de

Thomas Böhme
TU Illmenau, Institut für Mathematik
Postfach 10 05 65, 98684 Ilmenau, Germany
E-mail: tboehme@theoinfo.tu-ilmenau.de

Armin Mikler
University of North Texas
College of Art and Sciences, Department of Computer Science
76203 Denton, TX, USA
E-mail: mikler@cs.unt.edu

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Innovative internet computing systems : second international workshop ; proceedings / IICS 2002, Kühlungsborn, Germany, June 20 - 22, 2002. Herwig Unger ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Lecture notes in computer science ; Vol. 2346)
ISBN 3-540-43790-8

CR Subject Classification (1998): C.2, D.2, F.3, H.3-4, I.2.11, K.4.3-4

ISSN 0302-9743
ISBN 3-540-43790-8 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Christian Grosche, Hamburg
Printed on acid-free paper SPIN 10869985 06/3142 5 4 3 2 1 0

Preface

I^2CS 2002 was the second workshop on Innovative Internet Computing Systems, a series of international workshops on system and information management for the Next Generation Internet (NGI). The workshop series commenced with I^2CS 2001, which was held at the Technical University of Ilmenau. It brought together scientists whose research addressed different aspects of Internet-based and network-centric computing. This year's workshop was held in the inspiring atmosphere of the old Baltic Sea resort of Kühlungborn near Rostock (Germany).

The unprecedented pervasiveness of network access and the associated emergence of large distributed computing infrastructures have presented researchers with new challenges in Information and Web technology. The management and retrieval of web-based information, the classification of contents, and the management of web-based communities are some of the key areas addressed by some of this year's contributions. Other papers focus on the structure and retrieval of information from large distributed data bases as well as the representation of the distributed nature of information by the means of graph-theoretical models.

Like I^2CS 2001, this year's workshop was organized by the Gesellschaft für Informatik (GI) in Germany to support the exchange of experiences, results, and technology in the field. The 21 papers (2 invited, 19 regular contributions) presented at the conference and in the present volume were selected from more than 30 submissions. Every submission was carefully reviewed by three members of the program committee.

We would like to thank all those who contributed to this book for their excellent work and their great cooperation. Roswitha Fengler and Katrin Erdmann deserve special gratitude for their great efforts and perfect work concerning all administrative matters of the workshop. We wish to acknowledge the substantial help provided by our sponsors: the University of Rostock and the TKK Techniker Krankenkasse Rostock.

We hope all participants enjoyed a successful workshop, made a lot of new contacts, held fruitful discussions helping to solve the actual research problems, and had a pleasant stay on the coast of the Baltic Sea. Last but not least we hope to see you again at the third I^2CS conference in 2003, which will be held in the Leipzig area in the heart of Germany.

June 2002

Herwig Unger (Chair)
Thomas Böhme (Co-chair)
Armin R. Mikler

Organization

I^{2}CS was organized by the Gesellschaft für Informatik (GI) in Germany.

Executive Committee

Roswitha Fengler
Katrin Erdmann

Steering Committee

Herwig Unger (Chair)
Thomas Böhme (Co-chair)
Armin R. Mikler

Program Committee

A. Brandstädt	G. Hipper	A. Pears
J. Brooke	N. Kalyaniwalla	M.A.R. Dantas
M. Bui	K. Kleese	D. Reschke
N. Deo	N. Krier	A. Ryjov
M. Dietzfelbinger	P. Kropf	M. Sommer
W. Fengler	M. Kunde	D. Tavangarian
T. Haupt	R. Liskowsky	D. Tutsch
G. Heyer	S. Lukosch	T. Ungerer

Sponsoring Institutions

University of Rostock
TKK Techniker Krankenkasse

Table of Contents

Workshop Innovative Internet Computing Systems

Living Hypertext - Web Retrieval Techniques for Traditional Database-Centric Information 1
Ralf-Dieter Schimkat, Wolfgang Küchlin (Universität Tübingen), Frank Nestel (INA-Schaeffler KG)

Automatic Analysis of Large Text Corpora - A Contribution to Structuring WEB Communities 15
Gerhard Heyer, Uwe Quasthoff, Christian Wolff (Universität Leipzig)

A Data Mining Architecture for Distributed Environments 27
Mafruz Zaman Ashrafi, David Taniar, Kate Smith (Monash University)

Collaborative Highlighting for Real-Time Group Editors 39
Haifeng Shen, Chengzheng Sun (Griffith University)

Extending the Modeling Efficiency of the UML Activity Diagram for the Design of Distributed Systems 51
Olga Fengler, Wolfgang Fengler, Veselka Duridanova (Technical University Ilmenau)

An XML Knowledge Base System for Scheduling Problems 63
Leonilde Rocha Varela (University of Minho), Joaquim Nunes Aparício (New University of Lisbon), Sílvio Carmo Silva (University of Minho)

Compressibility as a Measure of Local Coherence in Web Graphs 75
Jeannette C.M. Janssen, Nauzer Kalyaniwalla (Dalhousie University)

On the Spectrum and Structure of Internet Topology Graphs 83
Danica Vukadinović, Polly Huang, Thomas Erlebach (ETH Zurich)

Characterizing the Citation Graph as a Self-Organizing Networked Information Space 97
Yuan An, Jeannette C.M. Janssen, Evangelos E. Milios (Dalhousie University)

Characterization and Management of Dynamical Behavior in a System with Mobile Components 109
Christian Erfurth, Wilhelm Rossak (Friedrich Schiller Universität Jena)

Ergotracer: An Internet User Behaviour Tracer 121
Carlos Gregorio Rodríguez (Universidad Complutense de Madrid), Pedro Palao Gostanza (Bankinter)

A Friendly Peer-to-Peer File Sharing System with Profit but without Copy Protection 133
Rüdiger Grimm, Jürgen Nützel (TU Ilmenau)

A Decentral Library for Scientific Articles 143
Markus Wulff (University of Rostock)

Optimisation of Distributed Communities Using Cooperative Strategies .. 153
Saiho Yuen, Peter Kropf (Université de Montréal), Gilbert Babin (HEC Montréal)

Proven IP Network Services: From End-User to Router and vice versa ... 169
Gerald Eichler, Ralf Widera (T-Systems Nova), Anne Thomas (TU Dresden)

Mobility Support for Replicated Real-Time Applications 181
Jörg Roth (University of Hagen)

Empirical Study of VBR Traffic Smoothing in Wireless Environment 193
Youjip Won, Bowie Shim (Hanyang University)

Compiling Rule-Based Agents for Distributed Databases 205
K.W. Ko, I.T. Kim, S.B. Yoo, K.C. Kim, S.D. Kim, J.S. Kim (Inha University), Y.S. Lee (Yonsei University)

A Formal Framework for E-Barter Based on Microeconomic Theory and Process Algebras 217
Natalia López, Manuel Núñez, Ismael Rodríguez, Fernando Rubio (Universidad Complutense de Madrid)

Invited Talk

Peer-to-Peer beyond File Sharing 229
Ulrike Lechner (University of Bremen)

Author Index 251

Living Hypertext — Web Retrieval Techniques

Ralf-Dieter Schimkat[1], Wolfgang Küchlin[1], and Frank Nestel[2]

[1] Wilhelm-Schickard Institute for Computer Science
`http://www-sr.informatik.uni-tuebingen.de`
Sand 13, 72074 Tübingen, Germany
`{schimkat,kuechlin}@informatik.uni-tuebingen.de`
[2] INA-Schaeffler KG
Industriestrasse 1–3, 91074 Herzogenaurach, Germany
`Frank.Nestel@de.ina.com`

Abstract. In this paper, we present a document metaphor called *Living Documents* for accessing and searching for digital documents in modern distributed information systems. Our approach is based upon a fine-grained document concept which glues computational services, data and meta data together. Viewing documents as micro servers is particularly well suited in environments where the document's content is changing continuously and frequently. Based on a case study of an existing state-of-the-art Web application, we show how to transform database-centric information systems into a hypertext of inter-linked *Living Documents*. We also discuss how to effectively use traditional as well as Web information retrieval techniques, namely topic distillation, in such hypertext environment. In particular, an extended version of Kleinberg's [11] algorithm is presented.

1 Introduction

It is generally agreed upon that the major task in information retrieval is to find relevant documents for a given query [9]. A document is a collection of digital information ranging from plain text files, data-related meta attributes to multi-media documents. Clearly a conceptual model of information retrieval dealing with text or multi-media documents should integrate different views on documents. Agosti et al. [1] point out that one of several differences in traditional information retrieval and information retrieval on the Web is the different kind of management of the collection of documents. In fact the web is a virtual collection since a real collection stored at one particular location such as one single database would be unmanageable.

In contrast to the Web, traditional information system design is about documents which are made persistent in a digital document archive, but their attributes (meta data) are kept in databases. Even virtual documents missing any digital content can be seen as an aggregation of their attributes. This leads to several potential drawbacks in classically designed information systems: (i) the specification of document attributes is bound to the database schema and is a priori determined at the time the database schema is set up. (ii) Furthermore, the static specification restricts a document's life cycle. For example, in most cases it is hard to add new kinds of meta data to the document's collection at run time.

Our goal is to provide a document metaphor and an implementation respectively which can be used in traditional as well as in Web information systems. Our approach is characterized by transforming documents into active containers managing their content and meta data in an uniform and extensible manner.

The main contributions of this paper are: (i) Introduction to a new document metaphor called *Living Documents*. (ii) Description of a complete implementation path of *Living Documents* based on a case study of a contemporary web information system. We centered our implementation around the concepts of mobile agents and general data description languages based on XML. (iii) We show how to deploy three different kinds of information retrieval techniques in *Living Documents*. In particular, we describe how to use state-of-the-art web information retrieval techniques as topic distillation within a web of inter-linked *Living Documents*. Finally, we present an extension of a well-known algorithm for the analysis of connectivity graphs in hypertext environments.

2 *Living Documents*

First, we give an introduction to the concept of *Living Documents*[3] from an abstract point of view neglecting any implementation details. In the next section we show an implementation path for *Living Documents*.

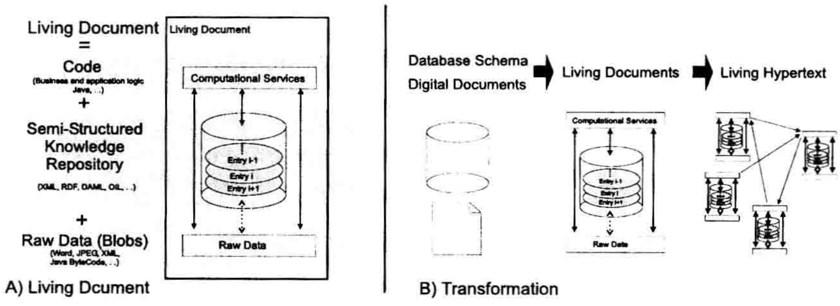

Fig. 1. A) Components of a *Living Document*. A *Living Document* is divided into three sections: *Raw Data* carries the document to manage, *Semi-Structured Data* contains all meta data about the managed documents, and the *Code* section keeps the computational services for accessing a *LD* and processing incoming requests (i.e. queries). B) Digital Documents are turned into *Living Documents*. *Living Documents* form a hypertext by keeping links similar to hypertext links to each other

[3] Home page of *Living Documents* at `http://www.living-documents.org`.

2.1 Towards a Micro Server Architecture

A *Living Document* (*LD*) is a logical and physical unit consisting of three parts, as depicted in Figure 1A:

1. code
2. semi-structured knowledge repository
3. raw data

CompServices are essentially *code* fragments which provide several facilities, such as access and query capabilities or general application services. The code fragments determine the degree of activities of a *LD* ranging from passive documents which are enriched with some arbitrary application logic to proactive documents. A proactive *LD* initiates complex tasks, discovers new services for instance and is more than just a reactive component. By deploying *LD*s the distinction between documents and applications blurs, because documents can contain application logic.

The *knowledge repository* of a *LD* provides facilities to store and retrieve information related to the document (raw data section) or to the whole *LD* itself. Each document has its own knowledge repository. Each knowledge repository is accessed through the code part. Basically a knowledge repository contains a set of *meta data* about the document itself. Each meta data is referred to as a document state information. A set of document state information builds a so-called document state report (DSR) which contains history-related information about who has accessed the document or when the document's attributes have been modified. In addition, it contains helpful links to other *LD*s which have some kind of relationship. Basically, a knowledge repository serves as an uniform access point for searching any kind of document-related meta data.

Each DSR is encoded as a semi-structured XML document according to the *SpectoML* [16]. Following an XML-based [21] implementation, the generation of a DSR is accomplished in an uniform way which favors neither a particular data format nor the use of special programming or scripting languages. The use of XML as the primary data format for document state information enables a DSR with query capabilities, such as the execution of structured queries to each document state information. Therefore, a DSR builds an XML-based knowledge repository which holds all relevant information about the entire document life cycle.

The *raw data* part can contain any information encoded as a digital document such as a word processing document, a music file or even serialized application code. Note that according to the definition given above, a *LD* does not need to have a real-world document contained in the raw data part. Thus, a *LD* solely consisting of computational logic and a knowledge repository is a well-defined *LD*.

Why is a *LD* called *living*? A *LD* is alive with respect to two key properties: First, implementing *LD*s as mobile agents they can move among nodes of a computer network, such as the Internet. That perfectly fits the notion of an autonomous and mobile entity pursuing its dedicated goals. Secondly, the ability to store and remove arbitrary artifacts into the knowledge repository changes the documents content naturally. It increases and decreases over time depending on the application domain and the environmental context the respective *LD* resides in. Even the raw data may evolve if reflected appropriately in the knowledge repository.

3 Case Study: Information Retrieval Using *Living Documents*

3.1 Case Study

The web-enabled n-tier client-server information system *Paperbase* serves as our motivating example. Its n-tier client-server architecture is typical for contemporary Web-enabled digital library and information systems. *Paperbase*[4] allows the creation of individual information workspaces using the Web. Users can easily create and manage their own workspace containing various media such as HTML, PDF, ASCII, or Office documents. The rationale behind *Paperbase* is to provide personal workspaces for users independent of their current physical location. Furthermore, they can easily share sub sets of their workspace among each other and collaborate.

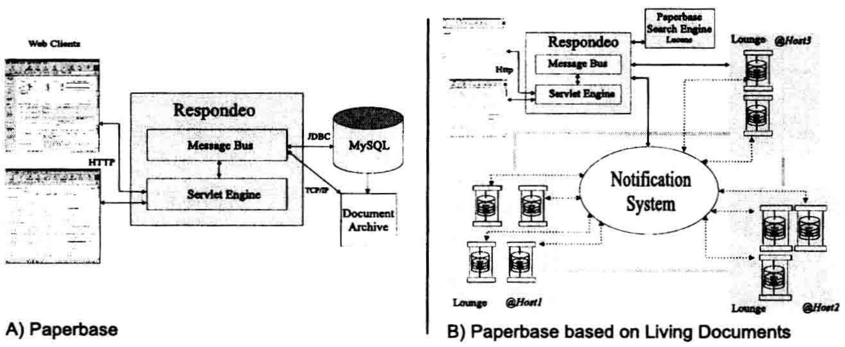

Fig. 2. A) Overview of the n-tier client-server architecture of *Paperbase*. B) Architectural overview of *PaperbaseLD* based on *Living Documents*

As depicted in Figure 2A, users issue requests over the Web which are forwarded to the application server *Respondeo* introduced in [17]. *Respondeo*'s message bus mediates incoming user requests to the requested back end tier. In the case of *Paperbase* the back end tier solely consists of one database which stores all information about the documents' attributes based on a relational

[4] *Paperbase* is developed at the department of the University of Tübingen. It currently contains about 1500 documents. See `http://www-sr.informatik.uni-tuebingen.de/~schimkat/pb` for further information about *Paperbase*.

database schema. Note, that the document's content itself is stored separately in a so-called digital document archive. Only a link to the archive is kept in the relational database.

3.2 Implementing *Living Documents*

Based on the n-tier client-server Web information system *Paperbase* described in Section 3.1, we designed and implemented *Paperbase* differently (*PaperbaseLD*) using the concept of *Living Documents* which act as micro servers for documents, as described in Section 2.

Computational Services. As defined in Section 2.1 each *LD* contains a set of computational services. Within *PaperbaseLD* an agent is the key abstraction for managing various computational services for a *LD*. It provides services for

- accessing the *LD*'s knowledge repository and raw data part
- querying the *LD*'s knowledge repository
- viewing the content of knowledge repository encoded as an XML document
- viewing the content *LD*'s raw data part.

We enriched *LD*s with mobility capabilities to take their location actively into account as argued in [19]. Therefore, we integrated *LD*s into the mobile agent framework *Okeanos* [15, 16]. *LD*s communicate and interact by exchanging messages in KQML (Knowledge Query Manipulation Language) [8]. In *PaperbaseLD* each "agentified" *LD* can dynamically reconfigure its computational services and add new services at run time. Each service and *LD*, respectively, is implemented in Java.

Knowledge Repository. As stated in Section 2.1, a knowledge repository contains a set of meta data or document state information which builds a DSR. In *PaperbaseLD* we encoded each DSR as a semi-structured XML document. Following an XML-based implementation, the generation of DSR is accomplished in an uniform way which neither does favor a particular data format nor the use of special programming or scripting languages.

Generally, each document state information belongs to a particular type of document descriptions. Within *PaperbaseLD* the meaning of a type is encoded as XML as well and kept separately from the document state information itself. Thus, a document state information contained in the *LD*'s knowledge repository is an instance of a particular type of a document description. The design rationale behind the separation of the actual document state information (syntax) and its meaning (semantic) is similar to the managing of semantically enriched XML documents in the semantic Web research community. However, from an implementation point of view we currently use a much simpler XML-based scheme to describe the meaning of document state information than the one proposed by the semantic web community - Resource Description Framework (RDF) [20]

In *PaperbaseLD* we currently defined several types of document state information, such as (i) access-related information about who is accessing the document with respect to time and location of the requesting document; (ii) history-related information about current and old locations of the mobile *LD*; (iii) the mapping of the relational database schema to document state information entries in the knowledge repository. This mapping is necessary due to the goal to provide at least similar retrieval facilities in *PaperbaseLD* as in *Paperbase*. We simply map each attribute in the relational schema which is directly related to the *LD* to a triple

$$DatabaseProperty = \langle type, name, value \rangle ,$$

where *type* is an unique schema descriptor for *PaperbaseLD*, *name* is a string built from the table and column name of the relational database of *Paperbase*, and *value* is the value as contained in the relational database system of *Paperbase*.

Raw Data Part. As specified in Section 2.1 the *LD*'s raw data part (Blob) can contain any information encoded as a digital document. From an implementation point of view, we use in *PaperbaseLD* various kinds of documents such as Office documents, HTML pages, PDF documents, and several other data formats.

3.3 Distributed Information Retrieval Using *Living Documents*

Figure 2B gives an architectural overview of *PaperbaseLD* using *LD*s. The main application components of *PaperbaseLD* are the application server *Respondeo*, a notification system called *Siena*[5], and several distributed so-called *Okeanos Lounges*. Note that *PaperbaseLD* does not have any database system at the back end tier in contrast to the architecture of *Paperbase*. The relational schema is stored together with its instances in the *LD*s' knowledge repository, as described in Section 3.2. Within *PaperbaseLD Respondeo* neither holds any application logic nor manages any documents. It solely serves as a gateway for interfacing to the Web and users respectively. *Siena* is used as a global notification middleware system where each *LD* publishes information and subscribes for document-related notifications. *Siena* uses the publish/subscribe communication paradigm as opposed to the rigid client-server request/response style. By deploying publish/subscribe, sender and receiver of notifications are decoupled from each other which leads in the case of *PaperbaseLD* to a loosely coupled coordination of all *LD*s. Finally, a *Lounge* is the abstraction used in the *Okeanos* framework for an agent environment hosting several mobile agents. Inter-connected *Lounges* in *Okeanos* allow agents to move to remote destinations directly.

Each document formerly stored in the document archive is – within *PaperbaseLD* – transformed into a *LD* which manages its knowledge repository

[5] For design and implementation details see the *Siena* home page at `http://www.cs.colorado.edu/~carzanig/siena/index.html`.

and raw data part. For illustration purposes in Figure 2B there are only three *Lounges* hosting 9 *LDs* altogether. If a user is requesting or searching for some *LDs* through the Web, it is up to each *LD* to respond to the request adequately.

In order to use *Living Documents* in a Web-like environment, each *LD* is able to generate dynamically an HTML view of its knowledge repository and document content. By this, we turned *LDs* into ordinary HTML pages. Furthermore, the generated HTML view depends on the actual content of and the query sent to the knowledge repository. Thus arbitrary HTML views of the same *LD* can be generated. By turning ordinary documents into *LDs* which are inter-linked with each other, a so-called *Living Hypertext* is established, as illustrated in Figure 1B. Generally, a *Living Hypertext* is a hypertext which consists of *LDs*.

Within *PaperbaseLD* each incoming request or notification is mediated through the computational services of a *LD*. Usually the handling of requests involves two different kinds of interactions between the services at the code part and the rest of a *LD*: First, the knowledge repository is contacted to determine if incoming requests can and should be handled. In addition, some accounting information is requested from the knowledge repository. Then, depending on the type of incoming request the services contact the raw data part for further and up-to-date information about the actual content of the document. For example, in order to perform a full-text search it is necessary not only to search for particular document state information stored in the knowledge repository, but also to search the content of the document itself.

In *PaperbaseLD* a request or search can be performed in two ways:

***LD* Compliant Searching.** An incoming search request is forwarded by the message bus of *Respondeo* to a designated *Lounge* which serves as an entry point into the network of *LDs*, as shown in Figure 2 (*Lounge* at *host3*). Basically any *Lounge* can play this kind of role. Then the request is turned into a *LD*, a so-called *LDSearch*. A *LDSearch* is a special kind of *LD* which only contains some processing logic and document state information about the type and content of the search request including the query itself. Then *LDSearch* interacts with its environment and dynamically determines available *Lounges* and their hosted *LDs* respectively. Generally, a *LDSearch* is a mobile *LD* which moves among the network of *LDs* and interacts with them locally. After the search process is completed, the results are returned to *Respondeo* and the user. After the search results have been returned, a *LDSearch* turns into an ordinary *LD* which just behaves as a regular *LD*. The hit list of documents retrieved by the original *LDSearch* are now part of the transformed *LD*'s raw data part. The uniform handling of search requests and ordinary *LDs* opens up several interesting possibilities in the area of information retrieval. Using *LDSearch* as a cached version of document hit lists can contribute to an improved distributed search performance within *PaperbaseLD*. Additionally, users or other *LDs* can make use of the knowledge about the search process contained in a *LDSearch*'s knowledge repository and document hit list. For performance reasons a search request can either be accomplished by a single *LDSearch* traveling around the network of

*LD*s or by creating a number of *LDSearch* clones to build a swarm to interact with remote *LD*s faster.

Cooperative Searching. By using the cooperative searching approach each micro server (*LD*) publishes information about its managed document to the notification system *Siena*. Each micro server also subscribes for notifications which are related to its managed document. As far as there is related information in *Siena* available, the interested micro servers are notified asynchronously according to the publish/subscribe communication paradigm used in *Siena*.

*LD*s publish primarily subsets of their document state information contained in the knowledge repository. For example, information about the type and description of the managed document is published to the notification system. Incoming user requests are handled by *Respondeo* which simply publishes user requests into *Siena*.

The cooperative search approach in *PaperbaseLD* only loosely couples *LD*s. Thus the content-based routing of notifications in *Siena* provides a cooperation mechanism which takes the independent and self-sufficient nature of micro servers adequately into account.

4 Topic Distillation with *Living Documents*

According to [13] "end users want to achieve their goals with a minimum of cognitive load and a maximum of enjoyment". Generally, searching for information which is stored in databases assumes that the user is familiar with and knows about the semantics of the respective database schema: What kinds of attributes exist? What are the relationships and dependencies between them ? The two search approaches described in Section 3.3 follow this kind of traditional, database-centered information retrieval approach. Note that even the cooperative search approach to *Living Documents* assumes knowledge about the underlying meta data scheme. In this section, we describe a third information retrieval approach to *Living Documents* which is used in the Web research community to search for relevant information on the Web - topic distillation. Topic distillation is the process of finding quality documents on a query topic [2]. It addresses the problem to distill a small number of high-quality documents that are most representative to a given broad topic. The goal in topic distillation is not to index, search or classify all the documents that are possibly relevant to a topic, but only the most authoritative information on the requested subject. In hypertext environments such as the web, a topic of a query can be found not only by analyzing the keywords of the query and the content of the retrieved documents, but also by taking the hypertext link structure into account.

Another key observation in large and dynamically changing hypertext environments is that (i) the preprocessing of documents including its content and link structure is not feasible because the number of documents might be too high. (ii) Furthermore, the hypertext structure might change continuously.

Within *PaperbaseLD* we created a hypertext of inter-linked *Living Documents*. Generally, there are two types of links contained in a *Living Document*: (i) A link in the raw data part is a so-called *original link*. For example, an ordinary web page which is turned into a *Living Document* keeps all its web links in its original HTML source. In addition (ii) there are so-called *meta links* between *Living Documents* which are stored as an entry in the *Living Document*'s knowledge repository. For example, each *Living Document* publishes its database mapping scheme (see Section 3.2) into *Siena*. Then all related *Living Documents* which share the same database attributes keep a meta link to each other in their knowledge repositories. Each link type is an additional source of information which can improve the retrieval process to find high-quality documents.

In the following we describe how to deploy topic distillation in a dynamic environment such as a hypertext of inter-linked *Living Documents* to find high-quality documents. Note, by turning digital documents into *Living Documents* the conceptual border between traditional database-centric information systems and the Web vanishes: A *Living Document* can play several roles depending on its current environment and context. In a Web-like environment it provides a dynamically generated HTML representation of its managed digital documents which makes it possible to deploy all the Web infrastructure such as search engines (e.g. Google, AltaVista). However, in a traditional environmental setting, as described in Section 3.1, it provides the same retrieval facilities as ordinary Client-Server systems.

The search engine *Smider* [14] was originally derived for topic distillation on the internet. It has been inspired mainly by three different ideas: First, it explores hypertext link structures to identify topic-related communities on the internet which is inspired by Kleinberg's work [11]. Second, the idea of selective spidering to answer a specific query [5]. Third, *Smider* incorporated the concept of focused crawling [4]: crawling the "best" web pages first. The focused approach is currently most prominently used by Google.

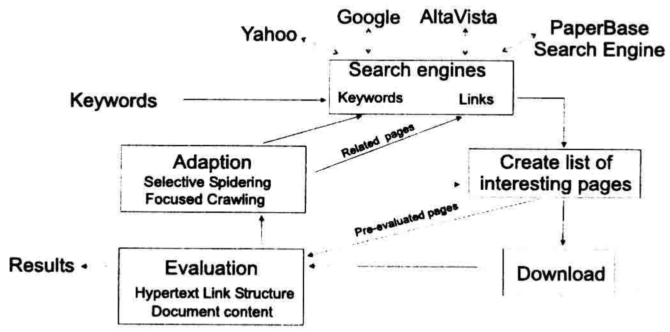

Fig. 3. Overview of *Smider*'s evaluation mechanism

4.1 *Smider* - The Principal Procedure

Smider is implemented in Java and is based on a multithreaded system architecture. While *Smider* is running it maintains a single priority queue of pages to be visited. The principal procedure of *Smider* to search for interesting documents related to given keywords, as depicted in Figure 3, is centered around an iterative evaluation process:

- *Smider* starts by queuing up few search engines as entry points to its search. This start set of documents contains either ordinary Web pages or *Living Documents* depending on the kind of search engine which has been contacted[6]. However, in our experiments we always used the *Paperbase* search engine as the primary search engine for *Living Documents*.
- Then it starts by spawning several spidering threads. From now on every spidering thread performs the following actions repeatedly: Get the most interesting page from the queue, retrieve it and feed links which have been found to *Smider*'s evaluation mechanism.
- After a certain number of documents (pages) have been retrieved, another thread starts which continuously evaluates all pages according to an algorithm which is an extension to Kleinbergs algorithm [11]. Our evaluation algorithm is described in detail in Section 4.2.

The concurrency of the evaluation and the spidering causes massive feedback within *Smider*: The actual priority of pages to be spidered is derived from their (preliminary) evaluation and of course the actual retrieval of pages changes the link structure which is known and changes the outcome of the next iteration of the evaluation process. This makes *Smider* fairly flexible, all kinds of *search engine pages* (e.g. ask Altavista for links, ask Google for next 10 links), *normal web pages* and *link resources* (e.g. Altavista $link : u$ searches) are all held within that one queue and compete for getting retrieved next. There is no fixed schedule and the search for different topics might retrieve pages of above three types in fairly different percentage and order.

4.2 An Extension of Kleinberg's Idea

It follows from the above that the evaluation method is very important for *Smider*. It does not only determine the final result but also the priorization of pages during search and therefore actually decides which pages are visited in which order.

The basic principle of Kleinberg's [11] algorithm is based on the matrix representation H of the vertices of the hypertext connection graph. Essentially one performs iterations for large eigenvalues of HH^T and H^TH to identify two special eigenvectors a and h of those matrices. Kleinberg demonstrates that the relative value of the entries of those eigenvectors yield good evaluation of the

[6] To be more precise, *Smider* uses the dynamically generated HTML representation of a *Living Document*.