Squirrel

Simple Query Interface for Ressource Retrieval in Electronic Libraries Submission to the 10th GLDV conference on Computational Linguistics, Leipzig, March 1997

Sergej Melnik, Timo Böhme, Karsten Böhm
{Melnik/boehme/boehm) @aix550.informatik.uni-leipzig.de

Abstract: *We describe a new WWW-based information system called Squirrel which demonstrates an attempt to integrate a relational database system (RDBS) with an information retrieval system (IRS) providing context based access to a SGML document collection. Indexing of the documents is performed on full-text basis whereas search capabilities include ranking of the retrieved documents. A functioning prototype has been implemented and is available on the Internet.*

Introduction

The problem

The Web represents today the information source. **In** some impetuously developing areas it is sometimes the only one. The vast amount of information available on the Web requires new methods of information retrieval. To tackle this problem there have been developed a number of search engines like Lycos [Lyc96], Alta Vista [Alt96] or InfoSeek [Inf96]. They index a considerable part of the documents on the Internet referencing them via hyper-links.

Although delivering surprising results in individual cases, these search engines exhibit severe drawbacks. Especially the lack of semantic differentiation leads often to problems. For example, a search for Java a year ago would have delivered more information about coffee and Asian islands than about a programming language. Today the situation is certainly quite the contrary.

Furthermore to follow a link is often the only way to find out what is behind it. This often leads to long 'coffee breaks' when a link points e.g. to an overseas site

and happens to contain only irrelevant information. On the other hand indexing the whole Web on a central place leads to inherent scaling problems. This is indicated e.g. by the fact that in spite of parallel computing this asynchronously created index gets fast outdated what makes a part of the document collection unreachable. This situation is criticised by many users as being time consuming and ineffective.

Our approach

To address the above mentioned weak points we have developed another approach independently of other related research work.

Regarding the problem mentioned above, we used a different idea of giving the user access to the information presented in a specific site. In our opinion the problem of finding relevant information cannot be solved by just building more 'clever' search engines, but in changing the way that are information presented.

The person which publishes documents on the Web has normally a good deal of understanding about its internal structure, semantics, context and its internal and external references. Therefore it is obvious that this person is privileged to organise the documents in a way which they can be accessed easily and supply a way to search quickly for information. This task - which is extremely difficult for machines – is currently also done by the search engines which lack all this information and produce only poor results.

Our aim is it to supply the necessary tools for creating, maintaining and accessing an Electronic Library which can be easily used and searched in a set of contexts. Furthermore the site should be presented to the user in a way that shows the semantical structure of the documents and their references between them.

Conventionally the document is the major information piece on the Internet. However, many large documents are sometimes highly structured covering distantly related topics. Retrieving such a document requires the user to perform another search, this time inside the document.

Obviously, the document structure needs to be taken into consideration. We specify the document structure using the Standard Generalised Mark-up Language (SGML). This allows us to mark-up e.g. document header and title as well as to subdivide documents into smaller semantically coherent units that we call *chunks.* These can be retrieved independently of the documents they belong to.

In order to provide efficient access to a document collection we store it as a repository in which chunks are indexed on the full-text basis building up a vector space. U sing the vector space model the Squirrel prototype provides semantic differentiation within the document collection organising chunks in a *context struc-*

ture. Thus in the example above the user can ensure the intended interpretation of 'Java' providing the context of programming languages. The context structure is implemented as a context tree projected onto the vector space. The context tree provides structured view on the repository enabling the user to localise relevant topics. Apart of this, the context tree serves for grouping search results according to the context structure. These results are ordered by relevance to the query employing ranking algorithms and presented to the user. After that user can retrieve on-the-fly formatted chunks with highlighting of the found keywords.

The graphical user interface is a special feature of the Squirrel system. It is used to facilitate the search and to visualise the search results. Written fully in Java [Fla96] - like all components of the system - the c1ient applet runs with any Java-enabled browser. The applet interacts with the server application which is responsible for performing SQL queries using JDBC [Jav96] protocol and converting outgoing documents into HTML format.

The remainder of this artic1e is organised as follows: in the next section we describe the conceptual basis of the Squirrel system, i.e. the context structure and related c1ustering mechanisms. Sections 3-6 deal with our prototype implementation. In section 3 we present a brief overview of the system. Section 4 contains the description of the underlying DBS. Section 5 is devoted to the document acquisition and storage. The description of the user interface can be found in section 6. Finally, the experiences gained while testing the system and planned extensions are stated in section 7.

2 Context tree and clustering

In order to ensure an acceptable recall e.g. the ratio of retrieved documents to all relevant documents) while seeking for a document in a repository, one must guarantee that no important keywords of any document have been missed during indexing. A natural way to do this is to index documents on the full-text basis. Unfortunately, providing an efficient means of search in the repository, full-text indexing alone does not solve the problem of semantic c1assification of the documents (or chunks in our case, i.e. parts of documents) found. At last, semantic differentiation is an important factor for increasing precision of the search (e.g. the ratio of relevant documents to all documents found).

To facilitate semantic differentiation of the terms (keywords) indexed in the repository we impose a context structure upon the document space. The aim to organise the document repository into a structured collection led us to using a *context tree* which groups document subspaces into hierarchically ordered topics or contexts. This hierarchy results from subdivision of individual topics focusing

broader contexts into narrower defined ones. One of the well known catalogues using this strategy is Yahoo [Yah96] which enjoys wide popularity with the Internet users. In contrast to Yahoo we do not rely on one strictly hierarchical constructed context, but allow a combination of several (possibly) unrelated contexts by the user to define the search-space more exactly.

In our implementation the context tree is composed of a number of context nodes each representing a particular topic. A parent — child relation on the node set defines the node hierarchy. As we mentioned above, each topic identifies a certain subspace in the vector space of the document repository. Thus, a node, defined by a vector (or set) of keywords, is located approximately 'in the middle' of the subspace it comprises. This is ensured using keywords with the highest average weigh for the chunks inhabiting a context node. We call these keywords unique keywords.

Due to its inherent nature, the notion of subspace or subtopic proposes using relative term vectors. Relative in the sense that it is sufficient to store only additional terms to distinguish sub-nodes within a given node, without replications of the super-node terms in every sub-node. The sum of relative vectors yields the corresponding absolute vector. The idea of document spaces and relative vectors is illustrated in figure 1.

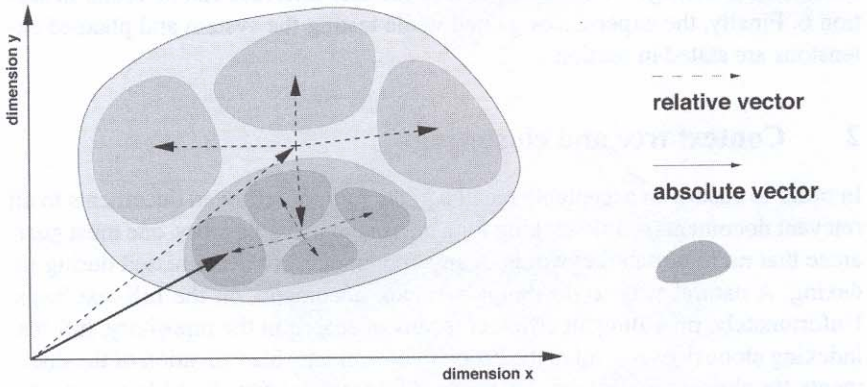

Figure 1: Schematic two-dimensional model of the document space.

Now that we have briefly described the main building parts of the context structure, we continue with how the context tree is constructed.

Intuitive navigating through the tree requires a high quality tree structure. Apparently, it is hardly possible to reach the ideal structure using fully automatic methods. On the other hand, manual maintenance of the context tree demands

high expenditures in time and human resources. We try to reach a compromise using semiautomatic procedures.

In our current implementation we employ a hierarchical agglomerative single-link clustering as well as heuristic clustering [SaI89]. Probably not the best, these methods are well known and relatively easy to implement. Hierarchical clustering differs from heuristic in that we can rely on the knowledge of all pairwise chunk-chunk similarities. Therefore the corresponding cluster generation is relatively expensive to perform. In return, hierarchical clustering produces a unique set of well-formed clusters for each set of data. We use this algorithm for initial context node partitioning provided adequate number of chunks in anode (please refer to section 5 for more detailed information on chunk processing). Apart of the control of the result quality, a human administrator is asked to name individual nodes. Heuristic clustering consists in finding one or more appropriate nodes to place a single chunk into. This technique requires manual placement confirmation.

3 System overview

In our opinion, one can better understand and evaluate the Squirrel prototype being familiar with its underpinning foundations. As we pointed out in the introduction, the main goal of the development of the system was fast and precise search. Concentrating our effort on this aim, we tried to pay an adequate attention to the issues of generalisation, accessibility and scalability. This gave birth to the conceptual framework of the system and eventually led to the current implementation. In following we describe the individual aims, their underlying concepts and the resulting implementation.

We considered the search being the key issue. The need to perform it *fast* requires indexing of the document repository. As mentioned earlier we use full-text indexing to improve the recall factor. Although we are aware that full-text indexing is already integrated in some database systems, we preferred to implement it on our own using a SQL [MS93] database. This leaves us the highest flexibility for experiments with different IR methods. The indexing is described further in section 5. To perform the search *precise* we followed the idea of context tree (see section 2) as well as ranking of search results. In our prototype ranking is
based on the relative term frequency in the document. During preprocessing of documents a numeric value identifying relevance of the given term for the corresponding chunk is generated and stored in the DB. This value is the basis of the calculation of the chunk relevance for the query.

Another important issue is generalisation. To provide format independence of the documents in the repository, we use SGML [Bry88]. The origin of SGML documents, converting documents from and into SGML and storing them in the Squirrel database is addressed c10ser in section 5. The interface to the DBS is based on SQL. Together with JDBC (Java Database Connectivity) [Jav96] they facilitate interoperability and provide the basis of working in heterogeneous environments. To meet the requirement of portability, we have chosen Java as our implementation language. Both server and c1ient side of the software are written in Java as well as DB administration and converting tools. Java enables us to run Squirrel servers (in future communicating one with another) across any Java-aware platform whereas the c1ient applet can access the server from any Java-enabled browser.

This leads us to the accessibility issue. From the very beginning the whole system has been designed for working on the Internet using the c1ient -server model. This inc1udes concurrent access from multiple users as well as hypertext support. With its multithreading functionality Java seems to be well suited for this kind of application. Besides that, in order to deliver hypertext (HTML documents) our server supports a small subset of HTTP (Hypertext Transfer Protocol) [HC96].

Apparently, maintaining a single repository one will inevitably encounter major scalability problems. A known solution represents distribution of resources. It is applicable to our case as transparent linking of context subtrees from different repositories into single context structure with parallel query processing. During the development of the prototype we steadily kept an eye on this perspective. For the time being, however, it is not integrated into the system framework.

Let us now take a look at how the goal and concepts mentioned above are implemented in the Squirrel prototype. The main components of the system are shown below in figure 2. In the remainder of this section we describe the functionality they embody as well as interaction of the system components.

The central part of the system is the Control unit which provides the interface to the database engine. Additionally it is responsible for tight IRS-RDBS coupling. Communication between RDBS and the control unit is carried out by means of SQL on top of the JDBC [Jav96] protocol, a standard SQL database access interface for Java. The control unit uses the Converter module which undertakes converting documents into different formats. Such operation can be requested while formatting outgoing HTML documents as well as by the Library manager which comprises a number of tools for administrating the document collection. It also semi-automatically preprocesses incoming documents before inserting them into the repository via the control unit.

The Client mediates in the interaction between the user and the Control unit over TCP/IP [Com95]. It forwards the query to the server, interprets received results and visualises them graphically in the frame of the context tree. The user interface of the client is described detailed in section 6.

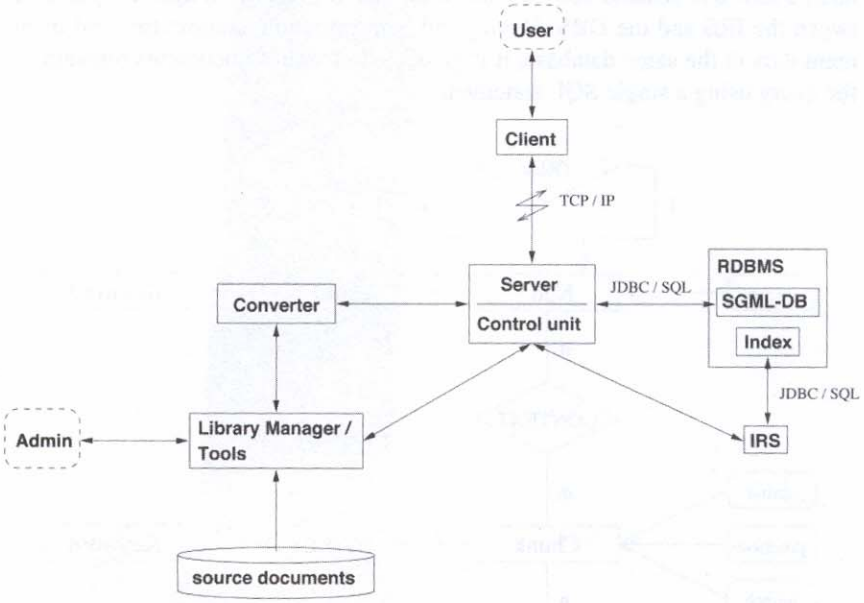

Figure 2: Components of theSquirrel system

Next section deals with the internals of the RDBS component which comprises the document repository and indexing engine.

4 Database structure

A SQL database builds the core of theSquirrel system. Its main tasks are maintaining the IR data and the document repository. To preserve applicability of the system we have done without proprietary DB solutions which already incorporate full-text indexing. For the same reason we have chosen JDBC. JDBC (Java Database Connectivity), like Microsoft's ODBC (Open Database Connectivity) [Gei95], is an interface definition for communication between database applications. All notable database companies have been contributing to this standard. It

is safe to assume that JDBC drivers will be available for all major DBS until the end of this year. We enlarge the spectrum of DB systems working with our prototype by using only a small subset of SQL.

The structure of the database is described in the following paragraphs. Below in figure 3 you can see the entity-relationship model representation of the database. From this schema can be concluded that we employ a tight coupling between the IRS and the DBS. Having index information, context tree and document data in the same database, it is possible to localise documents relevant for the query using a single SQL statement.

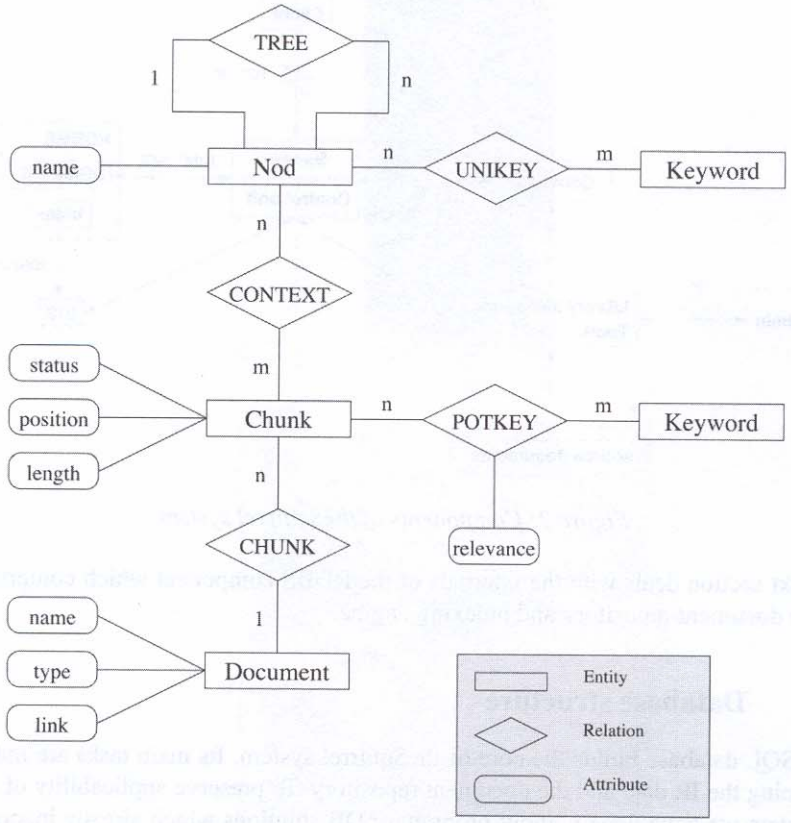

Figure 3: Entity-relationship schema of theSquirrel database

As mentioned earlier, chunks are the smallest independently retrievable document pieces. The document-chunk ownership is reflected in the relation chunk. To be able to retrieve the original or updated version of a document, the link

attribute of the document stores additionally the URL to the source file. Full-text index information for the IRS is also based on chunks. All keywords a chunk contains and their relevance within the chunk (for more information about the relevance calculation see section 5) are stored in the relation potkey. The relevance attribute is used by the ranking system which orders retrieved chunks by their importance according to the stated query.

Context tree structure is implemented as a parent-child relation (relation tree) on the nodes. Every node has a number of unique keywords (relation unikey) describing the place of the node in the vector space. The assignment of chunks to context nodes is stored in the relation context. As follows from the figure, a chunk can be a member of several nodes. In the next section we address the issues of data acquisition and filling the database.

5 Document acquisition and storage

This section presents the workflow of the Squirrel system from the viewpoint of the document life cyc1e. We begin with document origin, describe the chunk splitting process inc1uding converting to SGML and finally come to the stage where the stored chunks can be retrieved by the user. This process is illustrated in figure 4.

One aim of the Squirrel system is the implementation of an interface to information sources of different formats. We use the notion of *document* to describe a file containing structured text and self-contained objects like pictures etc. The objects can be stored and retrieved within the surrounding text but are not indexed by the IRS. Structured text ranges from plain ASCII text, formated text e.g. HTML, TeX) to database files. Currently we have implemented filters for a few document types. As an example of well-structured documents we used HOWTO manuals wide spread in the public domain sector. On the other hand, much up-to-date problem-solution oriented data is contained in news groups postings as well as mailing lists. Our prototype also supports these formats.

The life of a document found by a search agent or inserted into the database by the system administrator starts with converting it into SGML according to our DTD (document type definition). This is done by the general filter module which tries to choose the appropriate filter according to the document type and format. In this step the filter also splits the document into logical units (chunks) and marks them using SGML tags. In the case of structured documents these can be for example individual sections. Afterwards the position of every chunk within the document is stored in the database. We might add that all converting and filtering tools used in the Squirrel system have been developed by ourselves.

Once marked and saved, the chunks are indexed by adding all keywords (words which are not contained in a stop word list) to the database. For the ranking system the keywords are enriched by the relevance attribute which is calculated using the overall number of words in a chunk, the number of unique words, the absolute frequency and information about the position of the keywords (header or body of the chunk, to give an example).

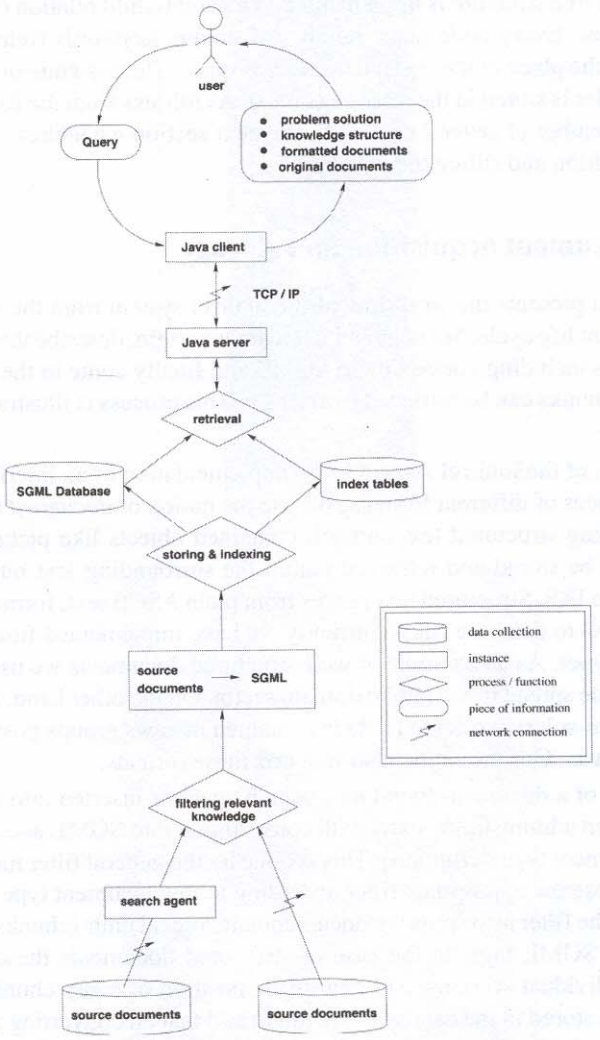

Figure 4: Workflow diagram of the Squirrel system.

At the last step the chunks have to be placed in related contexts. As pointed out in section 2 we make use of hierarchical and heuristic clustering. For this purpose we have developed two semiautomatic tools. Having reached this stage the chunks are ready to be searched for and retrieved by users interacting with the system using a client interface.

6 The user interface - access to the IR-system

The highest potential of Java for our prototype lies probably in the field of client-server applications. A Java application can open network connections directly to the database server without the slowing down CGI (Common Gateway Interface) whereas a Java applet can directly communicate with a mediating server. The latter fact allows us to solve the problem of stateful gateways in an elegant way. The Web is essentially stateless. To implement complete search sessions each individual CGI-style gateway must maintain the state of the requests made to each server. The solution lies in delegating the capturing of the state information to the client. Thus, the information about the chunks found in response to a user query is stored directly in the client applet enabling the user to refer to previous transactions without new server requests. Apart of the above mentioned drawbacks, CGI applications are hardly portable.

Basically we provide two major types of interface to our system: the Java-based frame displaying the context tree (the main interface) and the JavaScript interface which uses HTML forms. The latter is provided primarily for the users who are not equipped with Java-enabled browsers and is described in the full prototype documentation [MB96]. An example of the Java client window is shown in figure 5.

The window is divided into three parts. On the left-hand side you can see a graphical representation of the context structure of the document repository. As it can be very large and elaborated, only nodes found after a search or during navigating through the tree are displayed. On the right-hand side the search results (chunks) found in the selected node are displayed ordered by relevance and number of found terms. Selecting a chunk shows its content converted to HTML in the corresponding browser window. In the lower part of the interface window the user can type in a query. On default, the search terms are connected by the logical 'aR'. To mark a keyword as necessarily required one can put a plus sign in front of the keyword.

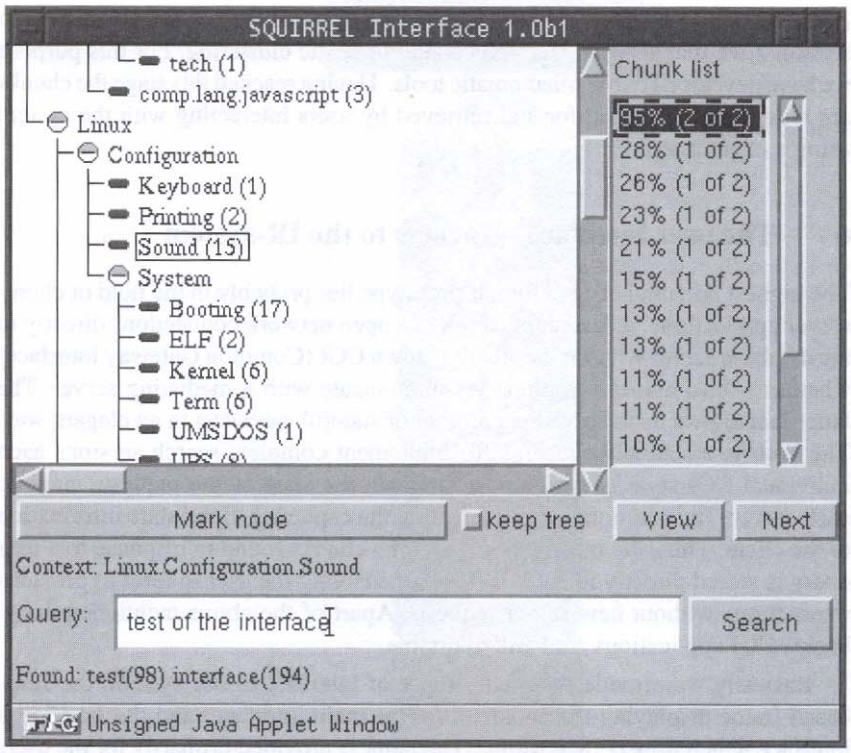

Figure 5: Example of theSquirrel interface window

In order to provide search context to the query, one can use the Mark/Unmark button. When some nodes are marked the search will be performed only in corresponding subtrees. As the number of nodes found in earlier stated queries grows, it may be convenient to enforce cleaning of the preceding results through deactivating the keep tree check box.

Stating a query is done in the following steps:

- Open the context tree and mark the relevant topics.
- Enter the query either as a number of keywords or as a sentence (stop-words will be automatically removed from the query).
- Disable the ‚keep tree'-check box if you are not interested in previous results.
- Click on the ‚search'-button to perform the query.

The context tree opens automatically all nodes containing relevant chunks to the query. The number of chunks found within anode is given in round brackets. Summarised information about the results are given in the status line beneath the query input line.

- By choosing anode from the context tree a list of found chunks ordered by relevance is displayed on the right-hand side of the window.
  Double c1icking a chunk from the list will retrieve it from the server which converts it to HTML, highlights the keywords from the query and inc1udes a link to the complete document. The chunk is displayed in the browser running the c1ient applet.

Conclusions

Experimenting with the prototype (using a small SGML database of just above 10MB) we gained useful experiences. First of all, the results of the queries stated delivered a good overview about the information contained in the document collection. Especially for the testbed of troubleshooting in the area of Linux- and Java-based systems it was helpful to use the system to review the Manuals, HOWTO's and NEWS-artic1es in an efficient way than to look for the desired information by browsing the documents on our own.

Using Java proved to be a right strategy. The system runs on *SunSPARCI* Solaris and Linux without any changes, even without recompiling the code. JDBC also seems to be a good choice. Through changing a couple of lines in theSquirrel configuration file the system switches to another DB engine the only requirement being availability of a JDBC driver. Sybase, Postgres95 and mSQL have been successfully tested.

The current version of theSquirrel system is still only a prototype. It does not employ thesauri, morphological decomposition or vocabulary normalisation. Further we plan to use distributed databases using request brokers, especially in the light of coming CORBA (Common Object Request Broker Architecture) support for Java [Mer96]. Currently the work is under way to give the Squirrel system a new profile. Primarily it is directed to processing of arbitrary SGML documents, that will be stored in accordance to their internal structure and not flat as it is the case now (except for chunk structure). Instead of chunks we think of the notion of 'dynamic sites' which are created on-the-fly and can contain multiple documents of different types. We see the advantage of this approach in that the hyperstructure of such 'site' can differ from the structure of underlying documents. This imposes certain requirements of the layout formatting component, especially for representing databases.

As a part of the new architecture we have already developed a new SGML processing engine. It was successfully used for creating this technical report which has been written in SGML and automatically converted to LaTeX [GM94] and HTML versions. The BibTeX-Bibliography has also been extracted automatically.

Started as an approach to address the problems of search engines, the direction of our efforts and thus the functionality of the Squirrel system has been continuously moving to the field of digital libraries. It was stimulating to discover similar ideas within the scope of DU (Digital Library Initiative) at the University of Illinois [SC96].

Acknowledgements

The authors are indebted to Prof. Erhard Rahm who encouraged us to write this paper and provided us with many helpful comments. The ERASMUS exchange programme gave us the opportunity to carry out the prototype implementation at the University of Kingston, Great Britain. We thank our ERASMUS supervisor Dr. Chris Hutchison.

Bibliography

[Alt96]
Alta Vista-Hornepage. http://www.altavista.digital.eorn, 1996.

[Bry88]
M. Bryan. *An author's guide to the Standard Generalized Markup Language.* Addison-Wesley, 1988.

[Corn95]
D.E. Corner. *Intemetworking with TCP/IP.* Prentiee Hall, 1995.

[Fla96]
David Flanagan. *Java in a Nutshell.* O'Reilly and Associates, Ine., 1996.

[Gei95]
Kyle Geier. *Inside ODBC.* Microsoft Press, 1995.

[GM94]
Michel Goossens and Frank Mittelbaeh. *Der LaTeX Begleiter.* Addison- Wesley, 1994.

[HC94]
M. Handley and J. Croweroft. *The World Wide Web - Beneath the Surf.* London: UCL Press, 1994.

[Inf96]
InfoSeek-Hornepage. http://www.infoseek.eorn, 1996.

[Jav96]
JavaSoft, Sun Microsystems, Inc. *The JDBC%o Database Aeeess API.* http://www.javasoft.comljdbc. 1996.

[Lyc96]
Lycos-Homepage. http://www.1ycos.com. 1996.

[MB 96]
Sergej Melnik and Timo Böhme. *SQUlRREL: why and how.* http://leipzig.informatik.uni-Ieipzig.de:8080, 1996.

[Mer96]
Bernhard Merkle. *Mokka für den Broker,* September 1996.

[MS93]
Jim Melton and Alan-R. Simon. *Understanding the new SQL: a eomplete guide.* Morgan Kaufmann, 1993.

[SaI89]
Gerard Salton. *Automatie text proeessing.* Addison-Wesley, 1989.

[SC96]
Bruce Schatz and Hsinchun Chen. Federating Diverse Collections of Scientific Literature, May 1996.

[Yah96]
Yahoo-Homepage. http://www.yahoo.com. 1996.