Proceedings of the Workshop on the Reuse of Web-based Information

Edited by
Anne-Marie Vercoustre,
Maria Milosavljevic and
Ross Wilkinson

Report Number CMIS 98-111
June 1998

CSIRO Mathematical
and Information Sciences
723 Swanston St. Carlton VIC3023
Telephone: (03) 9282 2610
Facsimile: (03) 9282 2600
www.cmis.csiro.au

Proceedings of the Workshop on Reuse of Web-based Information

Held in Conjunction with
The Seventh International World Wide Web Conference

Brisbane, Australia --- April 14, 1998

WELCOME TO THE GATHERING PLACE

Edited by:

Anne-Marie Vercoustre, INRIA, France, Anne-Marie.Vercoustre@inria.fr
Maria Milosavljevic, CSIRO-CMIS, Australia, Maria.Milosavljevic@cmis.csiro.au,
Ross Wilkinson, CSIRO-CMIS, Australia, Ross.Wilkinson@cmis.csiro.au

CSIRO Report Number CMIS 98-111
June 1998

URLs:
Reuse Workshop: www.mel.dit.csiro.au/~vercous/REUSE/WWW7-reuse.html
WWW7 Conference: www7.conf.au
CSIRO-CMIS: www.cmis.csiro.au
INRIA: www.inria.fr

Workshop Theme:

The workshop is focusing on the *reuse* of information that is currently delivered over the Web. By reuse, we mean the assembling of information from Web-based sources, and the restructuring of that information to either build a new Web application or to import it into a non-Web-based application, such as a database. The target research areas include (but are not limited to):

- mining, exploring and visualizing the Web,
- the structural extraction of relevant information, e.g., the Web as a large database,
- the semantic extraction of relevant information, e.g., the role of natural language processing and intelligent agents,
- the a posteriori restructuring or remodelling of Web pages and sites,
- the extraction of objects from the Web and their integration with legacy applications, or with other sources of information.
- the seamless integration of Web-based information, e.g., the design of information 'brokers'
- the development of descriptive models for Web information, and
- the role of XML in Web information reuse applications.

As the Web has grown the reuse of Web-based information has become increasingly important. Much of the existing research has focused on related issues such as 1) improving information search and retrieval capabilities, 2) enhancing the display of information in browsers, and 3) designing Web interfaces for legacy applications.

Advances in these areas have enabled a massive amount of information to be made available on the Web. The point is that people do not search for information just for the sake of it; they search for information because they need it and want to make use of it. The real challenge is to make the best use of that information since desired information is often distributed piecemeal among a number of sites, servers, and pages, each with its own disparate structure. The goal of reusing Web-based information is to extract and assemble existing information that a user is interested in and to deliver that information in a form that is directly usable by users or applications. Full automated access to Web servers makes for some fairly compelling applications, e.g. integration of corporate purchasing systems with e-commerce Web sites.

This workshop aims to draw together a number of research groups taking different approaches to the reuse of Web-based information, in order to promote the cross-fertilisation of ideas and highlight the prospects for future collaboration. The workshop will be a inter-disciplinary exploration into the reuse of Web-based information in controlled environment such as Intranets, as well as any part of the whole Web. Reuse of Web information in various contexts contributes to reducing the duplication of the information, while keeping it adapted to the real needs of the users.

Acknowledgment

We thank INRIA, France and CSIRO-Mathematics and Information Science, Australia, for supporting the chair and co-chairs in the organisation of this workshop, and the WWW7 Conference Committee for managing the registration and all the facilities.

CSIRO AUSTRALIA

INRIA

Program Committee:

- Thomas Baker, Asian Institute of Technology, Thailand
- Stephen Green, Microsoft Research Institute, Language Technology Group, Sydney, Australia.
- Curtis Dyreson, James Cook University, Townsville, Australia
- Craig Lindley, CSIRO-Mathematics and Information Science, Sydney, Australia.
- Udi Manber, University of Arizona, USA
- Giansalvatore Mecca, Universita' della Basilicata and Universita' di Roma TRE, Italia
- Philipp Merrick, webMethods, Fairfax, VA, USA
- Pr. Alberto O. Mendelzon, University of Toronto, Computer System Research Institute, Canada
- Maria Milosavljevic (co-chair), CSIRO-Mathematics and Information Science, Sydney, Australia.
- David Levy, Xerox PARC, Pao Alto, USA.
- Pr. Leon Sterling, University of Melbourne, Intelligent Agent Laboratory, Australia
- Ross Wilkinson (co-chair), RMIT and CSIRO-Mathematics and Information Science, Melbourne, Australia.
- Anne-Marie Vercoustre (chair), INRIA-Rocquencourt, France, (and CSIRO-Mathematics and Information Science, Australia)

List of Participants

- **Lisa Casey**, City of Sydney, Town Hall House, Sydney, Australia, lcasey@cityofsydney.nsw.gov.au,
- **Kerry Cody**, Department of Primary Industries, Brisbane, Australia, codyk@dpi.qld.gov.au,
- **Joseph Davis**, University of Wollongong, Australia, joseph_davis@uow.edu.au, (accepted paper)
- **Petros A. Demetriades**, King's College London, UK, petros@dcs.kcl.ac.uk (accepted paper),
- **Curtis Dyreson**, James Cook University, Australia, curtis@cs.jcu.edu.au, **PC member**, (accepted paper),
- **Carlos F. Enguix**, University of Wollongong, Australia, cfe01@wumpus.uow.edu.au (paper accepted),
- **Brendan Hills**, CSIRO-CMIS, Sydney, Brendan.Hills@cmis.csiro.au,
- **Tanja Joerding**, Institute for Softwaretechnology, Dresdes, Germany, tj4@inf.tu-dresden.de,
- **Ralph Lawson-Smith**, Radiology Westmead Hospital, Westmead NSW, Australia, irls@imag.wsahs.nsw.gov.au,
- **Richard Lin**, Monash University, Australia, rlin@indy10.cs.monash.edu.au, (accepted paper),
- **Craig Lindley**, CSIRO-CMIS, Sydney, Australia, Craig.Lindley@cmis.csiro.au, **PC member**, (accepted paper),
- **Kim Marriott**, Monash University, Victoria, Australia, marriott@cs.monash.edu.au (accepted paper),
- **Stef Miladinova**, Rochedale South, Qld, Australia, smiladin@writeme.com,
- **Maria Milosavljevic**, CSIRO, Sydney, Australia, Maria.Milosavljevic@cmis.CSIRO.AU, **(Co-Chair)**,
- **Matthew Montebello**, University of Wales, Cardiff, U.K., m.montebello@cs.cf.ac.uk, (accepted paper),
- **Ms. Hiromi Morikawa**, JSRD, Shinjuku-ku, Tokyo, hmorikawa@jsrd.or.jp,
- **Max Mühlhäuser**, University of Linz, Austria, max@tk.uni-linz.ac.at (accepted paper),
- **Judith Nicholls**, Fermilab, Batavia, IL, U.S.A, nicholls@fnal.gov
- **Kee Ong**, NetFranchise, Burlingame, California, USA, keeong@pacbell.net
- **Meg O'Reilly**, Southern Cross University, Lismore, Australia, moreilly@scu.edu.au,
- **Nadine Ozkan**, CSIRO-CMIS, Sydney, nadine.ozkan@cmis.csiro.au,
- **Anand Rajaramon**, Junglee Corp., Sunnyvale, CA, USA, Anand@junglee.com (accepted paper)
- **Joanna Richardson**, Bond University, QLD, Australia, richardj@bond.edu.au,
- **Francois Rouaix**, INRIA, France, Francois.Rouaix@inria.fr,
- **Philipp Ruetsche**, Web Office ETHZ, Zurich, Switzerland, ruetsche@sl.ethz.ch
- **Simon Ryan**, Telstra Research Laboratories, Clayton, VIC, Australia, s.ryan@trl.oz.au,
- **Bill Simpson-Young**, CSIRO, Sydney, Australia, Bill.Simpson-Young@cmis.csiro.au
- **Jim Stinger**, Hewlett-Packard, Palo Alto, USA, stinger@hpl.hp.com (accepted paper),
- **Martin Ström**, LinnéData AB, Sweden, martin.strom@lig.linnedata.se,
- **Anne-Marie Vercoustre**, INRIA, France, Anne-Marie.Vercoustre@inria.fr, **(Chair)**,
- **Ross Wilkinson**, CSIRO, Melbourne, Australia, Ross.Wilkinson@cmis.csiro.au, **(Co-chair)**

Workshop Program:

Tuesday, 14th April

9:00-9:15	**Welcome and Introduction** Anne-Marie Vercoustre
9:15-10:40	**Session 1: Searching for Information** Moderator: Ross Wilkinson *A personal Evolvable Advisor for WWW Knowledge base Systems* M. Montebello, W.A. Gray, S.Hurley *A Jumping Spider: Restructuring the WWW Graph to Index Concepts that Span Pages* Curtis E. Dyreson *Transforming the Internet into a Database* Anand Rajaraman
10:40-11:10	*Break*
11:10-12:10	**Session 2: Extracting Information** Moderator: Anne-Marie Vercoustre *Component Advisor: A Tool for Automatically Extracting Electronic Component Data from Web Datasheets* Malu Castellanos, Qiming Chen, Umesh Dayal, Meichun Hsu, Mike Lemon, Polly Siegel, Jim Stinger *Database Querying on the World Wide Web: UniGuide- An Object-Relational Search Engine for Australian Universities* Carlos F. Enguix, Joseph G. Davis, and Aditya K. Ghose
12:10-13:30	*Lunch*
13:30-14:30	**Session 3: Invited Talk, by Philipp Merrick, webMethods, U.S.A** . *Replaced at the last minute by another talk:* XML: The Key for Reuse?, Anne-Marie Vercoustre
14:30-15:30	**Session 4: Modelling Information** Moderator: Curtis Dyreson *The FRAMES Project: Reuse of Video Information using the World Wide Web* C. A. Lindley, B. Simpson-Young, and U. Srinivasan *Typing Concepts for the Web as a Basis for Re-use* Max Mühlhäuser, Ralf Hauber, Theodorich Kopetzky
15:30-16:00	*Break*
16:00-17:20	**Session 5: Presenting Information** Moderator: Maria Milosavljevic *An Architecture for Web Visualisation Systems* Petros A. Demetriades and Alexandra Poulovassilis *The Role of Reactive Typography in the Design of Flexible Hypertext Documents* Rameshsharma Ramloll *Using Constraints for Flexible Document Layout* Alan Borning, Richard Lin, Kim Marriott and Peter Stuckey
17:20-17:30	**Closing** Moderator: Craig Lindley

Report on the workshop

The workshop attracted about 25 people, mostly from Australia but also from the USA, UK, Austria, Germany and France. It was combined with the **workshop on Flexible Hypertext**.

The workshop included four main sessions, an invited talk and a concluding session. The first three sessions had a theme corresponding to the main steps in reusing information, namely:

- Searching and finding information
- Extracting information
- Combining/Presenting information

while a forth theme, *modeling Information*, was regarded as central to the whole issue of reuse.

The steps involved in the process of information reuse were highlighted in Anne-Marie's introduction. She also pointed out that extracting information may amount to following links among pages to find the relevant fragments or further links. Therefore searching and extracting are tightly related and the allocation of papers to one of the other session was somehow arbitrary.

Modeling information appeared to be a condition for extracting and reusing. The two papers in this session considered a priori modeling for supporting future reuse, rather than using a model for discovering and extracting information, like in some other papers (eg. A Jumping Spider and papers in session 2).

The last session focused on the flexible presentation of information, which is another important aspect in the reuse of existing information.

Unfortunately, for personal and professional reasons, Philipp Merrick from webMethods was unable to attend the workshop and the WWW7 Conference. At the last minute his presentation on using XML for e-business was replaced by an introduction to XML and its potential benefit for reuse, by Anne-Marie Vercoustre. Her presentation focused on the research perspective rather than the business perspective on XML.

After drawing some conclusions we had an unplanned wrap-up session with the Workshop on Hypertext Functionality and the WWW, co-chaired by Carolyn Watters and Fabio Vitali. This also resulted on a BOF session on Virtual Documents, and the formation of a special interest group on this topic.

Finding/Searching Information

In order to reuse information, we first require access to that information, either by: finding out what is relevant (by searching); accessing well-known places where information is regularly updated; being notified that new information has come; or a combination of the above.

Matthieu Montebello underlined the inefficiency of search engines and browsing regarding precision and recall, and arose the issue of such evaluation criteria in a world-wide environment. He focused on meeting user's need and interest and proposed a system that prompts the user with new URLs selected by peers, according to a dynamically evolving user's profile.

Curtis Dyreson made the distinction between *static* and *dynamic* approaches for search tools. A static search is done at periodic intervals, off-line, for example by a robot, and results are stored locally for further querying and reuse. A dynamic search occurs when a tool crawls part of the Web on-line for extracting information each time the information is requested. Curtis described a static system to index concepts that span more than one page.

Anand Rajaman presented the architecture of a commercial system developed by Junglee and used for integrating book offers from various electronic bookshops, job opportunities, etc. It was really "reuse at work". The system includes a tool box for building wrappers on sites, linguistic feature extraction, as well as a data verifier that checks the conformance of data type to expectations, in case the server's structure had changed without notification.

General discussion

A first question was: is searching for reuse different than usual searching?
One partial answer is that searching for reuse is in the context of a specific reuse task, rather than the user's task.
We had a long discussion on evaluation of search and reuse in the context of the Web, compared to TREC evaluation.
The problems include:
- the validation of data: TREC evaluation works on a well-established and fixed corpus. For web reuse there is a need for testing the validity (stability) of data for each wrapper (agent), regarding both the structure and the content.
- the nature of the web, where there may be a tension between "relevance" for a customer and "attractiveness" for the vendor. So the question "what is relevance" or "meaning" (eg. for images) is yet more complex than for classical IR.

Evaluation should:
- be user-based
- set the task involved in reuse (relevance for reuse)
- evaluate the efficiency (how much reuse is possible?), regarding the instability of data.

Extracting Information

Jim Stinger presented a tool for automatically extracting electronic component data from web datasheets. His system does not rely on the data format (tags) for extracting information. Instead it uses domain knowledge about products, and vendor specific catalogues which model the knowledge embedded in particular vendors' datasheets. The navigator visits vendors' pages, traverses along promising links or fills forms as required, and returns a list of pages to the extractor. The extractor iteratively processes the pages for each characteristic to be mined from the datasheet and sends them to the database. The difficulties encountered in automatic mining were many due to poorly written HTML pages and inconsistency between datasheets versions.
The second paper, presented by Joseph Davis, also uses a domain centered model to support more precise access to information. The approach here is quite different though, since it uses complex metadata based on ontologies. The success of the approach is dependent on a given number of sites relevant to the domain adopting the use of the UniGuide Scheme metadata and to attach these metadata to their pages. This approach is in the line of current initiatives for metadata development, like the Dublin Core initiative, or the W3C working group on RDF schema registery.
The two presentations highlighted the need for making the underlying model of the application explicit, since it is often difficult to infer it from the structure of the site(s).

Modeling Information

The 2 papers in this session are concerned with modeling information for supporting better reuse, the first one for reusing digital videos, the second for reusing Web styles and instances.
Craig Lindley introduced FRAMES, a system for reusing digital videos through synthesised new videos. FRAMES uses a rich model for describing videos which reflects the complexity of video semantics (multiple possible interpretations) as well as physical features. Although the query engine and content models can be used directly for search and retrieval of video content, the important point, with regard to reuse of the content, is that it can also be used for a wide variety of other purposes, for example, virtual video generation. A video synthesis can be specified by a virtual document prescription which contains queries to the video database using a combination of features available in the model.
Max Mühlhäuser focussed on hyperwebs, ie. a logically correlated set of nodes and links. He showed how *typing concepts* for hyperwebs can improve finding information as well as reusing such hyperwebs in augmented hypermedia. He introduced graph grammars as a way of typing the Web in the programming language sense. With this formalism, models and instances of hyperwebs can be precisely described and reused. This work is strongly rooted in hypertext/hypermedia research and deserved to find a better place in the Web community especially for intranet design.
Both presentations demonstrated the importance of a rich description of the primary sources (content, structure) in order to support effective reuse.

Presenting Information

We did not have specific papers on reassembling/reconstructing information, although several of the participants are working on Virtual Documents, which were discussed in a separate BOF session (see below). Virtual documents were also discussed by Craig Lindley in his presentation on building video synthesis.

Petros Demetriades presented an architecture for building various views on Web Information, as well as on browsing history. Views perform the actual interpretation of the data into some visual form. This architecture can accommodate both static and dynamic search, and facilitate experimentation and side-by-side comparison of different collection, storage and visualisation methods and techniques.

Kim Mariott showed the advantages of using constraint-based techniques for the automatic generation of good document layout. Documents and Web pages are made of content, structure and layout. Both structure and layout can change according to reuse content and context. Layout should adapt itself to a wide variety of environments, medias and viewers. Constraint-based layout provides this flexibility and supports a real *negotiation* between the author and the viewer for controlling the final appearance of the document.

Some Conclusions

Conclusions drawn by Craig Lindley were:

1. clarifying the notion of reuse: making a distinction between "use" of information, "reuse", and multi-purpose information. Reuse often amounts to restructuring or adding-value to primary sources (second level information), or use it for other purposes than the one initially intended. Multi-purpose information refers to flexible information delivery to accommodate various contexts. Reuse here is anticipated.
2. One particularly interesting issue was that the context of "use" is not necessarily the "user"

context, and these should be treated separately. There is a need for evaluation of search and reuse relevance and effectiveness, according to a context or a task. Relevance regarding meaning is difficult, especially for images which may have several *meanings*. Relevance can be quite different for a vendor, or a customer, and for entertainment.
3. The conclusions also highlighted the need for research on meta models for structure and semantics and a language for virtual inclusion/transclusion.

Wrapping session

Another workshop which was running concurrently dealt with the hypertext functionality of the web and where we might like the web to be enhanced. For example, in transclusion, instead of including a link to a piece of information, that information might be included in the document on-the-fly in order to achieve some purpose. This is like the traditional "stretch-text" idea where pieces of additional text are weaved into an otherwise static document for some reason, such as the user not understanding a particular term etc. We invited the members of this workshop to join us for the final wrap-up session and a member from each workshop gave a summary of the issues which had arisen over the course of the day.
Requirements for the Web (from this point of view):

1. The Web needs to enable typed links. Examples of types of links are: annotations, transclusion (inclusion by reference), personalized links, trails and guided tours, backtracking and history-based navigation etc. Currently, the Web only offers one type of link. For the hypertext community, this is insufficient.
2. Web applications need to be evaluated more rigorously. Evaluation is likely to become a big theme, as no one does it (or does it well) and as the need is clearly there.
3. The Web needs to provide facilities that support collaboration.
4. More user-centered approaches to web design are needed.

This allowed us to see the links (pardon the pun) between the two groups more clearly and led to the formation of a "Birds of a Feather" session later in the conference.

This BoF session focussed on "virtual documents" in order to discuss at some length some of the issues we consider important in building systems which can produce documents on-the-fly from either large pieces of text or smaller fragments using a grammar etc. We spent some time discussing the meaning of the term "virtual document" but didn’t reach a conclusion in one hour.

Further actions

Participants to the BoF session agreed on further collaboration about Virtual Documents, starting with be setting up a mailing list and a web page for those people who are working in this area in order to pool ideas and constraints, and to facilitate collaboration. For more information please to contact Brendan Hills: Brendan.Hills@cmis.csiro.au

A Jumping Spider: Restructuring the WWW Graph to Index Concepts that Span Pages

Curtis E. Dyreson
Department of Computer Science
James Cook University
Townsville, QLD 4811
Australia

Abstract:

A search engine can index a concept that appears entirely on a *single* page. But concepts can span *several* pages. For instance, a page on *trees* may be linked to a page on *lecture notes* for a *data structures* course. If the *trees* page does not specifically mention *lecture notes*, then a search engine search for *lecture notes on trees* will, at best, only partially match each page.

In this paper we describe a practical system, called a Jumping Spider, to index concepts that span more than one page. We assume that a multi-page concept is created by a *concept path*, consisting of some number of hyperlinks, that transits through pages with specific content. For instance, there must be a concept path from the *lecture notes* page to the *trees* page to create the *lecture notes on trees* concept. The concept paths must be relatively few (certainly much fewer than the overall number of paths in the WWW) or the cost of the index will be too great. At the same time, the paths must be easily identified, so that they are capable of being automatically computed and indexed quickly. Finally, the paths must be viable, in the sense that they really do connect multi-page concepts.

The Jumping Spider restructures the WWW graph (within the index) to create a graph consisting only of concept paths. The restructuring only permits paths from from pages in a parent directory to any transitively connected page in a child directory, or over a single link that connects two unrelated directories.

Motivation

Sally is teaching a course in data structures. She is currently lecturing about trees and is interested in how the material is taught elsewhere. Sally decides to look on the WWW to find the relevant information. There are two kinds of tools that Sally can use: search engines and WWW query languages.

Search engines

Search engines are the *de facto* standard for finding information on the WWW. While there are a great variety of search engines they all share the same basic architecture. A search engine consists of three components: a robot, an index, and a user-interface. The robot periodically traverses the WWW. For each page that it finds, the robot extracts the content for that page, usually a set of keywords. The keywords are used to update the index, which is a mapping from keywords to URLs. A user searches by inputting one or more keywords into the user-interface. The user interface consults the index and retrieves the URLs of pages that best match the given keywords. The result are often ranked by how well the keywords match. Search engines differ primarily in how content is extracted, how much of the WWW they search, and how keywords are matched and results ranked.

An important feature of a search engine is that the architecture is optimized to respond quickly to user requests. In particular, the user only interacts with the index. The WWW search, which is performed by the robot, is entirely separate. This architecture recognizes that searching the WWW is slow, and that robots impose an overhead on network bandwidth. We will call an architecture where the WWW search is distinct from a user query a *static* architecture.

The primary drawback to search engines (for our purposes) is that they only index single pages. Sally wishes to find the *lecture notes* on *trees* for a *data structures* course. In most courses, each lecture has its own page, separate from both a lecture notes page and course home page. Since the search terms do not appear together on a *single* page it is unlikely that the notes could be located by using a combination of *all* of the above descriptions (e.g., *trees* AND *lecture notes* AND *data structures*). On the other hand, using just a few of the terms (e.g., just *trees*) would find too many irrelevant URLs, and fail to isolate the notes within a data structures course.

In order to find the appropriate notes using a search engine, Sally would have to employ a two-pronged strategy. First she would use a search engine to find data structures courses. Then she would manually explore each course to find lecture notes on trees. Sally is unhappy with this strategy since it has a manual component. But she can use a *WWW query language* to automate the exploration of each course.

WWW query languages

The database research community has recently developed several languages to overcome search engine limitations. Languages such as WebSQL [MMM96, AMM97, MMM97], W3QL [KS95], and WebLog [LSS96] combine *content predicates* to search a page with *path expressions* to traverse links between pages. A content predicate tests whether a page or link to a page has some specified content. A path expression is a description of the path that connects pages.

Sally chooses to use WebSQL to find the notes. She decides on the following page navigation strategy. Starting with pages on *data structures*, navigate to pages which mention *lecture notes* that are within three links, and from those pages, navigate at most two links to find pages on *trees*. A WebSQL query for this navigation strategy is given in Figure 1.

```
SELECT notes.url
FROM
    DOCUMENT dataStructures
        SUCH THAT dataStructures MENTIONS 'data structures',
    DOCUMENT lectureNotes
        SUCH THAT dataStructures → | → → | → → → lectureNotes
    DOCUMENT trees
        SUCH THAT lectureNotes → | → → trees
WHERE lectureNotes CONTAINS 'lecture notes' AND trees CONTAINS 'trees';
```

Figure 1: An example WebSQL query

The **FROM** clause stipulates that the Uniform Resource Locators (URLs) associated with ‘data structures’ are retrieved from a search engine (by using a **MENTIONS** to build the **DOCUMENT** *dataStructures*). From this set of pages, pages that are within three (*local*) links are visited as specified by the path constraint ‘ $\rightarrow | \rightarrow\rightarrow | \rightarrow\rightarrow\rightarrow$ ’. The **WHERE** clause adds (with a **CONTAINS** predicate) the constraint that for each page visited, navigation should continue only from those that contain the string ‘lecture notes’. Finally from those pages, two links can be followed to reach the pages on ‘trees’.

Sally is making a fundamental assumption that pages are usually linked to conceptually-related information. For instance, a user will likely assume that traversing one link from a page on *data structures* will access information somehow related to *data structures*. So the graph of (hyper)links is a (very) rough approximation of the conceptual relationships between pages. This assumption is sometimes false since there are usually some links on a page that do not obey this rule, such as a link to a page maintainer’s home page or a link back to a central entry page.

Sally must also guess the maximum number of links that can separate related concepts. She guesses at most three links between one pair and at most two links between another. Sally may not realize that it is very important for her to be as *conservative* as possible in her guesses since the number of links she uses in a path expression has a huge impact on the performance of her query. The query explores $\boldsymbol{O}(k^n)$ links in the graph, where k is the number of links from a page and n is the number of links in the path expression; in other words the size of the graph explored is exponential in the number of links in the path expression. At the same time, Sally should not be too conservative since she might miss the desired information.

For the data structures notes at James Cook University Sally has guessed wrong since the lecture notes for all of the courses taught by the department are within three links of the data structures home page; it is one link back to a list of courses, then one link to a course home page, and finally, one to the lecture notes for that course. Paths of length three also exist from the data structures course through lecturers’ home pages to notes in other courses.

In general, in order to direct and constrain the navigation in a WWW query using path expressions Sally needs to have some a priori knowledge of the topology of the WWW in the region of desired content. In this case she has to know roughly how many links exist between pieces of information. Unfortunately, content is only loosely related to the graph topology of the WWW and although the user will likely be perfectly capable of specifying the content, the user will usually not have a priori knowledge of the specific topology (in the terminology of Lacroix et. al [LSCS97], the user has ‘‘zero knowledge’’ of the topology).

One other limitation is that current implementations of WWW query languages do not separate the search phase from the query. We will call architectures that search the WWW during a query *dynamic*. One advantage of dynamic architectures is that they do not become dated since the current version of the WWW is used during the search. But dynamic architectures have a high query evaluation cost since the WWW is explored during a query.

Hybrid systems - a Jumping Spider

In this paper we describe a hybrid system, called a Jumping Spider, that combines search engine and WWW query language features. Our position is that WWW query languages offer too *powerful* a mechanism for searching the WWW while search engines are too *weak*. The Jumping Spider is a desirable medium between the two approaches. Like WWW query languages, a Jumping Spider uses paths in the WWW to find concepts that span pages. But there are three key differences between a Jumping Spider and a WWW query language.

1. The Jumping Spider uses its own links between pages rather than the hyperlinks in a WWW graph. The Jumping Spider adds some links and removes others from the WWW graph to enhance and direct the search for concept paths.
2. The Jumping Spider has only one kind of path expression, which is a single link between concepts (although each Jumping Spider link may span any number of hyperlinks). We sacrifice the power of general path expressions in WWW query languages to attain reasonable query evaluation times.
3. The Jumping Spider has a static architecture like a search engine. The WWW search is done by a robot, off-line, entirely separate from a query. We believe that the user is more interested in fast query evaluation than up-to-date WWW graphs.

In the next section we describe how to build a *concept graph* that stores each link in a concept path. The Jumping Spider has been implemented and an example is presented. Finally we discuss related work and present a summary of this paper.

The concept graph

The motivating query presented in Section 1 is a general kind of query, which we will call a *concept refinement* query. In this kind of query a large concept space is identified initially. The concept space is then refined or ‘‘narrowed’’ in subsequent steps. In the motivating query Sally first requested information on data structures courses. Then that information space was narrowed to lecture notes, and finally to trees.

What this suggests is that a concept path must similarly refine or narrow the concept space. The WWW graph, however, does not have any such narrowing. Links can lead anywhere, they do not naturally lead from more general to more specialized information. Instead, a narrowing graph structure must be defined on top of the underlying WWW graph. In the remainder of this section we present an easily computable structure which is used by the Jumping Spider.

An observation about URLs

The structure we propose is based on URLs. A URL for a page pinpoints the page’s *disk location.* For example, the URL

```
http://www.cs.jcu.edu.au/ftp/web/Subjects/Informal.html
```

specifies that the page is in the `ftp/web/Subjects` directory relative to the server’s root directory.

Because the URL is related to the hierarchy of directories on a disk, it offers a primitive mechanism by which it is possible to make a initial guess as to which information is general and which is specialized. A general rule of thumb is that information in subdirectories is more specialized with respect to the information in parent directories. This is the common ‘‘cultural’’ convention for dividing information between parent directories and subdirectories in both WWW and non-WWW information repositories. So for instance, we might infer that the information in

```
http://www.cs.jcu.edu.au/ftp/web/Subjects/index.html
```

is ‘‘more general’’ than that in

```
http://www.cs.jcu.edu.au/ftp/web/Subjects/cp2001/index.html
```

since the former resides in a parent directory of the latter. We should note that we are *not* stating that the URL itself be used to infer the content of a page, since the name of the directory is often entirely unrelated to the content; few would know that cp2001 means a data structures course. Rather we are suggesting that some general/specialized relationship holds between the pages, at least, such a relationship is assumed by whomever maintains the pages. A page in a subdirectory contains information (whatever it may be) that is more specialized than that contained in a page in the parent directory (whatever that may be). Certainly, it will not always be the case that this is so, but it is a useful, cheap, and simple heuristic. We note that a similar heuristic is used by WebGlimpse [MSB97] (extends glimpseHTTP’s indexing of a directory) and also by Infoseek [Inf] (in grouping URLs with common prefixes).

Kinds of links

This assumption gives us enough information to distinguish different *kinds* of links.

- *back link* - A back link is a link to a page that is more general, that is, to a page in a parent directory.
- *side link* - A side link is a link to a page in an unrelated directory which is neither a parent directory nor a subdirectory, e.g., a link to a page on a different server, or from a page in the server’s root directory to one in a user’s `public_html` directory.
- *down link* - A link from a parent to a subdirectory is called a *down link*. A down link leads from a more general to a more specialized page.

This classification is incomplete since links between pages within the same directory have yet to be classified. Intra-directory links are classified according to the following rules.

- A link from an *index.html* (we use *index.html* in this discussion as the default name, the actual page name depends on the server, we mean the page that is loaded by default when the directory is used as the URL) page to a page in the same directory is a down link.
- A link leading to an *index.html* is a back link.
- For other links within the same directory, it depends upon whether there is an incoming down or side link to a page from some other directory. If so, then the page can be reached from outside the directory and links from that page to pages in the same directory are treated as down links. If not, then the links are treated as side links.

Figure 2 shows a part of the WWW graph in the region of the lecture notes on trees. Each node in the figure is labeled with a single content description, which we assume is determined by some search engine. In general each node will have many such content descriptions, but for brevity we show only a single description for each node. Figure 3 shows how the links in this region are classified. In the figure, back links have been eliminated. Side links, such as the one to the lecturer are shown with dashed lines. Down links are shown with solid lines.

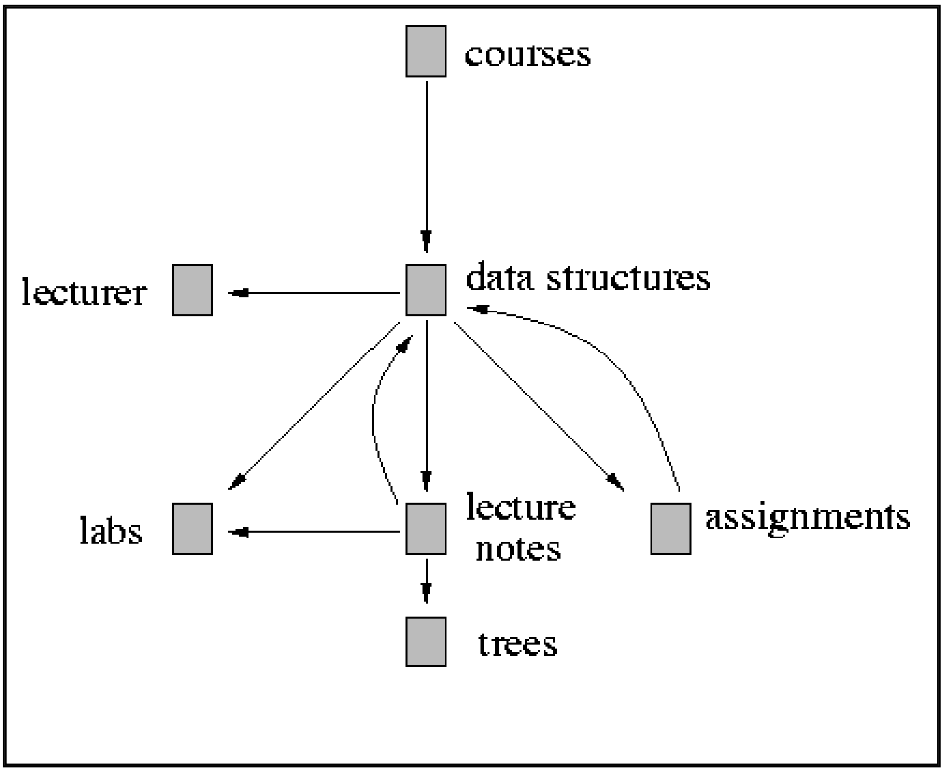


Figure 2: Part of the WWW graph in the region of the lecture notes on trees, each node label is a possible content description of the node (as might be determined by a search engine)

Building the concept graph

Having identified each kind of link we construct a *concept graph* from the WWW graph using the following rules.

1. Each link in the WWW graph is added to the concept graph *except* back links. In general, we believe that back links should not be traversed. They usually represent a link back to a central entry page, e.g., back to the server root page, and consequently are counter to the narrowing needed in a content-refinement query.
2. A link is added to the concept graph for each path consisting of any number of down links followed by at most one side link. Paths that consist entirely of down links have the best chance, in general, of leading to more refined content. A single side link is allowed to terminate such a path.

The concept graph constructed from the links classified in Figure 3 is shown in Figure 4.

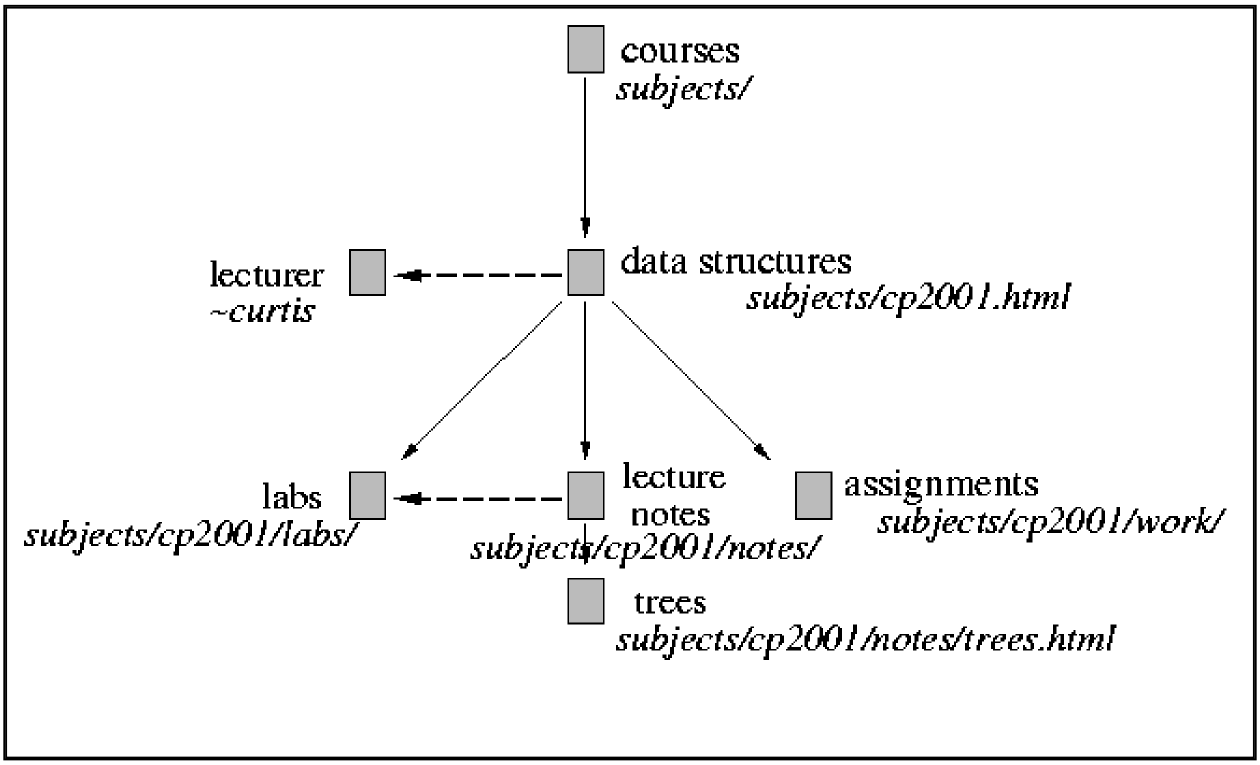


Figure 3: The links classified, solid lines are down links, dashed lines are side links (back links have been eliminated)

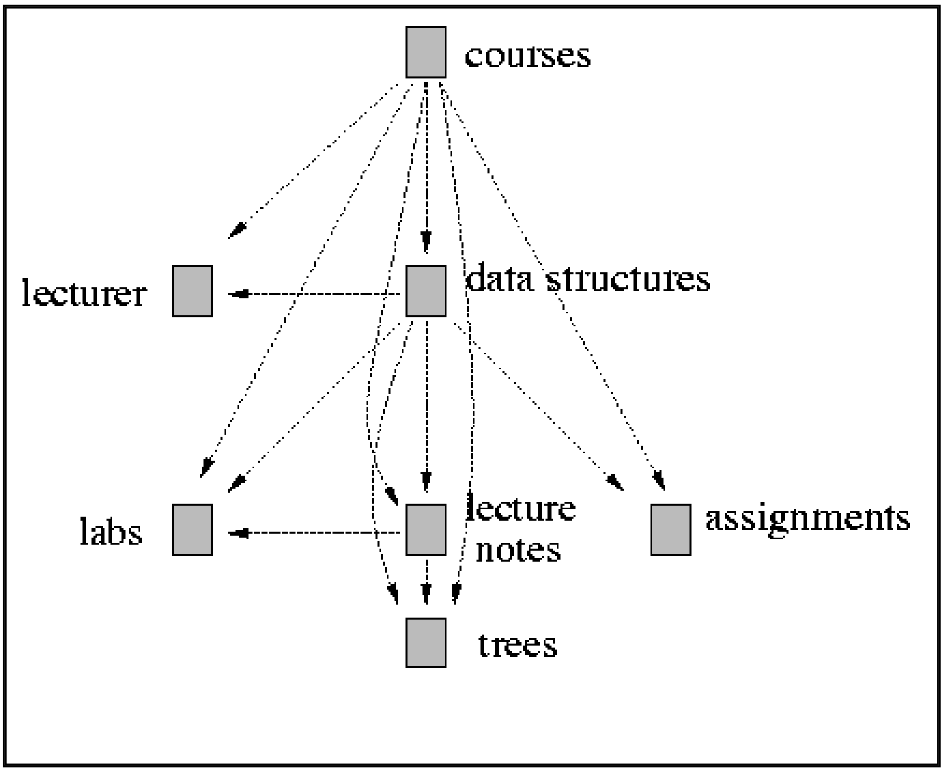


Figure 4: The resulting concept graph

While the concept graph is larger than the WWW graph, the increase in size is modest. Rule 1 eliminates some links, while rule 2 adds links for paths that are either completely within a single directory tree, or have at most one link outside of that tree. This adds $\boldsymbol{O}(NS\log_k(M))$ links to the concept graph, where N is the number of nodes in the WWW graph, S is the number of side links above a node in the directory tree, k is the branching factor in the directory tree, and m is the number of nodes in a directory tree. For real-world concept graphs N will be on the order of hundreds of thousands, M will be in the thousands, S will be less than ten, and k will be less than ten. We built concept graphs for ten sites and found that the concept graph was only between 50% and 300% larger than the WWW graph [Dyr97].

The benefits of creating a concept graph are two-fold. First, the directory structure is flattened, making concepts in subdirectories adjacent (but still subordinate) to those in top-level directories. Second, the possibility of navigating 'up' the directory hierarchy is eliminated.

Example queries

Users query the concept graph by giving a list of keywords. These queries are much like those that would be found in a WWW query language. But unlike these languages, complicated path expressions are not supported. A Jumping Spider permits only a single jump (link) between each keyword. All 'desirable' user navigations have been built into the concept graph. Below we list some example queries.

- Find the URLs of the *lecture notes* on *trees* in *data structures* subjects.

  data structures $\rightarrow$ *lecture notes* $\rightarrow$ *trees*

- Find the URLs of *lecture notes* on *trees*.

  lecture notes $\rightarrow$ *trees*

- Find the URLs of anything having to do with *trees* in a *data structures* course.

  lecture notes $\rightarrow$ *trees*

- Find the URLs of the *lecture notes* on *trees* in *data structures* subjects at *James Cook University (JCU)*

  JCU $\rightarrow$ *data structures* $\rightarrow$ *lecture notes* $\rightarrow$ *trees*

- Find the URLs of the *lecturers* in a *data structures* course at *JCU*.

  JCU $\rightarrow$ *data structures* $\rightarrow$ *lecturer*

- Find the URLs of all *lecturers* in a *courses* at *JCU*.

  JCU $\rightarrow$ *subjects* $\rightarrow$ *lecturers*

Evaluating queries

In this section we briefly describe the query evaluation technique that we implemented. Our implementation iteratively builds a path through the concept graph between pairs of URLs that match each keyword. The path is built ‘backwards’ (that is, from the n^{th} keyword to the first). The query evaluation architecture is a (relational) database, which is described in detail elsewhere [Dyr97]. The database has a *SearchEngine(concept, url)* table, which maps concepts to URLs, and a *Reachable(from, to)* table, which records which nodes can be reached from which others (it is the edges in the concept graph). Below we present an outline of the algorithm.

Assume that a query consists of a list of keywords, k_1, ..., k_n.

> /* Map each keyword, k_i, to a set of URLs U_i */
> U_1 := SELECT *url* FROM *SearchEngine* WHERE *concept* = k_1
> ...
> U_n := SELECT *url* FROM *SearchEngine* WHERE *concept* = k_n
>
> /* Try each candidate */
> for each z in U_n do
>
>> /* Establish a set of URLs that are still active */
>> *active* := {z}
>>
>> /* Iterate through the rest of the keywords */
>> for i := n-1 to 1 do
>>
>>> /* for each u in *active* determine all the nodes that can reach u */
>>> *canreach* := SELECT *from* FROM *Reachable* WHERE *to* IN *active*
>>>
>>> /* Which URLs match the current keyword and can reach the next keyword? */
>>> *active* := *canreach* INTERSECT U_i
>>
>> /* If anything is still active, we have a path */
>> if *active* is not empty THEN add c to *result*
>
> /* All done, publish the final result */
> report *result*

This is essentially a brute-force method with a worst-case complexity of $\boldsymbol{O}((uf)^n)$ where $u = |U_i|$ and f is the number of edges that can reach a particular node (uf is essentially the size of *canreach*, we assume that with hashing the intersection can be performed in $\boldsymbol{O}(1)$).

The primary difference between this navigation strategy and that in WWW query languages is that only a single link is ever traversed between concepts, whereas the evaluation of a path expression between two concepts in a WWW query language query may traverse many links.

We anticipate that the size of the concept graph will limit the number of servers that a Jumping Spider can index. For instance, a Jumping Spider could be constructed for information available from the servers at James Cook University, or for the servers at all the universities in Queensland, or just for information in computer science courses, available from universities world-wide.

Reducing the size of the concept graph

The concept graph can be dramatically reduced in size by assuming that a down link exists from a page to all pages in either that directory or a subdirectory. This assumption has two benefits. First, it leaves only side links to unrelated directories in the concept graph. We note that Altavista [Alt] already stores these links, so this strategy could be implemented using Altavista directly (we plan to do so shortly). Second, it improves the speed of query evaluation by reducing the size of the *Reachable* table.

Jumping Spider Demonstration

Building the Jumping Spider is relatively straight-forward. Like a search engine, a Jumping Spider, is a three-component architecture consisting of a robot, two tables, and a user-interface. The robot traverses the WWW and gathers information for a search engine index and the concept graph. The index and graph are stored as the pair of *SearchEngine* and *Reachable* tables.

The Jumping Spider architecture has been implemented in Perl, version 5.003. We chose Perl to facilitate easy distribution. The code and an example interface are available at the following URL.

http://www.cs.jcu.edu.au/~curtis/htmls/JumpingSpider/demo.html

Related Work

W3QL [KS95], WebLog [LSS96], WebSQL [MMM96, AMM97, MMM97], and AKIRA [LSCS97] are query languages that have been designed specifically to retrieve information from the WWW. Each of these languages supports sophisticated path expressions. AKIRA also supports ‘‘fuzzy’’ navigation whereby a user can retrieve information ‘‘near’’ to a particular page. In contrast, a Jumping Spider uses concept refinement and lacks path expressions to direct and constrain a query. We believe that many WWW queries will be concept refinement kinds of queries and that, for these queries, a Jumping Spider is more ‘‘user-friendly.’’

W3QL, WebLog, WebSQL, and AKIRA each have a *dynamic* query evaluation architecture in which a query will dynamically traverse the WWW. A dynamic architecture has several benefits. The most recent version of the constantly changing WWW graph is always used. The architecture is scalable since no large tables need to be computed or stored. A rich set of content predicates can be supported since content is extracted dynamically (WebLog and AKIRA have extensive sets of content predicates). Dynamic pages (e.g., pages generated by CGI scripts) can be explored, which W3QL supports. In contrast, we propose a *static* architecture. Prior to query evaluation, a subgraph of the WWW is traversed and the graph stored in Reachable and SearchEngine tables. These tables optimize query evaluation within the subgraph. Our belief is that the user will be willing to use a slightly out-of-date WWW graph and sacrifice the other benefits of a dynamic architecture if doing so helps to improve the speed of query evaluation.

We share with WebSQL a practical focus both on the issue of cost and the means to control that cost by having different kinds of links. WebSQL distinguishes between local and global links. We distinguish between side, down, and back links. WebSQL estimates the cost of a query as a function of the number of global links in a path expression. We use side links to limit the cost of a query.

Semi-structured data models have also been proposed for the WWW [Bun97, BDHS96, MAG+97, FFLS97]. A semi-structured data model consists of a graph that is a hierarchy with a well-defined root branching to data values through a sequence of ‘role edges.’ A semi-structured query can be thought of as a regular expression over an alphabet of role descriptions; the query is evaluated by matching the regular expression against the role descriptions on a path. Unlike these papers, we focus on the issue of creating a semi-structured data model from the unstructured information in the WWW. A straightforward approach would make a one-to-one mapping between hyper-links in the WWW and links in a semi-structured data model. Our approach is to remove some links (back links) and adding a number of links (all reachable down links). A further difference is that the concept graph described in this paper is neither a hierarchy nor has a root, queries can start from anywhere within the graph.

Finally, a related research area is that explored by WebGlimpse [MSB97]. WebGlimpse indexes pages within a ‘‘neighborhood’’ of a given page. The neighborhood can be a region of the WWW graph, such as the region within two links of a given page. In this paper, the neighborhood is a directory hierarchy (like glimpseHTTP). But the focus of this paper is on indexing jumps between neighborhoods. Essentially the index build by glimpse is a SearchEngine table in our implementation.

Summary

In this paper we presented a Jumping Spider, which is a tool to find information on the WWW. A Jumping Spider adds a limited, but desirable kind of WWW query language search to a search engine. The heart of a Jumping Spider is a concept graph that has links which lead from more general to more specialized information. We discussed how to build this graph by identifying different kinds of links in the WWW graph and restructuring the graph appropriately. Our strategy uses a simple heuristic, based on URLs. We plan to investigate other heuristics in future. Whatever heuristic is used, the key insight in this research is that the WWW graph should be massaged to improve the speed of query evaluation when using path expressions. We showed that the Jumping Spider implementation has an efficient, architecture which requires no changes to the WWW itself, and utilizes standard relational database technology to evaluate a query.

References

Alt Altavista Home Page. `http://www.altavista.com`.

AMM97
G. Arocena, A. Mendelzon, and G. Mihaila. Applications of a Web Query Language. In *Sixth International World Wide Web Conference*, Santa Clara, CA, April 1997.

BDHS96
Peter Buneman, Susan B. Davidson, Gerd G. Hillebrand, and Dan Suciu. A Query Language and Optimization Techniques for Unstructured Data. In H. V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 505-516, Montreal, Quebec, Canada, 4-6 June 1996.

Bun97
P. Buneman. Semistructured Data. In *SIGMOD/PODS ’97 (tutorial notes)*, Tucson, AZ, May 1997.

Dyr97
C. Dyreson. Content-based Navigation in a Mini-World Web. Technical Report 97/06, James

Cook University, 1997.

FFLS97
M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for a Web-Site Management System. *SIGMOD Record*, 26(3), September 1997.

Inf Infoseek Home Page. `http://www.infoseek.com`.

KS95
D. Konopnicki and O. Shmueli. W3QS: A Query System for the World Wide Web. In *Proceedings of the International Conference on Very Large Databases (VLDB '95)*, pages 54-65, Zurich, Switzerland, September 1995.

LSCS97
Z. Lacroix, A. Sahuguet, R. Chandrasekar, and B. Srinivas. A Novel Approach to Querying the Web: Integrating Retrieval and Browsing. In *ER97 - Workshop on Conceptual Modeling for Multimedia Information Seeking*, Los Angeles, CA, November 1997.

LSS96
L. Lakshmanan, F. Sadri, and I. Subramanian. A Declarative Language for Querying and Restructuring the Web. In *Proceedings of the Sixth International Workshop on Research Issues in Data Engineering (RIDE '96)*, New Orleans, February 1996.

MAG + 97
J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54-66, September 1997.

MMM96
A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, pages 18-20, Miami, FL, December 1996.

MMM97
A. Mendelzon, G. Mihaila, and T. Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1(1):54-67, January 1997.

MSB97
U. Manber, M. Smith, and G. Burra. WebGlimpse -- Combining Browsing and Searching. In *Usenix Technical Conference*, pages 6-10, January 1997.

An Architecture for Web Visualisation Systems

Petros A. Demetriades and Alexandra Poulovassilis
Department of Computer Science
King's College London,
Strand, London WC2R 2LS, UK

e-mail: {petros, alex}@dcs.kcl.ac.uk

Abstract

We describe an architecture for developing World Wide Web (web) visualisation systems. Our approach differs from that used in other such systems in that it is not an implementation of a particular visualisation technique but rather a framework which allows the development of extensible visualisation systems, and which facilitates experimentation and re-usability of components. This is achieved by providing a mechanism for interconnecting the three basic types of entities involved in visualising the web: Collectors, which are agents that gather web-related data to be visualised; DataStores, which are repositories into which the collected data is stored; and Views, which perform the actual interpretation of the data into some visual form. It allows many different Collectors and Views to be connected to the same DataStore and to simultaneously update and visualise it. In addition to facilitating experimentation and side-by-side comparison of different collection, storage and visualisation methods and techniques, this architecture also promotes re-usability of existing components, such as visualisation engines, web-spiders and data repositories.

1. Introduction

The popularity and widespread use of the web has caused a number of usability problems to surface. The two most troublesome ones have come to be known as "information overload" and the "lost in hyperspace syndrome". The former refers to the difficulty in discovering relevant information and resources on the web and the latter to the difficulties associated with orientation within, and navigation of, the web. These problems are closely related to the fact that the web and the Internet are very large and continue to expand at an exponential rate [1, 2]. Unless these problems are resolved, the usefulness of the web as an information resource will decrease as its size increases.

One approach to a solution is the development of effective techniques for visualising web-space. We define "web-space" as a collection of HTML documents (web-pages) that are somehow connected. Examples of web-spaces are arbitrary sub-sets of the entire web, users' browsing histories or the results of a web search. Visual interpretations of web-space can aid navigation and orientation in cyber-space in much the same way that conventional road maps aid navigation and orientation in real-space. Similarly, they can improve information discovery by facilitating exploration and navigation of the results of web-queries. We define a "web-query" as the process of searching for information on the web using any of the available methods. We term the set of URLs returned by a web-query the "hits" of the web-query.

The process of visualising web-space involves 3 activities:

1. collection of the web-space,
2. temporary or persistent storage, and
3. interpretation of the web-space into some visual form.

How we collect, store and subsequently visualise web-space, depends on its exact type. The three types of web-space identified above, for example, could be collected, stored and visualised as follows:

Firstly, an arbitrary subset of the web can be collected using a web-spider [10, 11] and stored in either a relational database using two relations (one for the web-pages and one for the links) or in a database which supports set attributes in one relation `<U,S>` where `S` contains the URLs of all the pages to which the page `U` links to. This information can be visualised as a directed graph where nodes are web-pages and edges are links between web-pages so that an edge (V_1,V_2) represents a physical link from page V_1 to page V_2 and indicates that the web-page V_1 contains a hyperlink to the web-page V_2. For example, Figure 5 shows how a subset of "http://www.dcs.kcl.ac.uk" comprising 5 pages could be stored and visualised. The example assumes that the index page A links to pages B, C, D and E, that pages C, D and E contain no links, and that page B contains a link back to the index page A.

Secondly, the results of web-queries can be collected either by a search client which interrogates Internet Index Servers such as AltaVista [3], Infoseek [4] and Excite [5] or by some other means such as WebSQL [6]. The results can be stored in both a relational DBMS and a DBMS which supports set attributes in one relation containing the URLs returned. Alternatively, if the approach we chose for our example system described in Section 4 is used then the results can be stored in an RDBMS as two relations, one containing the host part of the URLs returned and the other containing the relative path to each particular hit within the host , or in a DBMS that supports set attributes as one relation `<H,I>` where `I` is the set of hits that reside on host `H`. Figure 6 demonstrates this for a web-query that results in the 3 hits:
`http://www.dcs.kcl.ac.uk/petros/WWW7Paper.html`,
`http://www.dcs.kcl.ac.uk/alex/WebPaper.html` and `http://www.w3c.org/Visual.html`.
Such web-space can be visualised as a list of hits, grouped by host as shown in Figure 4.

Lastly, browsing history is not generally collected in a user-initiated manner, as for the above two types of web-spaces, but by a process continuously running along-side a browser or by a proxy server. It can be stored in an RDBMS in two relations one for the URLs of visited pages and one for the details of each visit. It can be visualised as a labelled, directed graph where nodes represent web-pages and labelled edges represent the order in which pages are visited. For example, if a user browsed the web-subset described in Figure 5 in the order A, C, A, E, A, D, A, B and then jumped to the web site of the WWW Consortium (http://www.w3c.org) using a predefined bookmark then this session could be stored and visualised as in Figure 7.

This need for different collection, storage and visualisation requirements for each type of web-space has led us to the development of an architecture to facilitate the experimentation and exploration of different collection, storage and visualisation techniques with minimal re-development and re-design effort.

We give a description of the conceptual structure of our architecture in Section 2 of the paper and proceed to give implementation details of one instance of this architecture in Section 3. To demonstrate the possible visualisation systems that could be built within we give details in Section 4 of one specific implementation that we have recently undertaken. We end in Section 5 by summarising our conclusions and future research directions.

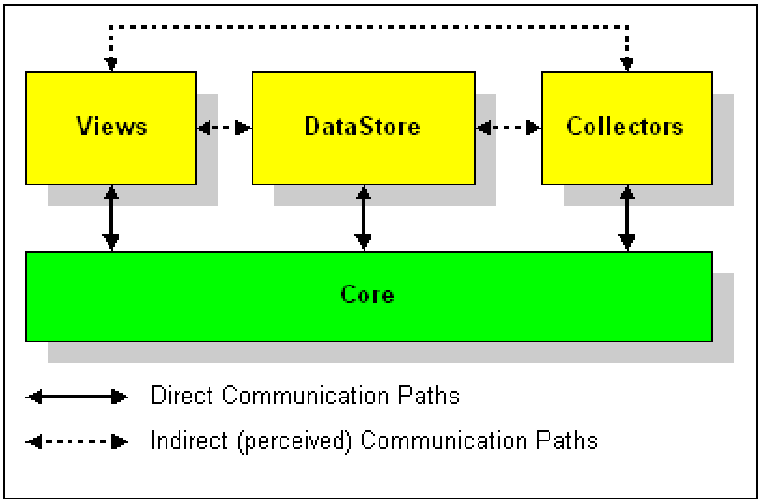
2. The Conceptual Architecture

Conceptually the architecture (Figure 1) consists of three independent, stand-alone entities:

- Collectors, which are agents that collect web-space,
- DataStores, which store web-space, and
- Views, which display visualisations of the web-space, and provide an interface for user interaction with the system.

A core layer binds these entities together and provides a mechanism for communication and reliable message passing between them. This further reduces the complexity that needs to be built into each entity by transparently performing certain operations.

Figure 1: Block Diagram of our Architecture



The core exposes a set of interfaces to all entities to abstract the operations of performing general querying and updating of the web-space irrespective of the actual data model used to store it, and message passing between the entities. In addition to its role as an "abstraction layer" between the entities the core also transparently performs some behind-the-scenes operations which further reduce the complexity of each entity. Firstly, whenever a view or collector performs a change to a DataStore the core immediately notifies all other entities of the change performed (in as precise terms as possible) so that they can alter their states to reflect the new state of the system. Secondly, since DataStores are essentially shared resources, it provides concurrency control and prevents access to them by more than one entity at the same time (this facility is only used for DataStores that do not provide their own concurrency control).

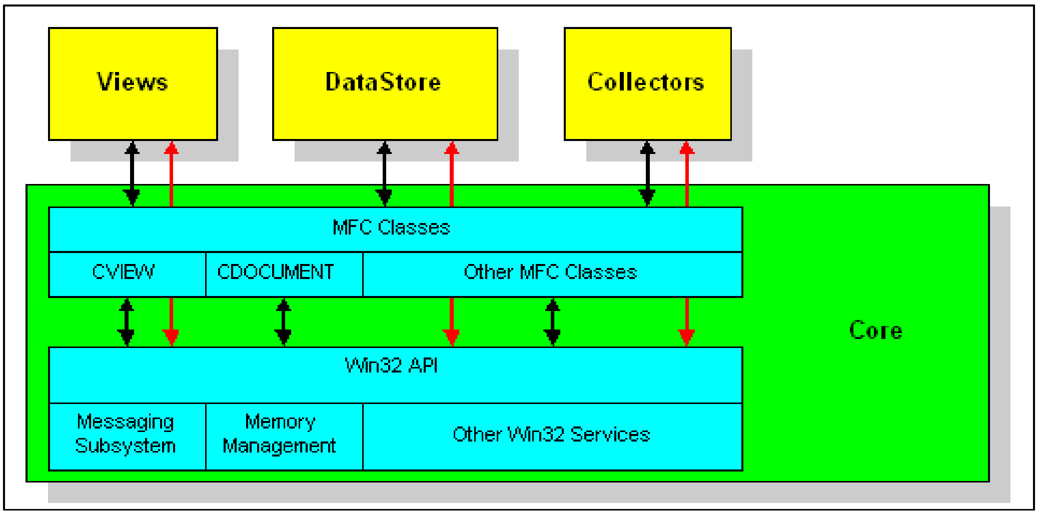
3. Physical Implementation of the Architecture

We have implemented a prototype of our architecture for the Windows 95 and Windows NT Operating Systems (collectively known as Win32). The prototype is written in C++ and uses the MFC application framework [15]. The main characteristics of this implementation are:

- Use of MFC's document-View architecture [16], to provide an automatic method of attaching multiple views to a single data repository (termed "the document" in MFC) and of propagating changes to the data to all attached views.
- Object Oriented: Each type of entity is represented as a class. Instantiations of these classes represent different instances of each entity. For example views are represented by instances of classes derived from the MFC class CView.
- Use of the messaging subsystem of the Windows OS to implement messaging between the entities, and shared memory for passing data between the entities.
- The single DataStore instance, and each instance of the View and Collector entities is executed as a different thread of the same process. This maintains concurrency while simplifying development as it allows the use of shared memory for parameter and data passing (under Win32 threads of a single process share the same address space).

Figure 2 shows the physical architecture and how it relates to the conceptual architecture of Section 2.

Figure 2: Physical Architecture and relation to Conceptual Architecture



Although this prototype is implemented on Win32 it is possible to bind together entities running on different platforms (e.g. UNIX) as long as there is some way of "talking to them". All that is required is to code a small stub of the required type of entity on the Win32 platform to act as the go-between the two systems.

This implementation suffers from two limitations. Firstly the core and at least a small part of each entity must reside and execute on the Win32 platform even if the actual entity resides and executes on a different system or platform. This reduces flexibility and slightly increases development time. Our architecture can be greatly enhanced if it is implemented in a distributed computing model such as that specified by CORBA [18] or DCOM [19] as this will allow much more flexibility in the kind of entities that could attach to an existing system and would enhance entity location independence and transparency.

Secondly, the update and query operations that the views and collectors can perform are limited to insertion, sorting, filtering and counting. More complex operations must be performed using a direct interface to the physical database system used in the DataStore. This is not a limitation of the architecture but rather a shortcut we have taken to reduce the development time of this first prototype. What is required in order to perform more complex queries is the implementation of an additional interface function to allow sending arbitrarily complex queries in the form of some DML, e.g. SQL, to the DataStore.

4. An example system: Visualising the results of web-queries

In order to demonstrate our architecture and test our prototype we have developed a system for visualising the results of web-queries. The objective of the system is to facilitate information discovery in three ways: Firstly, by storing the hits in a database and allowing the user to interactively explore them by performing operations such as sorting, searching and filtering. Secondly by having multiple collectors (two, in this simple demonstration system) that query index servers simultaneously. Thirdly by clustering hits by host and assigning a "hit-count" attribute to each host i.e. the number of hits that reside on it [1].

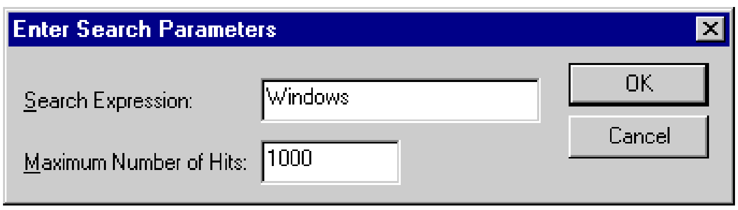
The system comprises the following entities:

1. Three Views: The HostsView displays host information, the HitsView displays hit information and the User Interface View serves as a container for the other two views and provides a unified interface to a user.
2. Two Collectors: These collect web-space by querying Internet index servers. They submit a query to their assigned index server, receive the HTML document returned, parse it, extract the hit URLs and descriptions from it and send this information to the DataStore for persistent storage. These two particular collectors query the Infoseek and Excite index servers.
3. Two Linked Lists as the DataStore: For this demonstration system we used two linked lists as our DataStore. One list stores the hosts and the other the hits.

Each of these entities is implemented as a different class as described in Section 3. The system operates as follows:

The user enters keywords describing the information sought, using the interface provided by the User Interface (UI) view (Figure 3). This view creates the two collectors and attaches them to the core using the *AttachEntity* function provided by the core and passes the query term to them. The collectors submit the query to the two Internet index servers simultaneously and wait for the servers to begin relaying the results. Upon receipt of the HTML documents, the collectors begin a cycle of parse, extract hit, send to DataStore until either the index servers report that there are no more results, or the required number of hits has been reached. They use the core function *InsertNode* to send each hit to the DataStore. Whenever the core receives an InsertNode request it forwards it to the DataStore and upon successful insertion, sends a *NodeInserted* message to all the views. The UI view does not act upon this message but the other two do. The HostsView displays any new hosts using the *GetNode* function and updates the count of hits per publisher using the *GetCount* function. The HitsView displays the new hit again using the *GetNode* function.

Figure 3: The query entry dialogue box



When the required number of hits has been reached or the index servers report no more hits, the collectors terminate and send a *Collection Termination* message. This is received by all other entities which act as follows:

- The UI view requests that the data is sorted in decreasing order of hit-count (using the *SortData* function).
- The visualisation views display any last additions to the data and enable UI controls for sorting and filtering the data.

When the core receives the *SortData* request it sends it to the DataStore. Upon successful completion it sends a *DataChange* message (along with the information that the change is one of sort order) to the views which act as follows:

- The UI view ignores it.
- The HostsView re-draws itself since the change in sort order affects it.
- The HitsView ignores it as the change does not affect it.

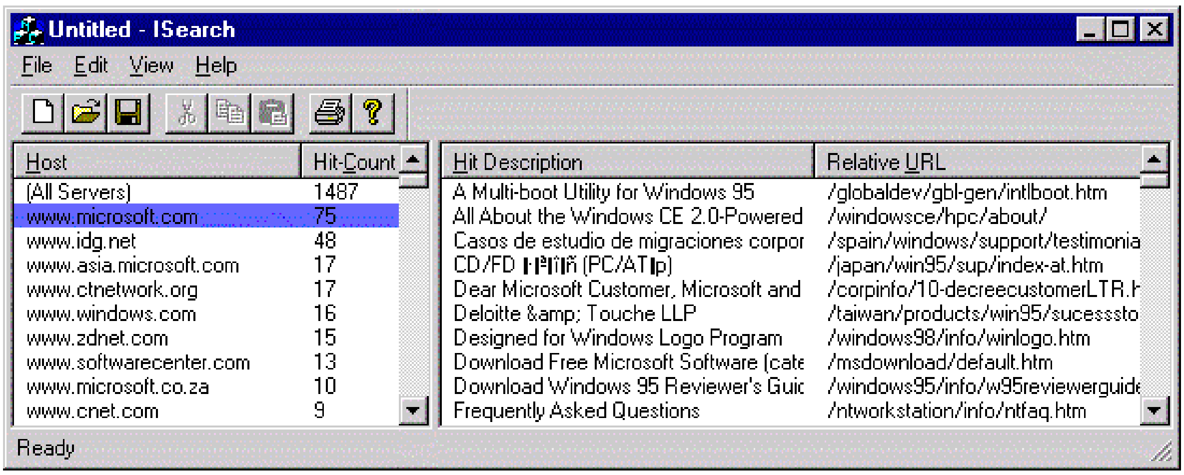
Now that the simple interactivity features of the views have been enabled, the user can interact with them. For example clicking on a host causes the hosts view to send a *SetFilter* request to the DataStore via the core. This request also states what the filter is.

Upon successful completion the core sends a *DataChange* message to the views along with details of what the change was. The views act as follows:

- The UI view ignores it.
- The HostsView highlights the host on which hits are now filtered.
- The HitsView re-draws itself (i.e. re-draws the list of hits which now only shows the hits on the host the user clicked on).

Figure 4 shows a screen shot of the three views. The enclosing frame with the menu bar and tool bar is the UI view. The left-hand side pane is the HostsView and the right hand side is the HitsView. The query term entered was "Windows".

Figure 4: Our demonstration system - A Search Client



5. Conclusions and Further Work

We have described an architecture for developing web visualisation systems. Our architecture is novel in that it facilitates:

1. Re-usability of existing DBMS, visualisation and web-space collection systems by providing a simple mechanism for interfacing to them, even if no source code is available and even if they are running on different platforms.
2. Comparison of visualisation techniques as applied to the web by facilitating the application of different visualisations to the same web-space.
3. Experimentation of storing web information using different data models.

We demonstrated the feasibility of this architecture by implementing a simplified version for Windows 95 and NT and proceeded to show how visualisation systems could be built within this by utilising it to build a simple search client application.

We are currently investigating the effectiveness of the Hypernode Model [17] for storing web-spaces. This model is based on nested, directed graphs termed hypernodes. We believe that the model's inherent support for nesting of graphs will provide a powerful tool for structuring and visualising large web-spaces. We are also investigating the use of Hyperlog [16] for browsing, querying and updating such web-spaces. Hyperlog is a graph-based database language that visualises schema information, data, and query output as sets of hypernodes. This information is stored, browsed and queried in a uniform way. In the context of visualising and manipulating web-spaces we wish to explore using Hyperlog for structuring and restructuring web-spaces and for authoring of customised documents and links.

References

[1] Network Wizards.Internet Domain Survey, January 1998.http://www.nw.com/zone/WWW/report.html

[2] Matthew Gray of MIT. Internet Statistics: Growth and Usage of the Web and the Internet, June 20, 1996.http://www.mit.edu/people/mkgray/net/

[3] Digital Equipment Corporation, AltaVista Internet Index Server. http://www.altavista.digital.com/

[4] Infoseek Corporation, Infoseek Internet Index Server.http://www.infoseek.com/

[5] Excite Incorporated, Excite Internet Index Server. http://www.excite.com/

[6] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the World Wide Web, Journal of Digital Libraries 1(1), pp. 68-88, 1997.

[7] Tim Berners-Lee. Universal Resource Locators. Internet working draft, 1 Jan 1994 (now expired).http://info.cern.ch/hypertext/WWW/Addressing/URL/Overview.html

[8] Martin Charles Golumbic. Algorithmic Graph Theory and Perfect Graphs, Academic Press, Inc., 1980.

[9] David Eichmann. The RBSE Spider -Balancing Effective Search Against Web Load. In proceedings of the First International World Wide Web Conference, CERN Geneva, May 1994.http://www.cern.ch/PapersWWW94/spider.ps

[10] Roy T. Fielding. Maintaining Distributed Hypertext Infostructures: Welcome to MOMspider's Web. In proceedings of the First International World Wide Web Conference, CERN Geneva, May 1994.http://www.cern.ch/PapersWWW94/fielding.ps

[11] Charles Petzold. Programming Windows 95. Microsoft Press 1996.

[12] Microsoft Corporation. Programmers Guide to Microsoft Windows 95. Microsoft Press 1995.

[13] Robert Lafore. C++ Interactive Course. Waite Group Press 1996.

[14] Microsoft Corporation. Microsoft Foundation Class Library.http://premium.eu.microsoft.com/msdn/library/devprods/vc++/vcmfc/D2/vcmfchm.htm

[15] Microsoft Corporation. MFC Document/View Architecture Topics.http://premium.eu.microsoft.com/msdn/library/devprods/VC++/Progsguide/F1F/D24/S51EE.HTM

[16] Stefan Hild and Alexandra Poulovassilis. Implementing Hyperlog, a Graph-Based Database Language. In Journal of Visual Languages and Computing (1996) 7, pp. 267-289.

[17] Alexandra Poulovassilis and Mark Levene. A Nested-Graph Model for the Representation and Manipulation of Complex Objects. In ACM Transactions on Information Systems, Vol. 12, No. 1, January 1994, pp. 35-68.

[18] Object Management Group. CORBA for beginners.http://www.omg.org/news/begin.htm

[19] Microsoft Corporation. COM Technologies (COM,DCOM, COM+, MTS, and ActiveX).http://www.eu.microsoft.com/cominfo/. February 1998.

[20] Jeromy Carriere and Rick Kazman. WebQuery: Searching and Visualizing the Web through Connectivity In Hyperproceedings of the Sixth International World Wide Web Conference, Santa Clara, California USA, April 7-11, 1997. http://proceedings.www6conf.org/HyperNews/get/PAPER96.html

[21] Gustavo O. Arocena, Alberto O. Mendelzon, George A. Mihaila. Applications of a Web Query Language In Hyperproceedings of the Sixth International World Wide Web Conference, Santa Clara, California USA, April 7-11, 1997. http://proceedings.www6conf.org/HyperNews/get/PAPER267.html

---

[1] The rationale behind the hit-count attribute stems from the fact that current index servers are keyword-based and therefore not very accurate in selecting documents that are relevant to a web-query expression. We envisaged that if a host contains a large number of documents that the index servers "think" are relevant to a query (i.e. has a high hit-count) , then it is more likely that they are all truly relevant to the query than the single document on some other host with a hit-count of one. The hit-count attribute can thus be considered a measure of the probability that documents residing on a host are relevant to a web-query. Figure 4 shows the results for the query "Windows" sorted by decreasing order of hit-count. As can be seen the first host listed (with by far the highest hit-count) is that of the web site of Microsoft Corporation. The use of "connectivity" and "structure" in such a manner is not new and has been proposed as a solution to this problem of imprecision of keyword-based searching in several different forms [20, 21]. Our experience with our prototype system supports this view, as a definite improvement over the standard method of searching the web (i.e. using an index server directly), was perceived. We however postpone a final judgement to a future publication, pending a thorough, larger-scale investigation.

Figure 5: Storage and visualisation of an arbitrary subset of the web

Storage

1. Relational DBMS: One relation for the web-pages (nodes) and one for the links (edges)

(Note that the host part "http://www.dcs.kcl.ac.uk/" of the URL for records A to E is not shown here to save space.)

NODES

Page ID	Page URL
A	index.html
B	teaching/yellow-book/index.html
C	teaching/msc/index.html
D	research/phd-CS.html
E	teaching/index.html

EDGES

Page ID	Links To
A	B
A	C
A	D
A	E
B	A

2. DBMS that supports set attributes: One relation for both nodes and edges

Page ID	Page URL	Links To
A	index.html	{B,C,D,E}
B	teaching/yellow-book/index.html	{A}
C	teaching/msc/index.html	ø
D	research/phd-CS.html	ø
E	teaching/index.html	ø

Visualisation

1. Directed Graph:

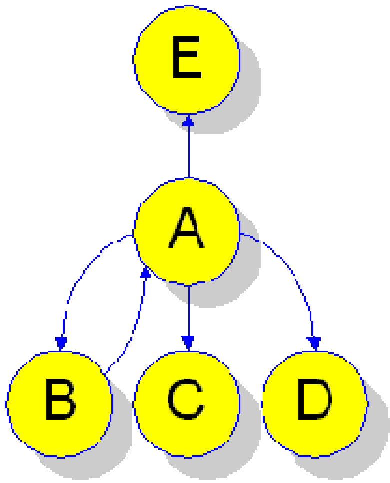


Figure 6: Storage of the results of web-queries

1. Relational DBMS: One relation for the hosts and one for the hits

HOSTS

Host ID	Host Part of URL	Hit-Count
1	http://www.dcs.kcl.ac.uk	2
2	http://www.w3c.org	1

HITS

Hit Part of URL	Host ID
/petros/WWW7Paper.html	1
/alex/WebPaper.html	1
/Visual.html	2

2. DBMS that supports set attributes: One relation for both hosts and hits

Host	Hit-Count	Hits On Host
http://www.dcs.kcl.ac.uk	2	{"/petros/WWW7Paper.html", "/alex/WebPaper.html"}
http://www.w3c.org	1	{"/Visual.html"}

Figure 7: Storage and visualisation of browsing history

Storage

1. Relational DBMS:

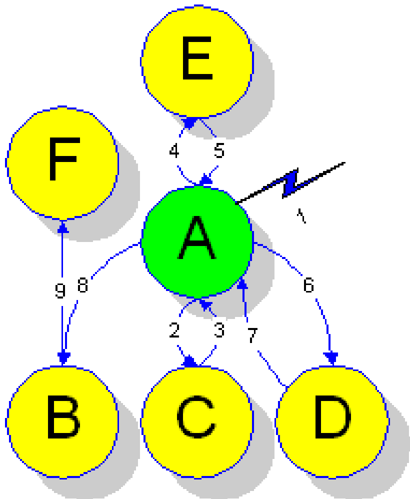
(Note that the host part "http://www.dcs.kcl.ac.uk/" of the URL for records A to E is not shown here to save space.)

ID	Page URL
A	index.html
B	teaching/yellow-book/index.html
C	teaching/msc/index.html
D	research/phd-CS.html
E	teaching/index.html
F	http://www.w3c.org/

Order	Date	Time	ID
1	20/03/1998	14:00:40	A
2	20/03/1998	14:02:00	C
3	20/03/1998	14:03:00	A
4	20/03/1998	14:03:02	E
5	20/03/1998	14:05:12	A
6	20/03/1998	14:05:15	D
7	20/03/1998	14:07:30	A
8	20/03/1998	14:07:35	B
9	20/03/1998	14:09:40	F

Visualisation

1. Labelled Directed Graph:



Component Advisor: A Tool for Automatically Extracting Electronic Component Data from Web Datasheets

Malu Castellanos, Qiming Chen, Umesh Dayal, Meichun Hsu, Mike Lemon, Polly Siegel, Jim Stinger
Hewlett-Packard Laboratories, 1501 Page Mill Road
Palo Alto, CA 94304, USA
{castella,qchen,dayal,mhsu,lemon,polly,stinger}@hpl.hp.com

1. Introduction

As the popularity of the Internet explodes, many suppliers of electronic components have found it compelling to publish product information on the Web. Their customers have also found it easier to obtain up-to-date product information directly from the Web. Electronic publishing of product information on the Web is expected eventually to replace the use of *printed datasheets* for electronic components [Rom97].

Electronic publishing further opens up an opportunity for corporate procurement organizations to streamline the production and use of their *Component and Supplier Management* (CSM) databases. A CSM database contains *parametric information* about components so that electronic design engineers in the company can search for suitable components via a structured query based on the properties of components, and corporate procurement personnel can better control and qualify the supply chain. CSM databases have traditionally been maintained manually by procurement librarians who are experts in the product domain. The librarians extract information about the components from printed datasheets or data books, transcribe the information into the format and terminology prescribed in CSM, and enter the extracted information into the CSM databases.

While electronic publishing of datasheets or data books makes it possible to obtain product information directly from the Web, it does not eliminate the need for a CSM database. Electronic datasheets are available on the Web as unstructured text, and the essentially keyword-based search capability existent on the Web cannot offer answers to queries such as "*Find the flash component on the market with cycle time no greater than 120ns and availability within 2 weeks*." However, with electronic publishing of datasheets, there is now the potential of largely automating the extraction process, thereby reducing the cost of CSM database maintenance, and significantly increasing its timeliness and quality of service. The CSM database can also become an active component of the CSM function, facilitating discovery of new components, suppliers, and supply chain dynamics, and leading to a transformation of the CSM function itself.

In this paper, we describe the *Component Advisor*, an on-going research project in extracting structured, parametric information about electronic components from datasheets published on the Web. The project is conducted as part of the *Web Content Mining* research effort at HP Labs, whose goal is to build intelligent Web mining agents. While we have chosen electronic components to be our initial domain, we postulate that the results can also be applied to other product domains.

Our approach is based on modeling domain knowledge, and using this to focus the work of the mining agent. We structure the domain model to consist of vendor-neutral knowledge about *products* and the conceptual structure of *documents (datasheets),* which is fairly regular within the industry, *and* knowledge that is specific to particular *vendors? datasheets.* Our hypothesis is that our *model-based* approach, which exploits regularity within a domain, is more scalable and resilient to changes than *format-based* approaches that required detailed modeling of individual documents. A prototype mining agent has been implemented in Java and Perl, and will be experimented with extensively to validate our hypothesis. In the longer term, parts of the domain knowledge may actually be updated or discovered by the mining agent as it roams the Web to extract information.

2. Related Work

Web content mining has received considerable attention lately, and many efforts have been reported; we briefly review some of these efforts below.

In the mediator architecture [Wie92], site-specific *wrappers* provide integration of information from heterogeneous databases; as the architecture is applied to integrate Web contents, wrappers become extraction procedures for documents on a Web site. Several techniques to build wrappers have been reported in the literature. Feature extraction techniques offer useful operators but do not tend to take advantage of domain knowledge. For example, in [HGCAC97], a method for feature extraction based on a set of text operators is proposed. Each text operator takes a text variable as input and assigns the resulting text to another text variable. Feature extraction consists of manually constructing an extraction script composed of these operators.

Other techniques attempt to automate, to different degrees, the generation of wrappers. [AK97] describes a method for semi-automatically generating wrappers based on heuristic rules. The idea is to exploit the formatting information from the source to hypothesize the hierarchical underlying structure of the page. NoDoSe [Ade97] is an interactive tool for semi-automatically determining the structure of documents and extracting their data. The user hierarchically decomposes the document and maps it to a previously defined structural model of the document. The task is expedited by a mining component that attempts to infer the grammar of list and record types from the user input. In [KWD97] a system to automatically extract data from Web pages is described. The data must be represented as a set of tuples; no deep structure is inferred. A machine learning algorithm infers the grammar of a document from a set of instances of the document type. It uses domain knowledge in the form of oracles that can identify interesting types of fields within the document.

The previous approaches make important contributions to the field; however, a limitation that they have in common is that they are too format dependent. The wrappers are based on the exact format of the Web document at a very fine level of granularity. Wrappers must be regenerated each time the format of the page changes, which happens often. Format independent approaches that overcome this limitation are needed.

Several *shopping agents* have been proposed [DEW97, Bar, KRU97]. A shopping agent is capable of collecting from the Web simple parametric information, such as price and availability, about products in a specific domain, and presenting the comparison information to the user. So far these efforts have been deployed in domains such as personal software and music CDs, using the search forms provided by the vendors to obtain the information; [DEW97] incorporates vendor form-learning logic. However, complex text documents such as datasheets have not been dealt with in these efforts.

Commercial CSM vendors, such as Aspect Development [Asp] and IHS Engineering [IHS], still rely primarily on a manual approach to collect and update information in their component databases. Data interchange standards are being proposed [ECIX], but they are not expected to be adopted widely in the short run. Web content mining, in the meantime, holds considerable promise for streamlining Component and Supplier Management.

3. Domain Model and Vendor Catalog

Two types of knowledge drive the Component Advisor, vendor-neutral and vendor-specific. Vendor-neutral knowledge is captured in the *domain model* and consists of two parts. The *product concept model* describes the product family including the hierarchy of product types as well as their characteristics. Figure 1 shows this for memory components. Each characteristic is modeled by a set of attribute-value pairs. The values of some attributes are the keywords that usually accompany the specification of the characteristic, like label and unit. The other attributes are the data type and constraints on the value of the characteristic. Besides the product family itself, the documents containing the product specifications -- datasheets in the electronic component domain -- also need to be modeled. This is done in the *document model* which describes the structure of the document in terms of its sections, along with the relationships between sections and product characteristics. This is shown in Figure 2 for memory datasheets. Regularities exhibited by the documents are also captured in this model to provide more hints for the identification of relevant data. In datasheets for example, it is common to find a specific formatting structure, like a table or a list, for a particular section.

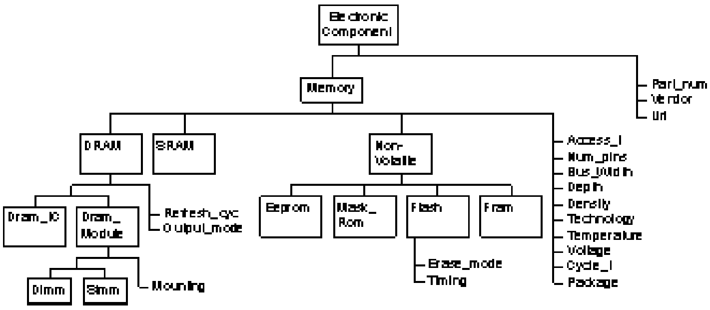


Figure 1. Product Concept Model: Memory.

Vendor-specific information such as home page URL, vendor specific terms (synonyms for those in the domain model), various indicators on the organization of the site, and so on, is captured in the *vendor catalog*.

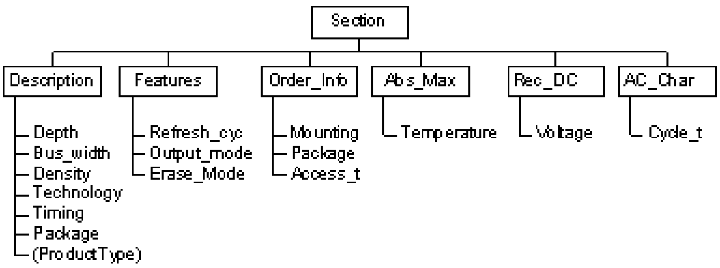


Figure 2. Document Model: Memory Data Sheet.

4. General Architecture

As stated in Section 1, the objective of the Component Advisor is to provide design engineers with up-to-date information about electronic components. This is done by a *mining agent back end* which mines the Web for component data in batch mode. Its tasks are to find the URLs of the product datasheets and analyze them to extract useful data. This data is then loaded into a database to be queried by design engineers through a *browser-based front end*. Figure 3 illustrates the general architecture of the Component Advisor.

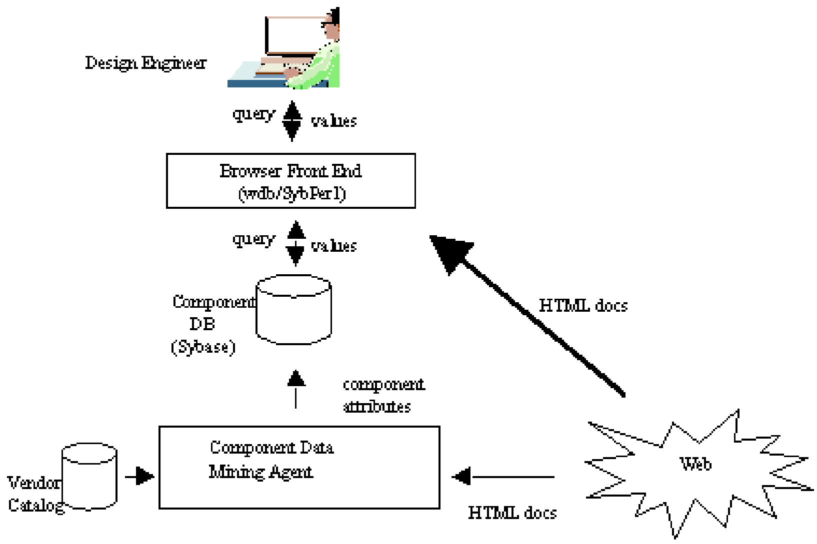


Figure 3. General Architecture of the Component Advisor.

More specifically, given the name of a manufacturer and its URL as input, the mining agent navigates through the pages of the manufacturer?s site to find the right page for the product family in order to obtain the URLs of the datasheets of individual products. Once the corresponding pages are retrieved, their contents are analyzed to identify and extract the relevant information. The *navigator* and the *extractor*, shown in Figure 4, respectively perform these two tasks. The *loader* (not shown here) takes the extracted data and loads it into the database. The whole process is under the supervision of the *controller* which invokes the appropriate system component at the right time.

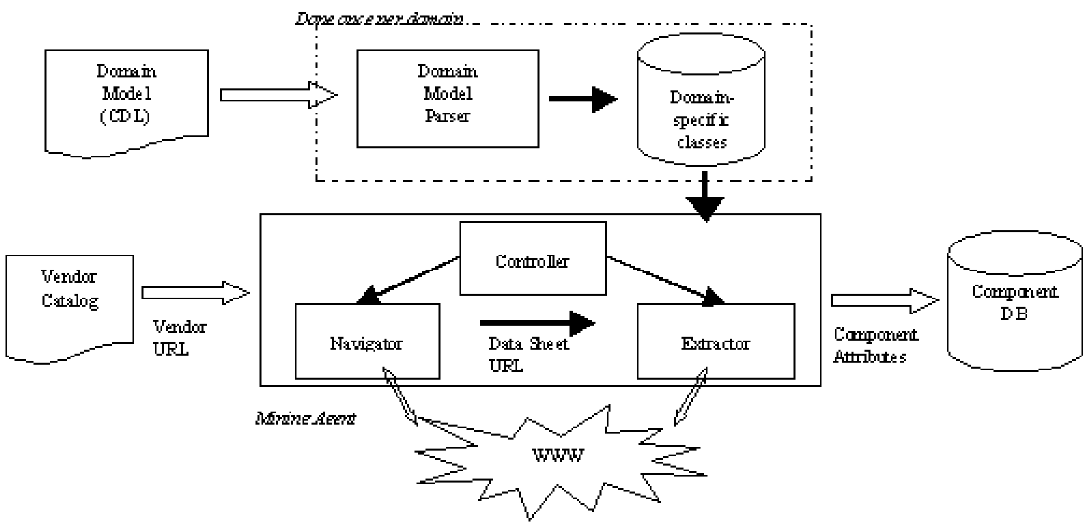


Figure 4. Mining Agent Architecture.

4.1 Navigator

Given the starting URL for a vendor?s Web site it is the job of the navigator to find the datasheets for that vendor?s components. Generally, the starting URL for a vendor will be the one for the vendor?s home page, but it need not be. Navigating from the starting URL to the vendor?s datasheets involves either following promising links (*link-based navigation*), or filling out search forms and submitting them (*form-based navigation*), or a combination of both.

We have broken down the navigation process into a series of actions to be performed. The first action is to retrieve the vendor?s starting Web page, which is done by the *visitor* module. The next action is to perform pattern matching on the Web page to locate either links to follow or forms to fill out, which is the job of the *pattern matcher* module. In the case of form-based navigation, the *form filler* module is called next to fill out the form, followed by a call to the *form submitter* module which submits the form to the search engine of the vendor. The results of the search that are returned by the vendor are then analyzed by a *link analyzer* module to determine whether these are the expected search results, i.e., a list of references to datasheets.

The challenge lies in determining which are the promising links to follow and which are the right forms to fill out. Backtracking may be necessary if a link or a form leads to a dead end. Once the navigator has found all datasheets for a given component family, the list of datasheet references is passed to the extractor for extraction of the component data.

4.2 Extractor

The extractor performs an iterative process on the list of datasheet references that the navigator obtained. For each datasheet, the extractor sequentially invokes different functionalities provided by its various modules. First, the *visitor* module retrieves the datasheet from the URL specified in the reference. The extractor then proceeds to analyze the Web document guided by the domain model. The product concept model is consulted for the description of the product type in terms of its characteristics. The datasheet structure is obtained from the document model, as well as the relationships between document sections and characteristics of the product.

By knowing in which section a characteristic is located, a *section recognizer* module is invoked to identify the portion of the document corresponding to that section. This can be done either by looking for a table of contents in the datasheet and obtaining the URL reference of the section, or by performing a search guided by heuristics to identify the section header in the document. Once the section is identified, a *structure recognizer* module has the task of recognizing the formatting structure specified for that section in the document model. Finally, a *pattern matcher* module is invoked to find the value of the characteristic.

In the product concept model only the keyword elements surrounding the specification of a characteristic?s value are defined. The idea behind this is to make possible the use of different techniques for the matching. The simplest one is to build regular expressions from the keywords, but at the cost of becoming dependent on the format of the text. Other more sophisticated techniques which make distance measurements can be used. In this way, if the matching fails with one technique, it is still possible to try another one. The architecture is extensible with respect to the sets of pattern matchers and structure recognizers.

The process is repeated iteratively for each characteristic to be mined from the datasheet. As the values are extracted they are written to a file in a specific format. Finally, when the extraction process on the datasheets of the vendor is completed, the file is passed to a loader to be loaded into the database.

5. Implementation and Discussion

The Component Advisor is implemented in Java using JDK 1.1. Those parts of the system that deal with pattern matching are written as Perl scripts. These scripts are called from Java using the Runtime and Process classes.

During implementation we uncovered a number of issues. One of these is poorly written HTML. For example, we encountered structures such as tables which are not defined with the appropriate HTML tags. This complicates the recognition of structures, in particular the identification of tables and their elements as well as the correlation of elements with the corresponding column headers. Another issue is that many datasheets are available only in PDF format and the HTML output generated by current PDF to HTML converters is poor at best. Datasheets and components may lack one-to-one correspondence, i.e., there may be one datasheet for multiple parts or multiple revisions of a single datasheet, complicating the extraction process. We also found a need to extend our domain model to cover such things as attributes whose value is derived from the values of other attributes.

When dealing with forms, we found that for some vendors the URLs returned in the search results are volatile, i.e., they become invalid after a short period of time. This required us to change the operation of the extractor so that these URLs are processed before they expire.

Before processing a datasheet, the extractor makes a pass through the HTML to normalize the text. For example, commas in numbers are removed, special characters, such as the degree symbol, are replaced with a sequence of standard characters (degC in this case), and number words are changed to their corresponding digits. The extractor also flattens table structures before extracting component information from them.

So far we have concentrated on extracting component information from one prominent vendor?s Web site. The prototype system runs successfully for memory component families such as DRAM, SRAM and non-volatile memories. For 172 DRAM datasheets it took the navigator approximately 1.5 minutes to navigate and the extractor approximately one hour and eleven minutes to extract the data. For 46 non-volatile memory datasheets it took the navigator 30 seconds and the extractor 20 minutes. This translates to approximately 2.5 datasheets per minute or about 25 seconds per sheet. This is unoptimized performance.

The operation of the Component Advisor is currently sequential; that is, the navigator finishes before the extractor takes over. One of the areas we want to investigate is the parallel operation of the navigator and the extractor and its effect on performance. It may also be possible for the extractor to do data extraction in parallel. Another area to investigate is the extraction of non-parametric attributes such as part footprints or schematic symbols.

6. Summary

This position paper describes Component Advisor, a prototype mining agent for extracting and reusing information from documents on the Web. The approach is based on modeling domain knowledge, separating vendor-neutral knowledge from vendor-specific knowledge, and using this model to direct the mining agent. We believe that this approach is more robust than format-based approaches that try to wrap each individual document or web site. We have partially validated the approach via a prototype implementation. Ongoing research is aimed at extending and refining the approach to assess how it will scale to a large number of vendors and product families. We are also interested in understanding how well the approach will carry over to other domains.

References:

[Ade97] Adelberg, B., "NoDoSe -- A tool for Semi-Automatically Extracting Structured and Semistructured Data from Text Documents". Northwestern University Computer Science Department of Computer Science, http://www.cs.nwu.edu/~adelberg/index.html

[AK97] Ashish, N., and Knoblock, C.A., "Semi-automatic Wrapper Generation for Internet Information Sources," *Proceedings of the Workshop on Management of Semistructured Data,* pp 10-17, Tucson, Arizona, May 1997.

[Asp] Aspect Development Inc., http://www.aspectdv.com

[Bar] BargainFinder Agent, Anderson Consulting, http://bf.cstar.ac.com/bf/

[DEW97] Doorenbos, R.B., Etzioni, O., and Weld, D.S., "A Scalable Comparison-Shopping Agent for the World-Wide Web", *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, 1997

[ECIX] Electronic Component Information Exchange, Silicon Integration Initiative. http://www.cfi.org/

[HGCAC97] Hammer, J., Garcia-Molina, H., Choa, J, Crespo, A., Aranha, R. "Extracting Semistructured Information from the Web", *Proceedings of the Workshop on Management of Semistructured Data,* pp 18-25, Tucson, Arizona, May 1997.

[IHS] IHS Engineering Inc., www.ihs.com

[Kru97] Krulwich, B., "Automating the Internet: Agents as User Surrogates". IEEE Internet Computer 1(4), pp 34-38, 1997.

[KWD97] Kushmerick, N., Weld, D.S.,Doorenvos, R., "Wrapper Induction for Information Extraction", *Proceedings of IJCAI*, 1997.

[Rom97] Romick, P., "The Information-Gathering Process for the Wired Engineer". Electronic Design, pp 61-65, January 6, 1997.

[Wie92] Wiederhold, G. "Mediators in the architecture of future information systems." *IEEE Computer*, March 1992.

Database Querying on the World Wide Web: *UniGuide*- An Object-Relational Search Engine for Australian Universities

Carlos F. Enguix, Joseph G. Davis, and Aditya K. Ghose

Decision Systems Lab

Department of Business Systems

The University of Wollongong

Northfields Ave.

Wollongong NSW 2522

joseph_davis@uow.edu.au **,** cfe01@wumpus.uow.edu.au **,** aditya@uow.edu.au

Please address all correspondence to Joseph G. Davis

(Word Count : 2981)

Abstract

The World Wide Web can be considered to be a huge semi-structured database that can provide us with a vast amount of information. Existing web search techniques have significant deficiencies with respect to robustness, flexibility and precision. The purpose of this research is to develop a domain-centred alternative to keyword and subject directory search engines. The specific domain being considered for the prototype implementation is that of Australian universities including all the internal entities that belong to each university such as faculties, departments, research centres, etc. that is available on the web. By modelling the ontology of this particular domain using an object-relational data model and restructuring the web data using an object-relational database, structured queries can be issued against this database in a fashion that current search engines do not provide.

1. Introduction

The explosion in the quantum of data available on the WWW in recent years has rendered the problem of discovering necessary information resources in reasonable time, somewhat difficult. The quality and usefulness of the global, distributed, hypermedia structure of the web is critically dependent on the availability of effective means by which users can obtain the required information efficiently. The efficacy of the existing search engines based on keywords and subject directories has been under severe strain.

An alternative approach that enables the database querying of web data is proposed in our research. We address some of the conceptual and practical questions dealing with, developing and structuring ontologies within well-defined domains such as health care, universities, etc. The ontology model, structured as an object relational database schema, is used to develop an object-relational database query search engine entitled "*UniGuide"*.

The paper is organised as follows: In section 2 we present an overview of the relevant literature. A model that captures the core constructs and their inter-relationships (ontology) of the university domain, the architecture, and implementation of *UniGuide* prototype are presented in section 3. The usage of the prototype from an end-user perspective is outlined in section 4. Sections 5 and 6 devoted to future research directions and conclusion respectively.

2. Overview of the Literature

2.1 Structuring the Web Data

The WWW can be considered as a huge semi-structured database, presenting all the problems implicit in semi-structured data [Abi97]. Extracting the structure of every HTML document is a challenging issue given the absence of predefined standard and schema. Often the schema can be derived only after the existence of data as compared to conventional databases where the schema is defined before the database is populated even though the schema can be very large and constantly evolving.

One of the possibilities to "put things in order" in this relative chaos, is to create a structured layer on top of the semi-structured layer along the lines proposed in [HZF95]. A feasible approach is that of attaching metadata that describes the kind of contents of individual web pages. This would permit us to view relevant information about web pages as a series of structured tuples of data. The approach is partly based on the assumption that metadata will/should be treated as first class objects [W3C97] and will serve as the interface from the WWW to a structured database. Because of the exclusive focus on metadata, there is no need for strict typing over the contents of the HTML documents but only over the required metadata. The type of metadata to be attached to web pages are basic and standard meta tags, name/value pairs that describe properties of the document. Examples of the most extended use of meta tags includes: keywords and description used by some of the most popular search engines. Other important standard is the meta tags proposed by the Dublin Core, a 15-element metadata set intended to facilitate discovery of electronic resources [COR97].

2.2 The Query Problem

A majority of the existing search engines provide a very simple interface to querying, a simple text box. We list below some of the most common deficiencies of current implementations:

- Most of the search engines are keyword-based, constrained to very limited structured querying, therefore providing more syntactic and less semantic precision

- The lack of control on querying data: the boundaries of the query are unknown, the output of a query is hard to predict

- The ability to establish relations between data elements is scarce or non-existent

2.3 Related work

Structured database querying on the WWW was proposed by Han, Zaiane, and Fu [HZF95]. A critical problem with their approach is that it was too generic, trying to model a schema that could represent the whole semantics of the WWW. A more realistic approach is to follow a strategy of "divide and conquer", identifying and isolating an arbitrary number of domains where a model can be derived and extracted. The ideal domains to "attack" are those that present large hierarchies of webpages. From this starting point we can identify and isolate a given number of entities that are present in almost every different domain instance (i.e. webpages of a particular university) of a given "characteristic" domain (i.e. university webpages). One should be aware that it is almost impossible to model all the different variations of a given entity, or to model all the possible entities in a "characteristic" domain. Probably we can represent up-to an 80 % of the possible entities of a given domain.

P. Atzeni et. al. [AMM+97] have proposed a data model and a view language in order to represent, query and restructure the information stored in structured web servers. Generally these servers are characterised by having their webpages stored in databases and having normalised not only the content of their webpages but the hypertextual structure (i.e. HTML tags) as well. This feature permits that attribute values can be extracted automatically using a text restructuring language.

Our focus is on domains where semantic models can be extracted. Although these domains are considered to be logically structured in general terms, they do not share a normalised hypertextual structure. This makes it almost impossible to extract attributes automatically from every given "target" webpage using general text restructuring programs. This impossibility justifies our approach of attaching structured meta-tags to webpages in order to extract the attributes of entity-instances represented in webpages.

3. *UniGuide*: Architecture

3.1 Introduction

The framework proposed in this paper is predicated on two significant assumptions:

- Ontologies or models of concepts and their relationships [MH97] represent powerful means to structure the global information base on the web

- The range and diversity of data on the web is so extensive that ontologies may have to be constructed separately for each relatively well-defined domain such as universities, health care, government departments, schools, etc.

Ontology is a term with a long pedigree in philosophy. It refers to things that exist (in the domain). For instance, it is reasonable to expect that the university domain will always have information regarding research entities, academic departments, courses, research outputs, and so on. Furthermore, these are likely to be inter-related in similar and predictable ways [MH97].

Our proposed method involves isolating a distinct domain, modelling its ontology using an object-relational data model, and storing structured data provided by UniGuide Scheme meta tags attached to the domain webpages into database tables corresponding to objects in the model. The UniGuide scheme meta tags are generated using forms-based input and finally attached to the required webpage in order to bring about the possibility of an indexing robot to populate the database automatically. This database becomes a resource that can be queried by end-users in a fashion that current search engines cannot match, allowing the execution of typical and non-typical SQL queries. We can emphasise that we can consider the process of querying the database as finding information rather than searching because the boundaries of a query can be delimited, a feature not available in keyword-based search engines.

UniGuide is a demonstration prototype implementation of the above approach. The ontology for all Australian University webpages is modelled as an object-relational data model. This model is then mapped to ILLUSTRA database tables and a set of queries that can be issued against this database is presented in subsequent sections.

3.2 The model: Object-Relational

The object-relational model of *UniGuide* is shown in figure-1. It shows the current entities that have been modelled but the model is extensible. The objects in the model represent the webpages of corresponding entities in the universities. Therefore there is a relation of 1:M relation (R[1:M]) between entity-instances and URLs. Generally, a webpage may contain many entity-instances but an entity-instance may have one and only one URL.

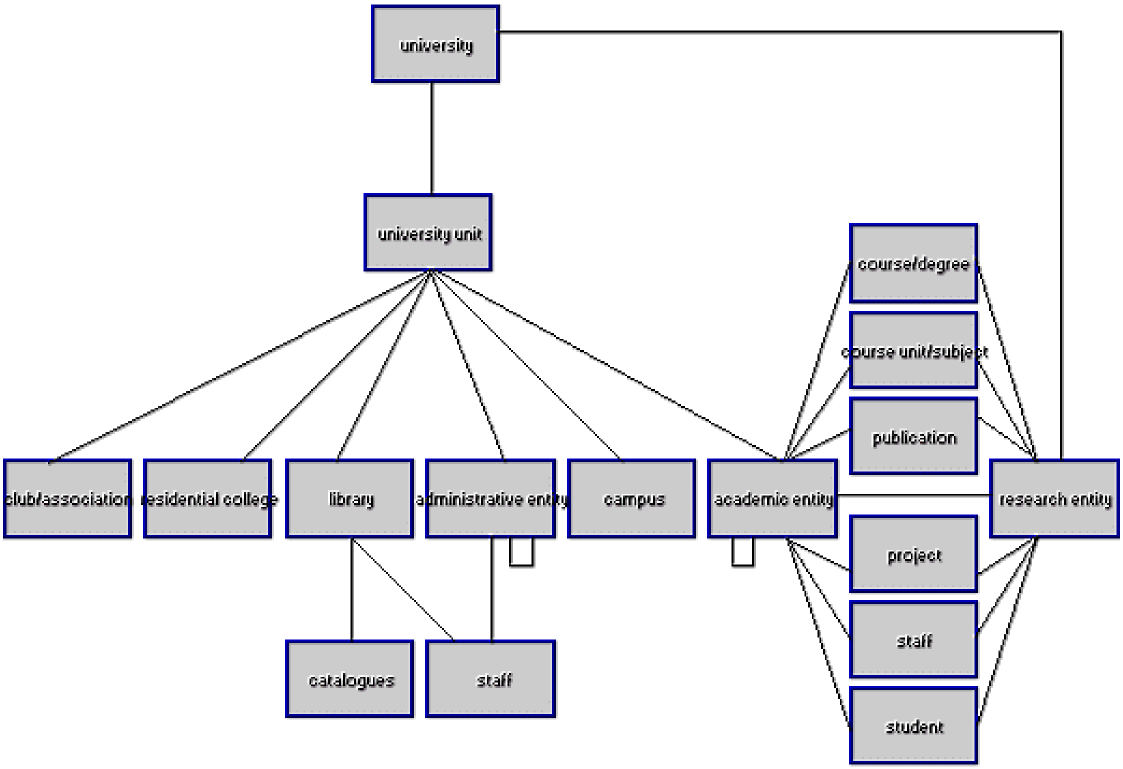


Figure 1: Synthesised graphical representation of the Object Relational Model of *UniGuide*

- A university may contain many university units.
- University unit is the superclass of all the following subclasses: club/association, administrative entity (division, office, etc.), library, residential college, campus and academic entities (faculties, departments, schools, etc.).
- A library may have a set of catalogues, and a set of staff members.
- An administrative entity can be one of the following types: Centre, Department, Division, Group, Institute, Office, etc. and may contain other administrative entities (i.e. Division of the Registrar contains various offices). An administrative entity may contain a set of staff members.
- An academic entity comprises the following types: faculty, department, school, unit, program, etc. An academic entity may contain other delegated academic entities (i.e. Faculty contains various departments)
- A research entity can be of type: research institute, research group, research centre, etc. Research entity is not considered as a subclass of university unit because a research entity can be part of many universities or can be a totally independent organism.
- Academic entities and research entities may have a set of publications, projects, courses, course units (subjects), staff members and students.

Finally we can distinguish relations between entities:

- Academic entity-research entity: an academic entity can-have/collaborate-with many research entities.

- Research entity-university: a research entity may belong-to/collaborate-with many universities

All entities contain a timestamp attribute in order to store date and time of last modification. This attribute will provide an effective mechanism for a customised indexing robot to decide whether a previously inserted entity-instance has changed or not, and may be updated or not. The same case applies to forms-based manual input.

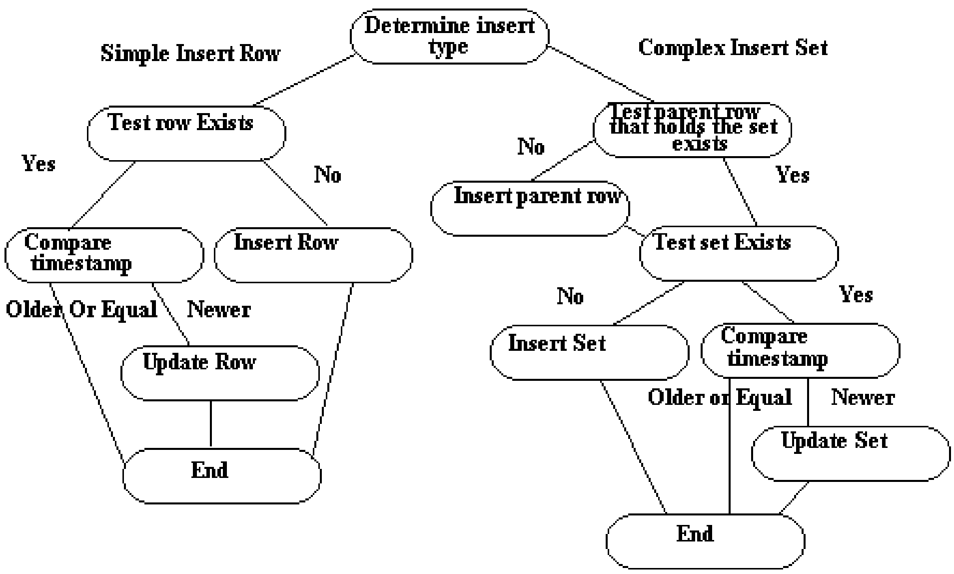


Figure 2. Simplified Chart Core Algorithm Automatical Database Population (indexing robot)

3.3 Components: Overview

A WWW search engine is defined as a retrieval service, consisting basically of a database, search software and a user interface [Pou97]. *UniGuide* has similar components, with some subtle differences. The database (ILLUSTRA ORDBMS), is an object-relational hybrid, with the capability to handle sets, arrays, abstract data types, object identifiers, references, relations, user defined functions, inheritance, rules, etc. [SM96].

The search software is based on SQL3 queries, SQL queries that can call to external C functions (ILLUSTRA API) with the ability to run queries as well ("callback" feature), rules, ILLUSTRA Web Datablade® Applications and Javascripts®. For security reasons queries are "actually" generated on the server-side. Only the generation of the interface and the input/output of data are done on the client-side. The system comprises more than 100 rules used intensively in order to control referential integrity, constraints, uniqueness of sets, automatic actualisation of object references and hypertext links.

3.4 Meta Tags

Our proposal is to include meta tags that simulate to be tuples or rows of data or metadata on a given range of particular webpages. The following figure (Fig. 3) contains two meta-tags representing entity-instances of academic entity publication and academic entity course/degree:

```
<!-- mandatory columns marked with * -->
<!-- Please Enter Values inside ' ' -->
<meta name="academic_entity_publication"
content="
(~ uni_id [*university]= 'Macquarie University' ~),
(~ academic_entity_type [*academic entity type]= 'Department' ~),
(~ academic_entity_name [*academic entity name]= 'Computing' ~),
(~ pub_name [*publication name]= 'MINNI: Micromouse Incorporating Neural
Network Intelligence' ~),
(~ pub_type [publication type]= 'paper' ~),
(~ pub_date [publication date]= '1997' ~),
(~ pub_topics [topics covered]= 'Neural networks, Robotics' ~),
(~ pub_authors [authors]= 'Jondarr Gibb, Len Hamey' ~),
(~ pub_desc [short description]= 'MINNI is a system whereby a back
propagation neural network is used to control the steering of a micromouse
(small robot) in following a straight path.' ~)">


<meta name="academic_entity_course/degree"
content=        "
(~ uni_id [*university]= 'University of Technology Sydney' ~),
(~ academic_entity_type [*academic entity type]= 'Department' ~),
(~ academic_entity_name [*academic entity name]= 'Computer Science' ~),
(~ course_name [*course name]= 'Bachelor of Science' ~),
(~ course_spec [course speciality]= 'Computing Science' ~),
(~ course_type [course type]= 'Undergraduate' ~),
(~ course_degree_type [course degree type]= 'Single' ~),
(~ course_semesters [course semesters]= '6' ~),
(~ course_credits [course credits]= '144' ~),
(~ course_desc [course description]= 'This course aims to provide a sound
education in all aspects of computing for students who intend to make a
career in the profession' ~)">
```

Figure 3. Examples of UniGuide Scheme meta-tags

These meta tags can be generated automatically by the *UniGuide* Meta tag Generator. This may allow a customised indexing robot that indexes only specific meta tags: *UniGuide* scheme meta tags.

4. *UniGuide* from the end-user perspective

4.1 Introduction.

There are three well-defined sub-sections: Submit URL, which allows the user to manually populate the database, Queries, and the Meta Tag Generator, that generates UniGuide Scheme meta tags. We shall describe only the query section.

4.2 Queries

4.2.1 Simple Queries

Entities are grouped hierarchically by domain/sub-domains. When the user clicks on a given entity (left-frame), an HTML form is generated dynamically on the right hand-side (right-frame). The user can specify the range of values to search on the text boxes. The search options are contextual depending of the type of data of the column (i.e. LIKE option is activated only for columns of type text). The output of a query is displayed in a tabular form. Other information includes the SQL query generated and the number of rows affected.

A simple query example follows: "Give me all the information available about Universities located in Sydney and are public"

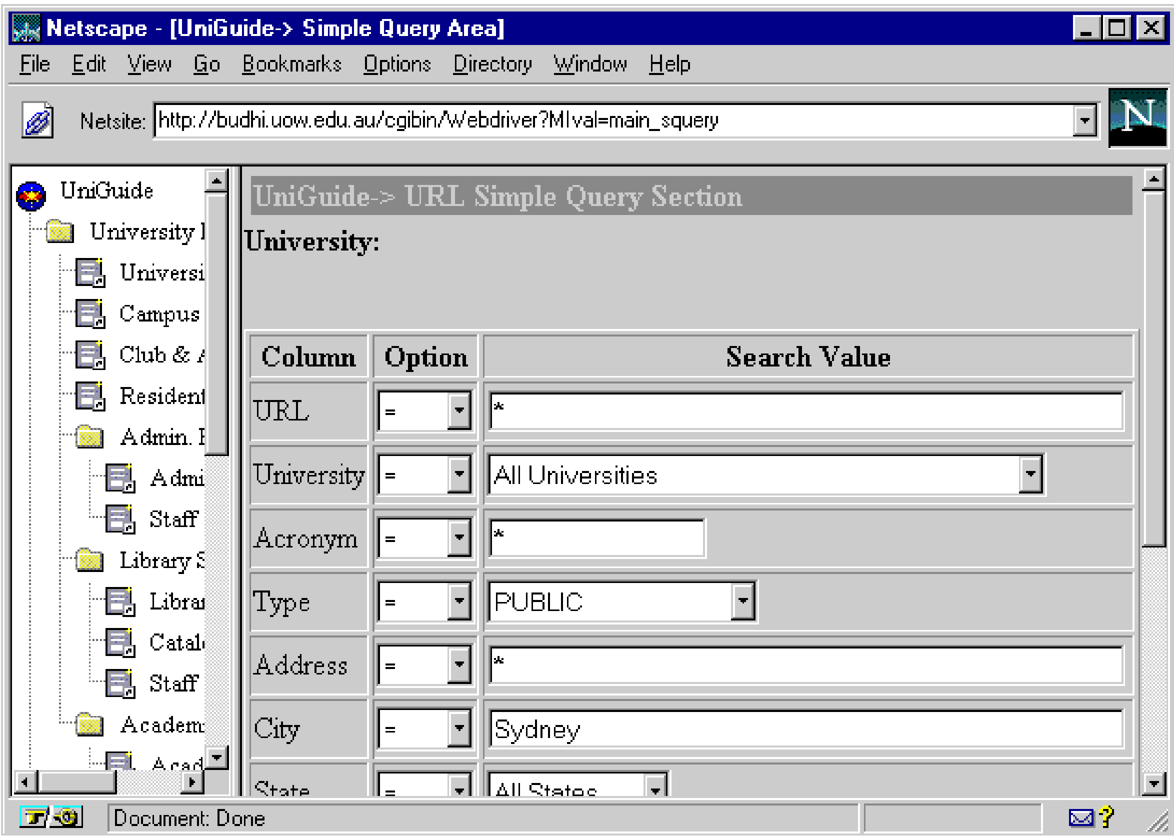


Figure 4: *UniGuide* Simple Query Form Interface. Note: (*) accepts all values.

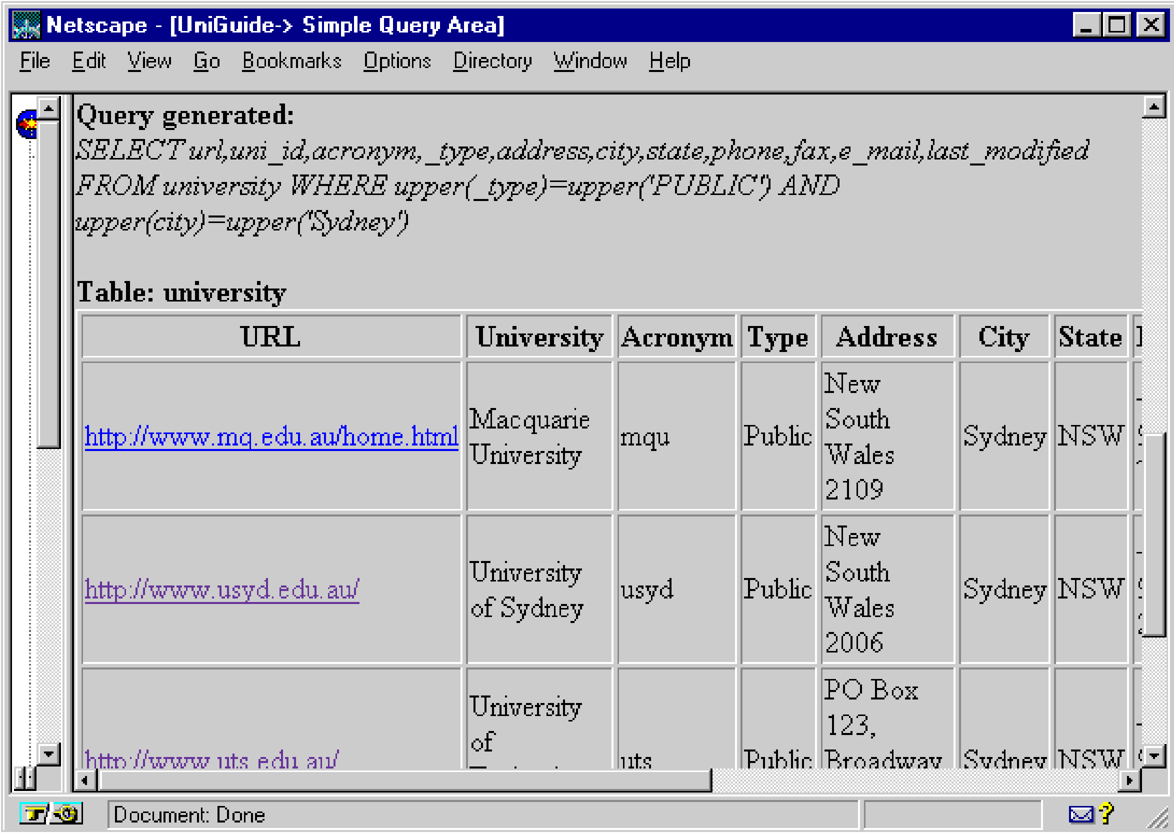


Figure 5: *UniGuide* Simple Query Form Interface: Query Results

Some other examples can be:

- "Give me all the homepages of academic staff members of a particular university who have PhDs and are interested in Artificial Intelligence"
- "Give me all the Masters in Commerce courses offered by Australian Universities"

4.2.2 Predefined Queries

From our viewpoint, predefined queries are complex queries, constructed in a similar way to parameterizable views. Queries can include summarised data, simulation of transitive closure, relations between entities, etc. Some examples of more elaborated queries can be:

- "Give me all the hierarchical structure of the Faculty of Engineering of a particular university (schools within the faculty, departments within schools, etc.)"
- "Give me all the research projects that involve collaboration between two or more academic entities and are funded by a given company"

4.2.3 Configurable Queries: WebQBE and FreeSQL

End-users will configure and customise the required query. Our goal is to provide an interface similar to a typical Query By Example interface. Another option currently implemented is a more complex interface that allows advanced end-users to elaborate free SQL queries with the aid of predefined queries, functions, operators, and a list of tables and columns available in the schema.

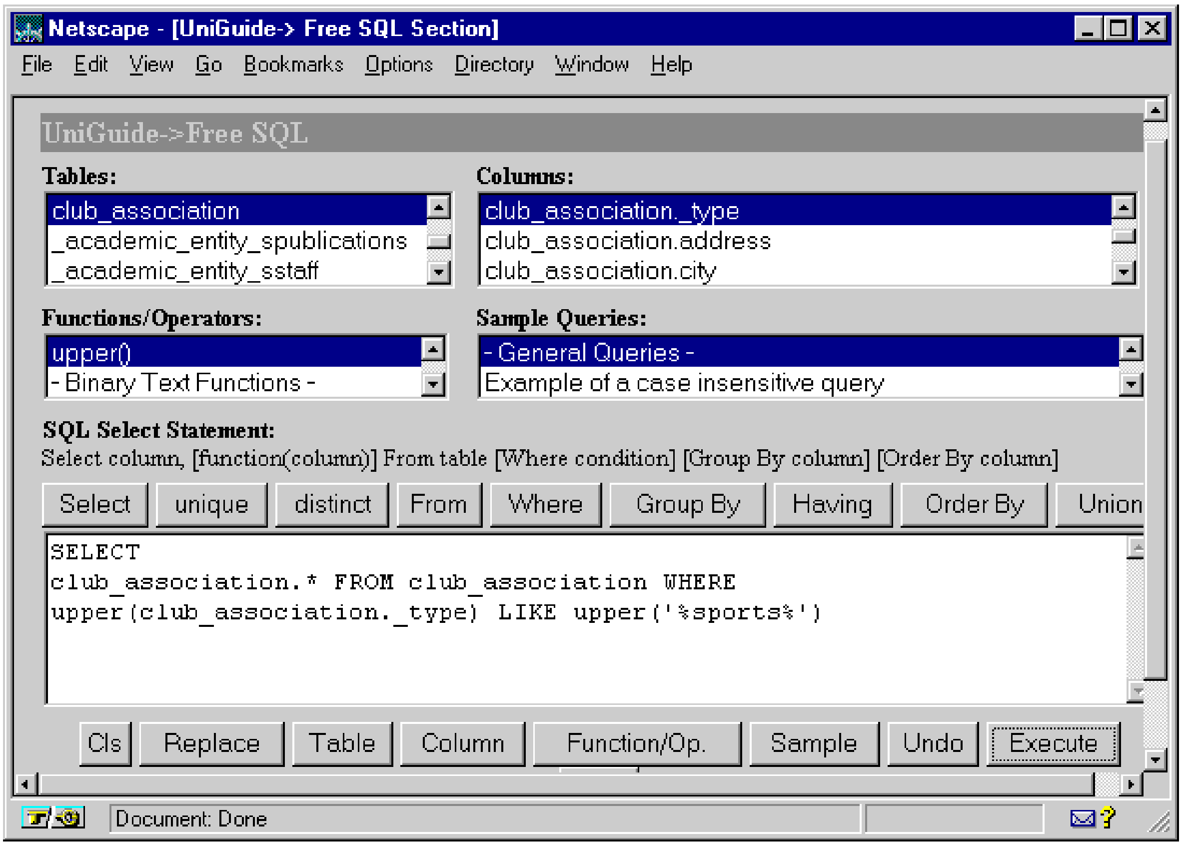


Figure 6: *UniGuide* FreeSQL interface

5. Future Research Directions

Developing an interface for *UniGuide* that fits the needs of both inexperienced and advanced users constitutes a challenge, especially in hiding the complexity of the schema. Also, currently we are developing in JAVA a breadth-first multi-threaded indexing robot that captures UniGuide meta-tags from University domains. Other important issues are whether we should continue providing strict integrity rules to the system (i.e. reject tuples that violate referential integrity, or constraints) or should there be a natural evolution of the system towards "weaker" integrity rules, fuzzy referential integrity etc.

6. Conclusions

A new kind of search engine has been proposed as an alternative to current implementations, with the ability to provide more structured and complex queries. This work is part of an ongoing research program exploring object-relational database approaches to searching the web. The success of this project is partly dependent on the consensus of a given number of Australian Universities to adopt the use of UniGuide Scheme meta tags in order to populate the database automatically. Finally we conclude that although the proposed solution is domain-specific, wherever a model can be "extracted" and a standard can be established respect to metadata, our approach can be customised to adapt to the requirements of that specialised domain (i.e. ideal for large intranets: government departments, large companies, etc.).

7. References

[Abi97] Serge Abiteboul. Querying Semi-Structured Data. ICDT 97 6th International Conference on Database Theory Delphi, Greece, January 8-10, 1997. http://www-db.stanford.edu/~abitebou/pub/icdt97.semistructured.ps

[AMM+97] Paolo Atzeni, Giansalvatore Mecca, Paolo Merialdo, Elena Tabet. Structures in the Web. Technical Report RT-INF-19-1997 Department of Computer Science and Automation. January 1997. http://www.inf.uniroma3.it/tech-rep/inf-19-97.ps

[COR97] Dublin Core Metadata Element Set: Reference Description. October 2, 1997. http://purl.org/metadata/dublin_core_elements

[MH97] Kunhanandha Mahaligan and Michael Huhns. A tool for Organising Web Information, IEEE Computer, June 1997, pages 80-83.

[HZF95] J. Han, O. R. Zaïane, and Y. Fu, " Resource and Knowledge Discovery in Global Information Systems: A Scalable Multiple Layered Database Approach", Proc. Of a Forum on Research and Technology Advances in Digital Libraries (ADL'95), McLean, Virginia, May 1995.

[Pou97] Alan Poulter. The design of World Wide Web search engines: a critical review. Program, vol31 no.2 April 1997, pages 131-145.

[SM96] Michael Stonebraker, Dorothy Moore. Object-Relational DBMSs: The Next Great Wave. Morgan Kaufmann Publishers, Inc 1996

[W3C97] W3C.Hypertext Links in HTML.W3C Working Draft 28-Mar-97 http://www.w3.org/TR/WD-htmllink - meta

The FRAMES Project: Reuse of Video Information using the World Wide Web

C. A. Lindley, B. Simpson-Young, U. Srinivasan
CSIRO Mathematical and Information Sciences
Locked Bag 17, North Ryde NSW 2113, Australia
Phone: +61 2 9325 3107, Fax: +61 2 93253101
Craig.Lindley@cmis.csiro.au
Bill.Simpson-Young@cmis.csiro. au
Uma.Srinivasan@cmis.csiro.au

Abstract

The FRAMES Project within the Advanced Computational Systems Cooperative Research Centre (ACSys CRC) is developing technologies and demonstrators that can be applied to the reuse, or multi-purposing, of video content using the World Wide Web. Video content is modeled in terms of several levels of filmic semantics including a combination of automatically detected video and audio characteristics (eg camera motion, number of speakers) integrated with higher level, authored semantic interpretation models. Active processes for searching semantic descriptions expressed at these different levels can then be accessed either by direct query over the web, or using virtual video prescriptions containing embedded queries that are routed to the FRAMES database system by a virtual video engine.

Introduction

Accessing video content on the web today is limited to watching pre-prepared video clips or programs and, in some cases, having random access within that footage. Random access is normally provided by the use of slider bars but can also be facilitated by hyperlinks from key frames or textual references. The use of such video material is usually restricted to that explicitly intended by the service (ie sequential viewing of prepared video material) and does not facilitate the reuse of this material by other applications and for other purposes.

Issues that need to be addressed to ensure that video resources on the web are reusable include copyright issues, conventions with regard to video metadata, standardised transport and streaming protocols, and protocols for other types of access to the video (eg getting single frames or getting specific macroblocks from a compressed frame of video to assist in motion analysis). Effective modelling of video content is another key requirement supporting reuse by high level description. A lot of recent research has led to the development of commercial tools for similarity-based retrieval of images and video by matching a prototype object or feature specification against similar features of database media components (eg. IBM, 1998). However, the similarity of visual features is not adequate in itself for generating novel and coherent video montage sequences having specific higher level meanings. This paper presents an architecture currently under development that addresses this need by integrating low level feature models with high level models of objects, events, actions, connotations, and subtextual interpretations of video sequences. This type of modelling aims to ensure that the presentation of pieces of raw footage is not restricted to a single program about a single topic but can occur as part of a huge number of different programs over time. This paper gives an example of such reuse in the form of a virtual video generation system that makes use of the models of video content.

Previous work within the ACSYS CRC has developed the Film Researchers Archival Navigation Kit (FRANK) demonstrating remote web-based search and retrieval of video data from video archives (Simpson-Young, 1996). The FRANK system was based on searching and navigating through textual representations of video material, specifically transcripts and shot lists, and providing a tight integration between text-based navigation and the navigation through the video. The FRAMES project within the Advanced Computational Systems Cooperative Research Centre (ACSys CRC) is now developing an experimental environment for video content-based retrieval and dynamic virtual video synthesis using archival video material. The retrieval and synthesis is supported by rich and structured models of video semantics. The generation of dynamic virtual videos is based upon multi-level content descriptions of archived material, together with a specification of the videos that are to be created expressed as a *virtual video prescription* (a document created by an author that provides characteristics and constraints of the virtual video that is to be produced).

This paper describes the semantic metamodel that FRAMES uses and the proposed FRAMES architecture. The paper then describes the processes used to generate dynamic virtual videos based upon prescriptions and content descriptions.

The FRAMES system will initially be trialed using client software at the Sydney site of the CRC (the CSIRO Mathematical and Information Sciences Sydney laboratory) remotely accessing video archives at the Canberra site of the CRC (on the Australian National University campus). In this case, the bandwidth available will be adequate for VHS-quality video. It is expected that the architecture and technology developed in this Extranet setting will later be applied both to low bit-rate video over typical Internet connections and to broadcast quality networked video. The technology may be applied to stand-alone dynamic virtual video systems, to intranet systems accessing internal corporate video databases, or to internet technologies to provide remote and interactive access to on-line video databases.

FRAMES Content Modelling

A video semantics metamodel is the core component of the FRAMES system. Based upon the film semiotics pioneered by the film theorist Christian Metz (1974), we identify five levels of cinematic codification to be represented within the metamodel:

1. the *perceptual level*: the level at which visual phenomena become perceptually meaningful, the level at which distinctions are perceived by a viewer within the perceptual object. This level includes perceptible visual characteristics, such as colours and textures. This level is the subject of a large amount of current research on video content-based retrieval (see Aigrain et al, 1996).
2. the *diegetic level*: at this level the basic perceptual features of an image are organised into the four-dimensional spatio-temporal world posited by a video image or sequence of video images, including the spatiotemporal descriptions of agents, objects, actions, and events that take place within that world. An example of an informal description at this level may be "Delores Death enters the kitchen, takes a gun from the cutlery drawer and puts it into her handbag". This is the "highest" level of video semantics that most research to date has attempted to address, oter than by associating video material with unconstrained text (allowing video to be searched indirectly via text retrieval methods, eg. Srinivasan et al, 1997).
3. the *cinematic level*: the specifics of formal film and video techniques incorporated in the

production of expressive artefacts ("a film", or "a video"). This level includes camera operations (pan, tilt, zoom), lighting schemes, and optical effects. For example, "Low key, hard lighting, CU [Delores Death puts the gun in her handbag]". Automated detection of cinematic features is another area of vigorous current research activity (see Aigrain et al, 1996).

4. the *connotative level*: metaphorical, analogical, and associative meaning that the denoted (ie. diegetic) objects and events of a video may have. The connotative level captures the codes that define the culture of a social group and are considered "natural" within the group. Examples of connotative meanings are the emotions connoted by actions or the expressions on the faces of characters, such as "Delores Death is angry and vengeful", or "Watch out, someones going to get a bullet!".
5. the *subtextual level*: more specialised meanings of symbols and signifiers. Examples might include feminist analyses of the power relationships between characters, or a Jungian analysis of particular characters as representing specific cultural archetypes. For example, "Delores Death violates stereotypical images of the passivity and compliance of women", or "Delores Death is the Murderous Monster Mother".

The connotative and subtextual levels of video semantics have generally been ignored in attempts to represent video semantics to date, despite being a major concern for filmmakers and critics. Modelling "the meaning" of a video, shot, or sequence requires the description of the video object at any or all of the levels described above. The different levels interact, so that, for example, particular cinematic devices can be used to create different connotations or subtextual meanings while dealing with similar diegetic material.

Metamodel components are drawn upon in the FRAMES authoring environment in order to create specific models, or *interpretations*, of specific video segments. There may be more than one model at any particular level, for example, corresponding to different theories of subtext or created by different people. Cinematic and perceptual level descriptions may be generated automatically to an increasing extent. Subtextual and connotative descriptions are necessarily created by hand; in the FRAMES system this will be done using a structured GUI interface to an object relational database. The diegetic level represents an interface between what may be detected automatically and what must be defined manually, with ongoing research addressing the further automation of diegetic modelling (eg. Kim et al, 1998). While low level models can be created automatically to some extent, higher level descriptions created by authors both facilitate and constrain the use of specific video segments in automatically generated dynamic virtual videos. Hence high level annotation should be regarded as an extension of the authoring activities involved in film and video making to the dynamic, interactive environment.

FRAMES Architecture

Figure 3 shows the core FRAMES architecture. From the point of view of reusability of web resources, the major components in this architecture are the query engine and the video server. Each of these can be independently accessed over the Internet for specific applications (eg the video server can be accessed directly for sequential playback of stored video and the query engine can be used for querying a video database returning links to offsets into specific video content). However, these components demonstrate their full power when combined and when used together with applications such as that discussed in the next section.

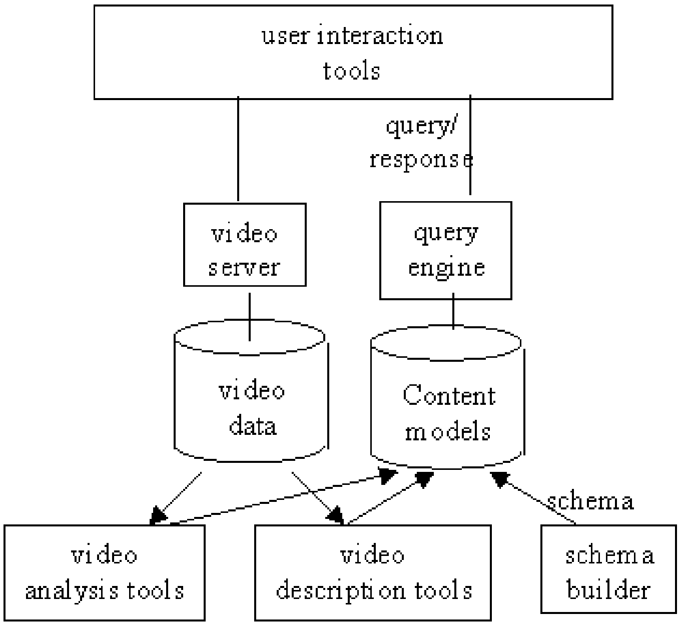


Figure 1. Core FRAMES Architecture.

The query engine is located behind an HTTP server and receives a query (which uses an XML query language) from the client using the HTTP POST method. The engine queries the content models (using methods described in the next section) and returns a list of hits, each of which gives references to specific video content (including location of the footage and start and end offsets) and metadata associated with a particular piece of footage (such as copyright information etc). This list would be in one of the following formats depending on arguments in the request:

a. Presentable data using HTML This would be used if the list is not intended for further automated processing and the browser is unable to display XML appropriately. In this case, video references are given as URLs (including time offsets as query parameters) forming hypertext links. This would be adequate if the user just wants to receive links to appropriate positions within video streams and to click on those links to see the corresponding video from the video server.
b. Structured data using XML This would be used if the list is intended to have further processing or the browser is able to display XML with a corresponding stylesheet that will be referenced. This would be appropriate if the sender of a query was a virtual video generation program or some other application that would want to do further processing of the query results.

The content models themselves are currently implemented as object-relational models under an Object Relational Database Management System, and the query engine uses SQL queries, either directly or to implement an Application Program Interface (API) for the association engine (see next section).

FRAMES Virtual Video Synthesis

Although the query engine and content models can be used directly for search and retrieval of video content, the important point, with regard to reuse of the content, is that it can also be used for a wide variety of other purposes, for example, virtual video generation.

Virtual videos can be specified by an author in the form of virtual video prescriptions (Lindley and Vercoustre, 1998). These are based on the concept of virtual document prescriptions (Vercoustre and Paradis, 1997, Vercoustre et al, 1997). The first version of the FRAMES dynamic virtual video synthesis engine, currently under development, will support three methods by which video segments may be retrieved and inserted within a dynamic virtual video stream:

1. Access by *direct reference* using an explicit, hard-coded reference to a video data file plus start and finish offsets of the required segment (using the CMD language, Simpson-Young and Yap, 1997 or referencing as used by SMIL - Hoschka 1998). While this is a simple, fast, direct, and specific method of accessing video data, it requires knowledge of the exact contents of the data, the data must not change, and it is not robust against changes in the location of the data. This method also does not support dynamic reselection of data based upon parameters passed into a virtual video specification.
2. Access by *parametric match* overcomes these deficiencies. Database queries may also contain complex logical and pattern matching operations. In parametric search, the initial query forms a hard constraint upon the material that is returned. A simple example of a parametric query is a search for a specific character in a particular location (eg. "SELECT ALL sequence.id FROM video_db WHERE character.name == "Delores Death" AND location == "Sydney").
3. Access by *associative chaining* is a less constrained form of accessing video data, where material may be incorporated on the basis of its degree of match to an initial search specification and then incrementally to successive component descriptions in the associative chain. Since association is conducted progressively against descriptors associated with each successive video component, paths may follow semantic chains that progressively deviate from the initial matching criteria. This is the technique used in the Automatist storytelling system demonstrated in the ConText (Davenport and Murtaugh, 1995) and the ConTour (Murtaugh, 1996) systems for generating dynamic evolving documentaries from a changing database of video material. The Automatist system uses simple keyword descriptors specified by authors and associated with relatively "self-contained" video segments. The FRAMES system extends this approach by using the highly structured semantic model described above, which allows much greater descrimination on descriptor types, and more specific forms of relationship between sequenced video components. Associative chaining starts with specific parameters that are progressively substituted as the chain develops. For example, "ASSOC_CHAIN FROM video_db; character.name = "Delores Death" AND location = "Sydney"" will begin with the character Delores Death in Sydney (if present in the database), but may quickly progress to another character in Sydney, or to Delores Deaths adventures in Melbourne, etc..

Specific filmic structures and forms can be generated in FRAMES by using particular description structures, association criteria and constraints. In this way the sequencing mechanisms remain generic, with emphasis shifting to the authoring of metamodels, interpretations, and specifications for the creation of specific types of dynamic virtual video productions.

Figure 2 shows the FRAMES architecture with the additional of the virtual video engine.

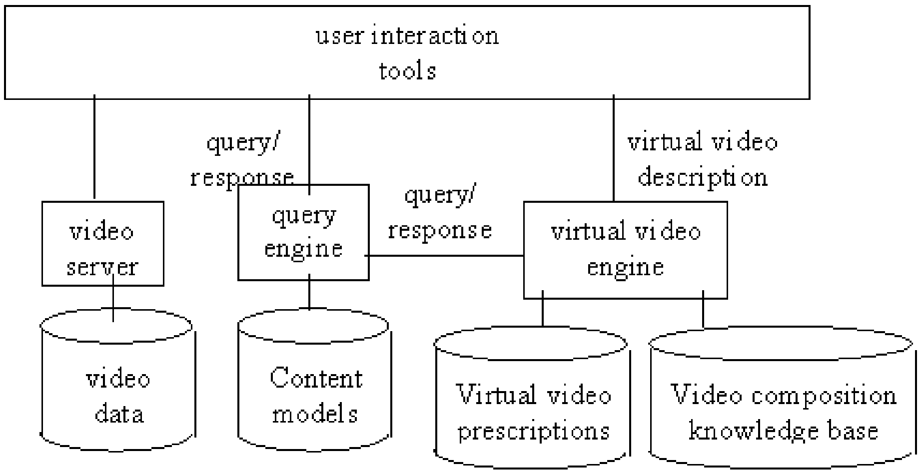


Figure 2. FRAMES Architecture showing virtual video production..

The virtual video engine is also located behind an HTTP server. It receives a set of parameters from an HTTP POST message, selects a virtual video prescription accordingly and resolves the prescription by satisfying the constraints given by the parameters. There are two general cases of virtual video generation: a) non-interactive virtual videos; and b) interactive virtual videos. In the case of non-interactive virtual videos, the virtual video prescription is fully resolved into a virtual video description (Lindley and Vercoustre, 1998). The virtual video description takes the form of an XML document that is sent back to the client. Virtual video descriptions use the same XML DTD as virtual video prescriptions but are a special case in that all queries have been resolved to specific video data addresses. For interactive virtual videos, a users interactions dictate the direction the video takes (as in Davenport and Murtaugh, 1995), so each user interaction results in new requests being made resulting in virtual video description chunks being continually sent to the client.

Conclusion

For video content on the web to be reusable, there need to be rich models of the video content at several levels of semantic codification, and these models need to be accessible on the web independently of the video content itself. We have discussed the FRAMES system which uses an architecture that facilitates reuse of video data by using mulit-level semantic models for the generation of virtual videos from underlying databases and video archives. This architecture operates through the world wide web, allowing video material to be incorporated into new and dynamic productions from diverse and distributed sources.

References

Aigrain P., Zhang H., and Petkovic D. 1996 "Content-Based Representation and Retrieval of Visual Media: A State-of-the-Art Review", *Multimedia Tools and Applications* 3, 179-202, Klewer Academic Publishers, The Netherlands.

Davenport G. and Murtaugh M. 1995 "ConText: Towards the Evolving Documentary" Proceedings, ACM Multimedia, San Francisco, California, Nov. 5-11.

IBM 1998 QBIC Home Page, http://wwwqbic.almaden.ibm.com/stage/index.html.

Hoschka P.(ed) 1998, "Synchronised Multimedia Integration Language" W3C Working Draft 2-February 1998 (work in progress).

Kim M., Choi J. G. and Lee M. H. 1998 "Localising Moving Objects in Image Sequences Using a Statistical Hypothesis Test", Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, Churchill, Victoria, 9-11 Feb., 836-841.

Lindley C. A. and Vercoustre A. M. 1998 "Intelligent Video Synthesis Using Virtual Video Prescriptions", Proceedings of the International Conference on Computational Intelligence and Multimedia Applications, Churchill, Victoria, 9-11 Feb., 661-666.

Metz C. 1974 *Film Language: A Semiotics of the Cinema*, trans. by M. Taylor, The University of Chicago Press.

Murtaugh M. 1996 *The Automatist Storytelling System*, Masters Thesis, MIT Media Lab, Massachusetts Institute of Technology.

Simpson-Young B. 1996 "Web meets FRANK: Tightly Integrating Web and Network Video Functionality", AusWeb'96, Gold Coast, July '96.
http://www.scu.edu.au/ausweb96/tech/simpson-young

Simpson-Young B. and Yap, K. 1996 "FRANK: Trialing a system for remote navigation of film archives", *SPIE International Symposium on Voice, Video and Data Communications*, Boston, 18-22 November.

Simpson-Young B. and Yap K. 1997 "An open continuous media environment on the web", AusWeb-97.

Srinivasan U., Gu L., Tsui K., and Simpson-Young W. G. "A Data Model to Support Content-Based Search in Digital Videos", submitted to the *Australian Computing Journal*, 1997.

Vercoustre A-M. and Paradis F. 1997 "A Descriptive Language for Information Object Reuse through Virtual Documents", in *4th International Conference on Object-Oriented Information Systems (OOIS'97)*, Brisbane, Australia, pp299-311, 10-12 November, 1997.

Vercoustre A-M., Dell'Oro J. and Hills B., 1997 Reuse of Information through Virtual Documents, Proc. of the *2d. ADCS Symposium*, Melbourne, Australia, April 1997.

A Personal Evolvable Advisor for WWW Knowledge-Based Systems

M.Montebello, W.A.Gray, S.Hurley
Computer Science Department
University of Wales, Cardiff.
email: (m.montebello,w.a.gray,s.hurley)@cs.cf.ac.uk

Abstract

The immense size of the distributed WWW knowledge-base and the dramatic rapid increase in the volume of data on the Internet, requires techniques and tools that reduce users' information overload and improve the effectiveness of online information access. Despite the potential benefits of existing indexing, retrieving and searching techniques in assisting users in the browsing process, little has been done to ensure that the information presented is of a high recall and precision standard. In this position paper we present a system that reuses the information generated from search engines together with previously developed systems, and adapts it, by generating user profiles, to better meet the needs and interests of the users by improving recall and precision measures.

Introduction - Background and Motivations

In recent years there has been a well-publicized explosion of information available on the Internet, and a corresponding increase in usage. This is particularly true of the World-Wide Web (WWW) [Berners-Lee et al., 1994] and its associated browsers which allow relative easy access to the information available, and thus make it accessible to a wider audience. The WWW is a major knowledge dissemination system that makes the world's staggering wealth of knowledge and experience, stored on server machines scattered across the internet, accessible to the on-line world.

When people access the web, they are either searching for specific information, or they are simply browsing, looking for something new or interesting (often referred to as *surfing*). The WWW's sheer scale and its exponential growth renders the task of simply finding information, tucked away in some Web site, laborious, tedious, long-winded and time consuming. The fact that a user's time is valuable and that relevant information might not be accessed, imposes serious restrictions on the efficient use of the WWW and the benefits that users can expect from their interaction.

It is well documented that traditional search engines provide services which are far from satisfactory [DeBra and Post, 1994,Spetka, 1994,Srinivasan et al., 1996]. Users are faced with the problem of these search engines being too generalised and not focused enough to their real and specific needs. This triggered further research to develop more sophisticated techniques and agent like systems that make use of the user profile to personalise the service they provide and add value to the information they presented [Pazzani et al., 1996,Green and Edwards, 1996,Mladenic, 1996,Pazzani et al., 1996].

The Personal Evolvable Advisor (PEA), presented in this position paper, is a system we have developed to reuse information generated by search engines and utilise previously developed retrieval systems. Conceptually, the PEA is similar to a meta-search engine, but with the major difference that it employs user profiling to specifically target documents for individual users. In this way duplication and redundancy of information is significantly reduced, while the real needs and interests of the users are fully addressed in a more focussed retrieval.

PEA - Current Implementation

Our goal with PEA is to achieve a high recall and high precision performance score on the information presented to the user. Recall measures how efficient the system is at retrieving the relevant documents from the WWW, while precision measures the relevance of the retrieved set of documents to the user requirements. In order to obtain a high recall execution we make use of the hits returned by a number of traditional search engines together with the output from retrieval systems that have been previously developed. The reason for doing this is twofold. Firstly, we could have developed our own search engine and argued that it utilises the ultimate retrieval techniques and produced results similar to other systems. However, by making use of what other systems generate, we ensure that we obtain all the information that all of them would retrieve at the same time, and not have the problem of developing an ultimate system. Secondly, there are numerous WWW crawlers available, bombarding servers and clogging networks. By using them, we simply use other systems' knowledge-bases, rather than duplicating it, and move up to the next level of the information ''food chain'' [Selberg and Etzioni, 1995], in this way our recall is as good as can be achieved with any current system. On the other hand in order to add value to the retrieved results and maximise the precision and efficiency with which the system achieves high recall scores, we generate user profiles to predict and suggest the most suitable information for specific users. Through various interactions the system will be able to optimize the targeting and predicting of what users are interested in, thereby improving the precision factor of the retrieved information.

The processes required by an information retrieval and filtering system include several tasks that PEA decomposes into a number of simpler tasks. Figure 1 shows the major components of the system: the WWW and the external systems at the bottom level, the underlying application software on the next level up, and the GUI at the top.

The WWW is one of the components over which we have no control. It requires no local development, but its heterogeneous, unstructured and uncensored nature causes developers

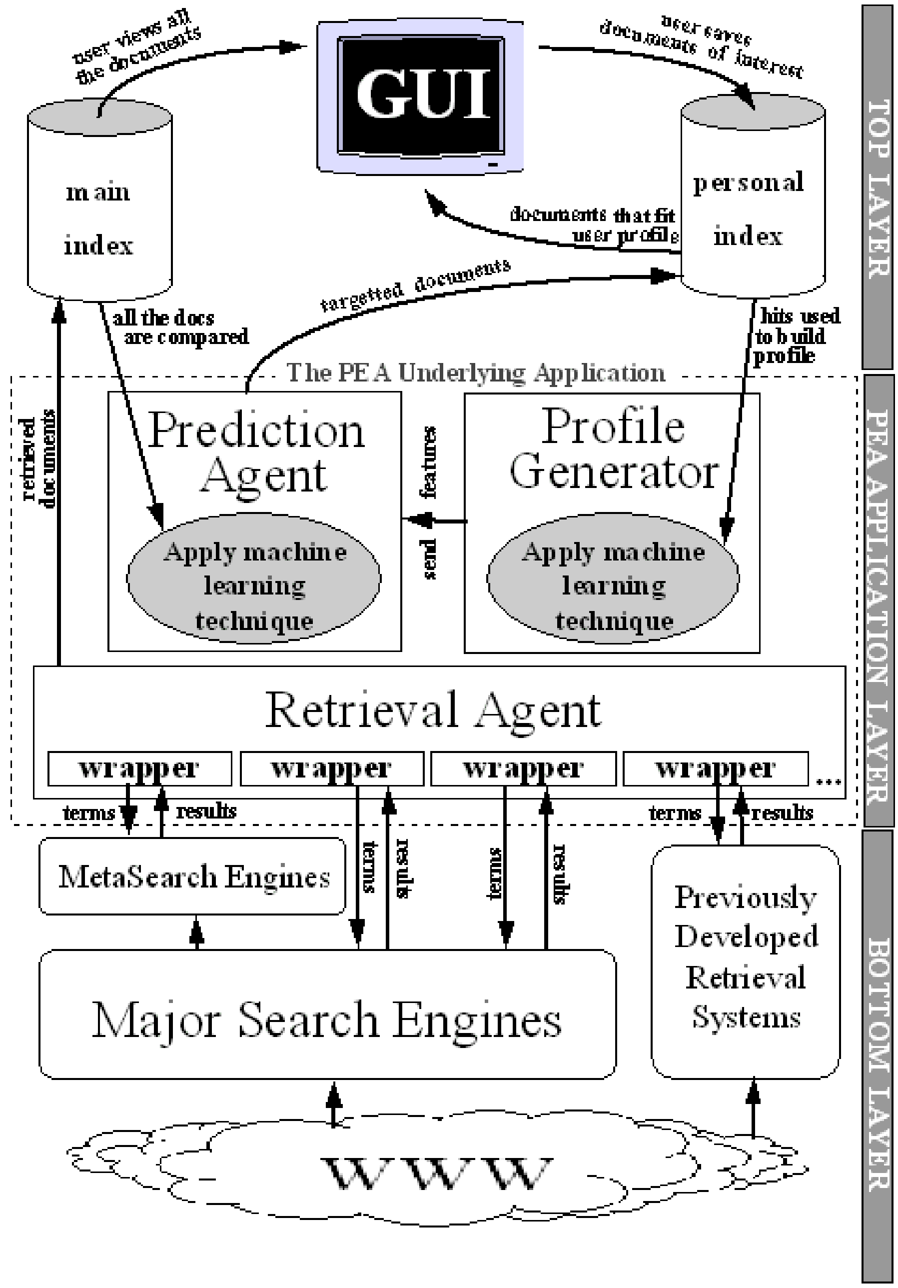


Figure 1: PEA Architecture

to face awkward coding situations in order to be able to cater for all the kinds of data found on the WWW. The WWW can be assumed to be a very large heterogeneous distributed digital information database. In order to optimize the management and exploit the potential of the WWW's vast knowledge-base we require to search and retrieve efficiently and effectively specific information for users.

The external systems utilised include some of the major search engines and also some other retrieving systems that have been developed by other research groups[Eichmann and Wu, 1996,Mladenic, 1996,Selberg and Etzioni, 1995]. They use the WWW as their source of input and we use their output as the input for PEA. All the external systems are considered to be black boxes and action is taken upon the information they output. Wrappers are used to manage the appropriate and proper handshaking between the diverse search engines and the other retrieving systems and the application layer.

Query terms are used to locate documents and retrieve results from the external systems. These results need substantial re-formatting as they usually include completely useless information like advertisements, local links and site specific information.

The underlying application layer has the difficult task of performing all the work required, transparently from the user. It makes use of the information retrieved from the external systems and attempts to improve on the recall/precision metrics mentioned earlier. By using state-of-the-art external systems we attempt to achieve a high recall rate, while using a personalised profile with specific interests for each user, we attempt to also achieve a high precision rate.

The three main components of the PEA underlying application layer (retrieval agent, profile generator and prediction agent - Figure 1) perform the necessary work to satisfy our initial motivations.

The Retrieval Agent

The retrieval agent, is responsible for aggregating all the hits returned by the external systems. It collates the results, by removing duplicates and ensuring integrity, and stores the formatted and pre-ranked results as a single list in a local database, known as the *main index*. The Java programming language was employed to develop this part of the application due to its ease in performing TCP/IP connections to allow retrieval of documents and their processing. This agent interacts with the external systems via appropriate wrappers. Every query term is employed by the wrapper which will command the associated system to locate documents from its local index and return related results. These results are basically a series of document addresses (URLs - Universal Resource Locator) which are listed within an HTML page that the external system returns. A scan through the WWW page will quickly identify the URL links and list them. Some of the links are useless to the user, so the retrieval agent initially removes adverts, duplicates, and site specific links. It then analyses the vetted URLs and accesses the document on-line. This will identify whether the link is still accessible, has moved or been removed completely. If the document is valid, then an initial paragraph from the document is extracted and saved locally in the main database index together with the reference search term, its reference within the index, the URL, and the document title. All these details will be available to the user through the GUI, and also to the prediction agent to identify if the particular document is relevant to a particular user or not.

The Profile Generator

The task that the profile generator sets out to achieve is to analyse each users' personal index and generate a profile. If users have different interests stored in their personal index, then a separate profile is required and generated for each interest. No novel machine learning technique has been developed for the profile generator. It uses specific techniques previously employed by other similar systems [Edwards et al., 1995,Green and Edwards, 1996,Payne and Edwards, 1997]. The difference is that users are able to select which technique they would like to use to generate their profile, and predict other relevant documents in future interactions. This profile generation utilizes the *term frequency/inverse document frequency* machine learning technique[Salton and McGill, 1983], but other machine learning techniques are being implemented. Profile generating systems like MAGI and UNA were relatively easy because specific and fixed fields were provided in the data they were extracting information from. Documents like email and USENET news articles have inherently static field holders embedded in them, e.g. ‘‘to’’, ‘‘from’’, ‘‘date’’, and ‘‘subject’’. These are typical examples of anchored features a developer can rely on when designing the filtering procedures. On the other hand, when considering how to perform the same task on WWW documents (normally HTML), no fixed fields are provided. Even though HTML version 3 introduced the META tag, which allows authors to specify indexing information; it is unreliable as authors can fail to use it. A developer cannot assume that HTML document authors abide by standard conventional fields within documents e.g. ‘‘ `<HTML>`’’, ‘‘`<TITLE>`’’, and ‘‘`<BODY>`’’, due to the weak typing nature of HTML. Despite this, there are many systems that filter HTML documents e.g. WebHunter [Lashkari, 1995], LIRA [Balabanovic and Shoham, 1995], Letizia [Lieberman, 1995], WebWatcher [Armstrong et al., 1995][Joachims et al., 1997], SULLA [Eichmann and Wu, 1996], Personal WebWatcher [Mladenic, 1996], and others described in [Etzioni and Weld, 1994][Holte and Drummond, 1994] and [Perkowitz and Etzioni, 1995].

We assume that normally, when searching or even browsing, a user bookmarks a page of interest and proceeds with the activity he/she was performing. Taking this activity into perspective, all that is required is to take into consideration what the user bookmarks, and utilise this information to generate the profile. While this method may have problems of over identification, it is more reliable than asking users to assign ratings, as it is less demanding on the user’s time. Another problem that many of these HTML filtering systems ignore is that machine learning techniques have a slow learning curve and require a sufficient number of examples before they can make accurate predictions. As a result a profile generator encounters problems when dealing with completely new situations. Generally this is true for all such systems and as [Maes and Kozierok, 1993] rightly argue, the user and the profile agent will gradually build up a trust relationship over time. Issues regarding how many profiles to generate for a user - one specific profile per user, a general profile for a group of users, different profiles for different users or different profiles for the same users - have been tackled differently. Some profile generators develop the ‘specific user profile’, especially those systems which have been produced to cater for specific items like emails or newsgroups, while others specialise in a ‘specific topic profile’, like WebFind [Monge and Elkan, 1995], MetaCrawler [Selberg and Etzioni, 1995], PAINT [Oostendorp et al., 1994], and CURRY [Krishnamurthy and Tsangaris, 1996] which recommend documents to users with the same interests or needs. Other systems, like Syskill and Webert [Pazzani et al., 1996], learn a separate profile for each topic of each user. They argue that many users have multiple interests and it will be possible to learn a more accurate profile for each topic separately since the factors that make one topic interesting are unlikely to make another interesting. We take this argument one step further, and argue that what one user finds interesting in a specific topic, differs from what another user describes as interesting about the same topic. Therefore, different profiles need to be generated for every different interest a user has if the predicted results are to be focused accurately.

The Prediction Agent

The user interest profile generated by the profile generator will be used by the prediction agent in combination with the extracted features from documents in order to predict and suggest new interesting documents to a user. Documents that have been retrieved and stored within the main index by the retrieval agent will have their features extracted and compared to the profile of each individual user generated by the profile generator. This is performed on every item a user has shown interest in, and if any of the documents from the main index happen to fit the user’s interests or needs, then they will be eventually suggested to the user the next time the user logs in (Figure 2). Each suggestion, if considered interesting, may be explicitly added to the personal database by the user, or deleted completely. The user might even prefer that he/she is notified, via email, that documents of interest have been located. The machine learning techniques employed to generate the user profile is also applied to extract features from documents. In this way the targetted documents reflect, and are consistent, with the specific user profile generated.

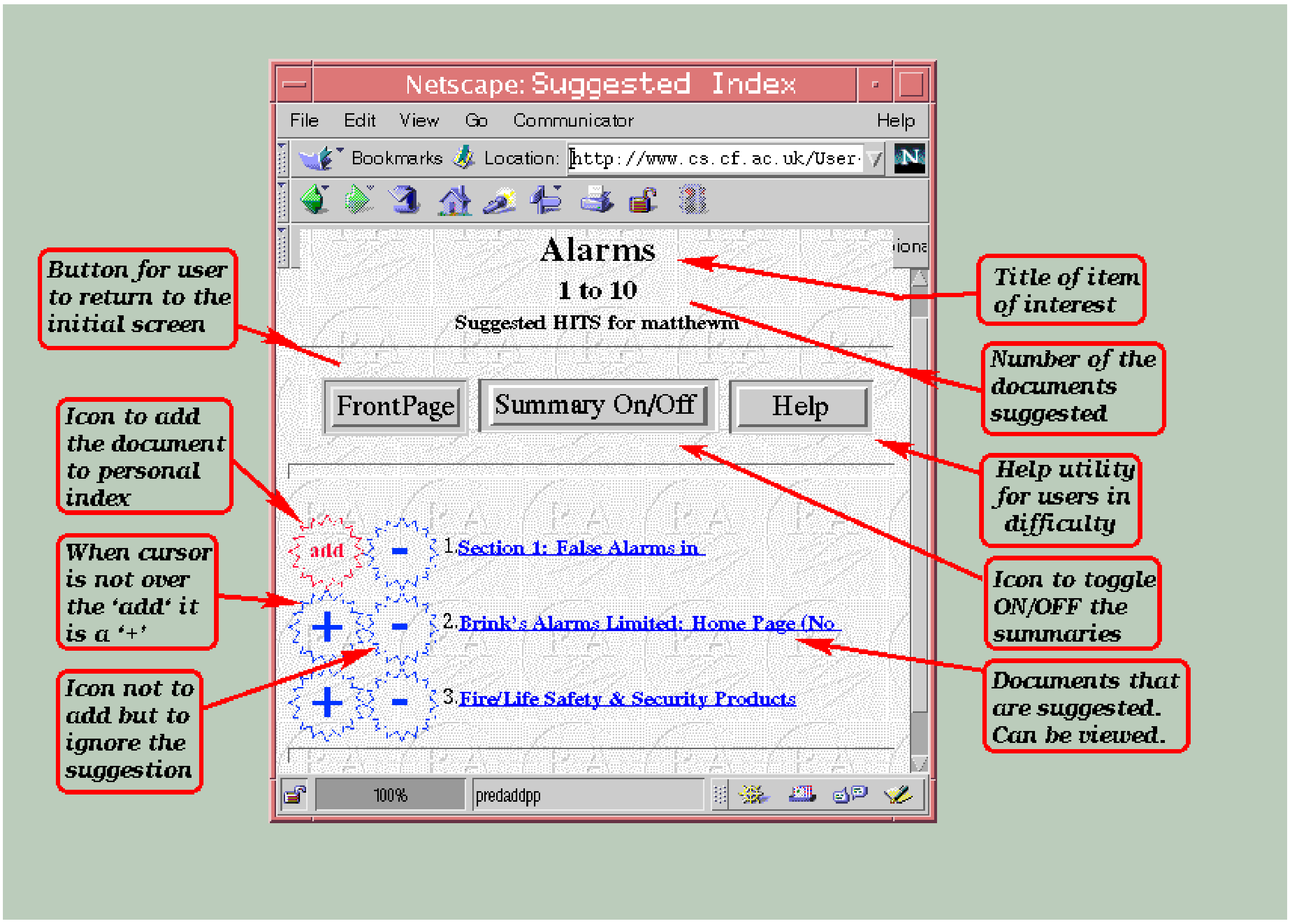


Figure 2: Suggested Documents

Evolvability

One final point to make is about the evolvability of PEA. The choice of external systems coupled with the appropriate wrapper within the retrieval agent, and the machine learning technique employed in the application layer can be selected from a list of systems and techniques incorporated in the system. PEA allows this list to be amended and hence other systems and techniques that might be developed in the future can be easily incorporated. The use of available systems means the system evolves as they evolve, relatively easily. The only amendments to PEA being if a new wrapper is required.

PEA GUI - An Example

PEA requires an administrator to manage the general needs and demands of a specific interest group of users. Search terms tailored to any type of interest group can be initialised by the administrator and furthermore users will be able to suggest any other terms to add to the main search list. Documents relevant to the specific area of interest are retrieved and stored by the underlying application within the main index, and when a user logs-in he/she is able to benefit from the systems' high recall fidelity. Having analysed the documents, individual users can bookmark and highlight specific items as interesting and appealing. These will be saved inside their personal database index. At this stage the underlying application plays another important role in attaining precise targeting of documents to individual users by generating a profile from the personal database index and predicting other documents from within the main index. Users can decide to add the suggested documents to their personal database index or remove them completely. As new and suggested documents are entered in the personal database index the user profile becomes more focused and finely tuned, as a result of which higher precision results will be achieved.

Related Work

Several research systems and commercial off-the-shelf agents have been developed which are similar to our work, but the closest systems are the so called metasearch engines. [Selberg and Etzioni, 1995] fed search terms to six major search engines and made use of the outcome within the MetaCrawler system. A number of front-end metasearch engines have also been introduced on the market, among them Surfbot, WebCompass, WebFerret, and WebSeeker. These all have very similar capabilities to the MetaCrawler plus additional features such as monitoring specific documents, verifying links and providing relevance ranking. All these related systems are only as reliable as the search engines that they depend upon. This means that their recall score might be very high because a search is done on many of the most popular search engines on the WWW with a single command, potentially retrieving all possible indexed documents. On the other hand, their low precision factor will require the users to check through the documents returned by the metasearchers to identify which ones are of interest. In our system, this task is performed by the profile/prediction components within the underlying application, which combines the optimization of both recall and precision.

Concluding Comments

In this position paper we have presented a system, PEA, that adds value to the information traditional search engines and other metasearch engines generate from the WWW. We argue that by reusing the information output from several retrieving/indexing systems we ensure a high recall score, while generating a specific user profile to predict and target other documents to specific users, we also ensure a high precision score. Users are able to select their own profile generator/prediction agent from a number of alternatives, reflecting different machine learning techniques employed. New techniques can be integrated into this evolvable system by the system administrator, who can also easily maintain the system's resources and update the search terms specific to a user group. In the future we will be investigating the integration of other machine learning techniques that have been developed and employed by other systems. This will help us to evaluate which technique is best suited to cater for the needs of different users. Evaluation of the recall/precision scores is also required to ensure that value is added to the normal services provided by the search engines and the meta-search engines. This will be done by analysing the feedback given from a group of users who are presently making use of the system and who will eventually assess the extent to which the information presented is of high recall/precision quality.

References

Armstrong et al., 1995
Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. (1995).
WebWatcher: A Leraning Apprentice for the World Wide Web.
AAAI Spring Symposeum on Information Gathering.

Balabanovic and Shoham, 1995
Balabanovic, M. and Shoham, Y. (1995).
Learning Information Retrieval Agents: Experiments with Automated Web Browsing.
AAAI Spring Symposeum on Information Gathering.

Berners-Lee et al., 1994
Berners-Lee, T., Caillian, R., Luotonen, A., Nielsen, H. F., and Secret, A. (1994).
The World-Wide Web.
Communications of the ACM, 37(8):76-82.

DeBra and Post, 1994
DeBra, P. M. E. and Post, R. D. J. (1994).
Searching for arbitrary information in the WWW: the fish-search for mosaic.
In *Proceedings of the 2nd. international world wide web conference*.

Edwards et al., 1995
Edwards, P., Bayer, D., Green, C. L., and Payne, T. R. (1995).
Experience with Learning Agents which Manage Internet-Based Information.
In *Proceedings of ML95 Workshop on Agents that Learn from Other Agents*.

Eichmann and Wu, 1996
Eichmann, D. and Wu, J. (1996).
Sulla - a user agent for the web.
In *5th. International World Wide Web Conference*.

Etzioni and Weld, 1994
Etzioni, O. and Weld, D. S. (1994).
A Softbot-Based Interface to the Internet.
Communications of the ACM, 37(7):72-79.

Green and Edwards, 1996
Green, C. L. and Edwards, P. (1996).
Using Machine Learning to enhance software tools for internet information management.
In Franz, A. and Kitamo, H., editors, *AAAI-96, Workshop on Internet-Based Information Systems*, pages 48-55. AAAI Press.

Holte and Drummond, 1994
Holte, R. and Drummond, C. (1994).

A Learning Apprentice for Browsing.
AAAI Spring Symposeum on Software Agents.

Joachims et al., 1997
Joachims, T., Mitchell, T., and Freitag, D. (1997).
WebWatcher: A Tour Guide for the World Wide Web.
IJCAI97.

Krishnamurthy and Tsangaris, 1996
Krishnamurthy, B. and Tsangaris, M. (1996).
Curry: A customizable url recommendation repository.
In *5th. International World Wide Web Conference*.

Lashkari, 1995
Lashkari, Y. (1995).
Feature Guided Automated Collaborative Filtering.
Master's thesis, MIT, department of Media Arts and Sciences.

Lieberman, 1995
Lieberman, H. (1995).
Letizia: An Agent that assists Web Browsing.
In *International Joint Conference on Artificial Intelligence*.

Maes and Kozierok, 1993
Maes, P. and Kozierok, R. (1993).
Learning Interface Agents.
In *Proceedings of the 11th National Conference on Artificial Intelligence*, pages 450-465.

Mladenic, 1996
Mladenic, D. (1996).
Personal WebWatcher: Implementation and Design.
Technical Report IJS-DP-7472, J Stefan Institute, Llubljana, Slovenia.

Monge and Elkan, 1995
Monge, A. E. and Elkan, C. P. (1995).
Integrating external information sources to guide Worldwide Web Information Retrieval.
Technical Report CS96-474, University of California, San Diego.

Oostendorp et al., 1994
Oostendorp, K. A., Punch, W. F., and Wiggins, R. W. (1994).
A tool for individualizing the Web.
In *Proceedings of the 2nd. WWW conference '94: Mosaic and the Web*.

Payne and Edwards, 1997
Payne, T. R. and Edwards, P. (1997).
Interface Agents that Learn: An Investigation of Learning Issues in a Mail Agent Interface.
Applied Artificial Intelligence, 11(1):1-32.

Pazzani et al., 1996
Pazzani, M., Muramatsu, J., and Billsus, D. (1996).
Syskill and Webert: Identifying Interesting Web Sites.
AAAI Conference.

Perkowitz and Etzioni, 1995
Perkowitz, M. and Etzioni, O. (1995).
Category Translation: Learning to understand Information on the Internet.
In *International Joint Conference on Artificial Intelligence*.

Salton and McGill, 1983
Salton, G. and McGill, M. J. (1983).
Introduction to Modern Information Retrieval.
McGraw-Hill.

Selberg and Etzioni, 1995
Selberg, E. and Etzioni, O. (1995).
Multi-service search and comparison using the meta-crawler.
The Web Revolution. Proceedings of the 4th. international world wide web conference.

Spetka, 1994
Spetka, S. (1994).
The TkWWW Robot: Beyond browsing.
In *Proceedings of the 2nd. WWW conference '94: Mosaic and the Web*.

Srinivasan et al., 1996
Srinivasan, P., Ruiz, M. E., and Lam, W. (1996).
An investigation of indexing on the www.
ASIS '96 Annual Meeting of American Society for Information Science., 33:79-83.

Footnotes

...Systems
Research funded by the Radiocommunications Agency, UK.

...Surfbot
http://www.surflogic.com/

...WebCompass
http://www.quarterdeck.com/qdeck/products/webcompass/

...WebFerret
http://www.webferret.com/netferret/webferret.htm

...WebSeeker
http://www.ffg.com/seeker/

Transforming the Internet into a Database

Anand Rajaraman

Junglee Corporation

anand@junglee.com

1. Introduction

Virtual database (VDB) technology makes the World Wide Web and other external data sources behave as an extension of an enterprise's relational database (RDBMS) system. According to some estimates, as much as 90% of the world's data is outside of relational database systems. Vital data is scattered across web sites, file systems, database systems, and legacy applications. These data sources differ in the way they organize the data, in the vocabulary they use, and in their data-access mechanisms. Many of them do not even support native query operations. Writing applications that combine data from these sources is a complex, often impossible, task because of the heterogeneity involved.

VDB technology lets applications ask powerful SQL queries of data that is scattered over a variety of data sources. The VDB gathers, structures and integrates the data from these disparate data sources and provides the application programmer with the appearance of a single, unified relational database system. VDB technology enables the development of an exciting new breed of applications that use *all* the data.

As an illustration of the applications enabled by VDB technology, consider job hunting on the Web. In order to make a meaningful career choice, a job seeker needs information on available opportunities as well as related data - such as information on housing, school districts, and crime statistics in the job area. Information on job openings is scattered across thousands of different web sites - company home pages and several aggregate sites, such as newspaper classifieds sites. Keyword search capabilities on words appearing in the job listing are the only available search choice.

VDB technology converts all these data sources into a single virtual relational database. Using an application based on VDB technology, the job seeker can now obtain answers to the following query posed to the Web, "find marketing manager positions in a company that is within 15 miles of San Francisco and whose stock price has been growing at a rate of at least 25% per year over the last three years". This single query would span the Web employment listings of many corporations, in addition to web sites that have geographical mapping information and websites that contain historical records of corporate equity prices. The query would also return, for each position, related information including statistics on housing prices, school districts, and crime statistics. Section 3 provides details on this and other VDB applications that are deployed on several high-traffic web sites, including those of *Yahoo!, The Wall Street Journal, The Washington Post*, and *The San Jose Mercury News.*

VDB technology is particularly effective in enterprise settings when combined with data warehousing. Using VDB technology, corporations can include data from nontraditional sources (e.g., files in directory systems) and external sources (e.g., the World Wide Web) in the data warehouse, enabling key decision support applications such as data mining and online analytical processing (OLAP).

2. Technology Architecture

2.1 The Virtual Database

Figure 1 is a run-time view of a simple Virtual Database (VDB), which we'll call the *Books VDB* for future reference. This VDB integrates the contents of two bookstores (Amazon.com and Powell's Books) and the New York Times Book Reviews and presents a unified schema with two tables, books and reviews. The database application operates on this unified schema, issuing SQL queries through the JDBC or ODBC API; the application itself is built using standard RAD tools such as Delphi, PowerBuilder, Visual Basic, or similar Java toolkits. In Figure 1, the application issues the standard SQL query:

```
SELECT *

FROM books, reviews

WHERE books.author = "gardner" AND books.isbn = reviews.isbn
```

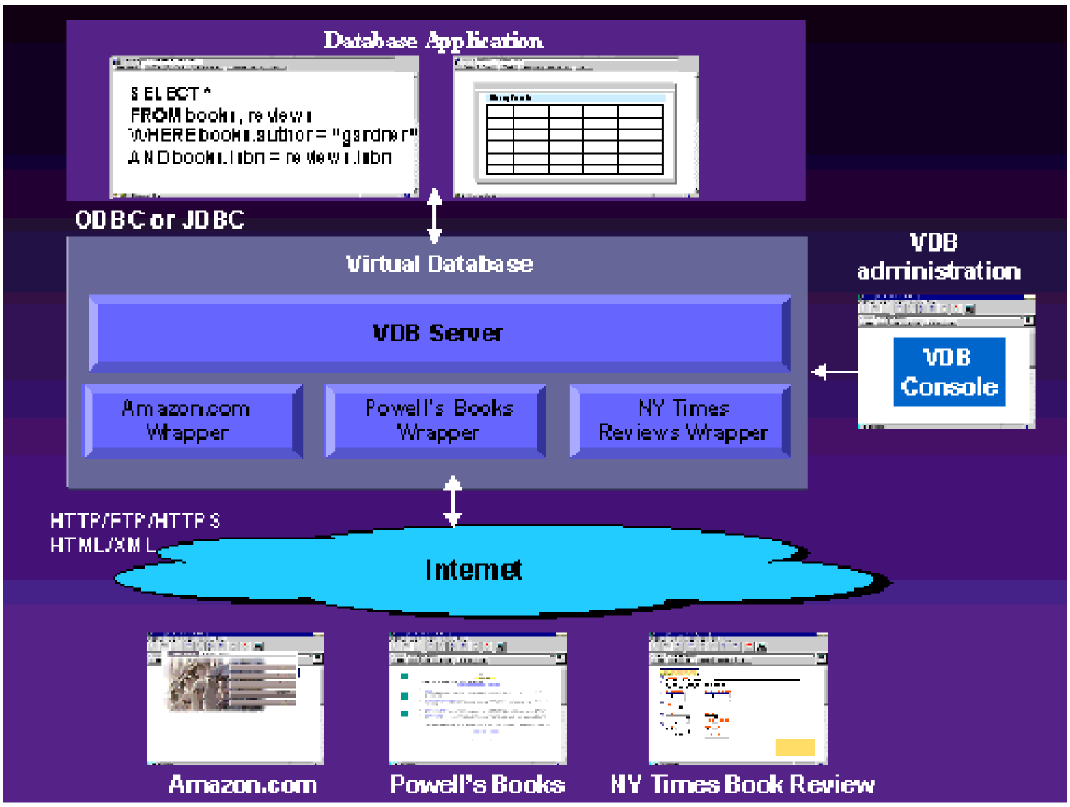


Figure 1. The Books Virtual Database.

The VDB is accessed through the VDB Server, and is administered through the browser-based VDB Console. The VDB also contains, for each external data source, a *wrapper* that interfaces the data source to the VDB server. A wrapper makes an arbitrary external data source, such as a web site, behave like an

RDBMS, while the VDB Server integrates these separate relational databases into a unified Virtual Database (VDB).

Figure 2 shows an individual wrapper in action. The wrapper interfaces with the web site, typically using HTTP and HTML. It handles HTTP protocol-related issues such as forms, cookies, and authentication. The wrapper is accessed via the JDBC API, through which clients can issue SQL queries. A SQL query issued to the wrapper in this case would result in the wrapper filling out a HTML form on the Amazon.com web site, navigating and parsing the resulting HTML pages, and transforming the data into rows in a relational table. The wrapper uses *extraction rules* that apply sophisticated linguistic processing to extract attributes from the web pages, *data transformation* rules to map and format the data to fit the schema, and *data validation* rules to ensure data integrity.

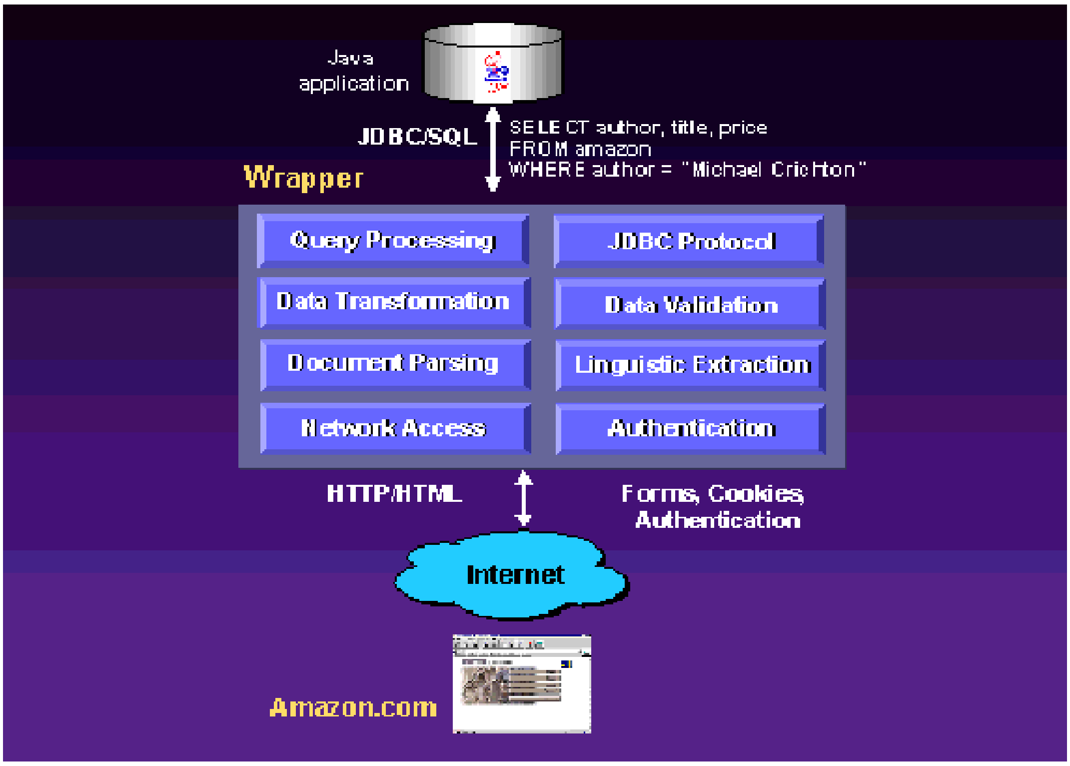


Figure 2. A standalone wrapper.

Lightweight Java applications that interact with one (or a few) data sources can interface directly with wrappers. The application sees each data source as a separate JDBC source with its own schema, and must connect to each source separately and combine the data as needed.

Sophisticated applications that use more than a few data sources use the full functionality of the VDBMS, as shown in Figure 3. The VDBMS exposes tables in multiple data sources as *virtual tables* in a single Virtual Database (VDB), and supports full RDBMS functionality over virtual tables including view definitions and query processing across sources. In the example of Figure 3, the VDB defines the view *books* as the union of the *amazon* and *powell's* virtual tables. When the VDBMS receives the query shown in the figure, the *query processor* component decomposes the query, determines the fragments to be sent down to the individual data sources, and combines their results. The *query result cache* caches results from data sources for performance.

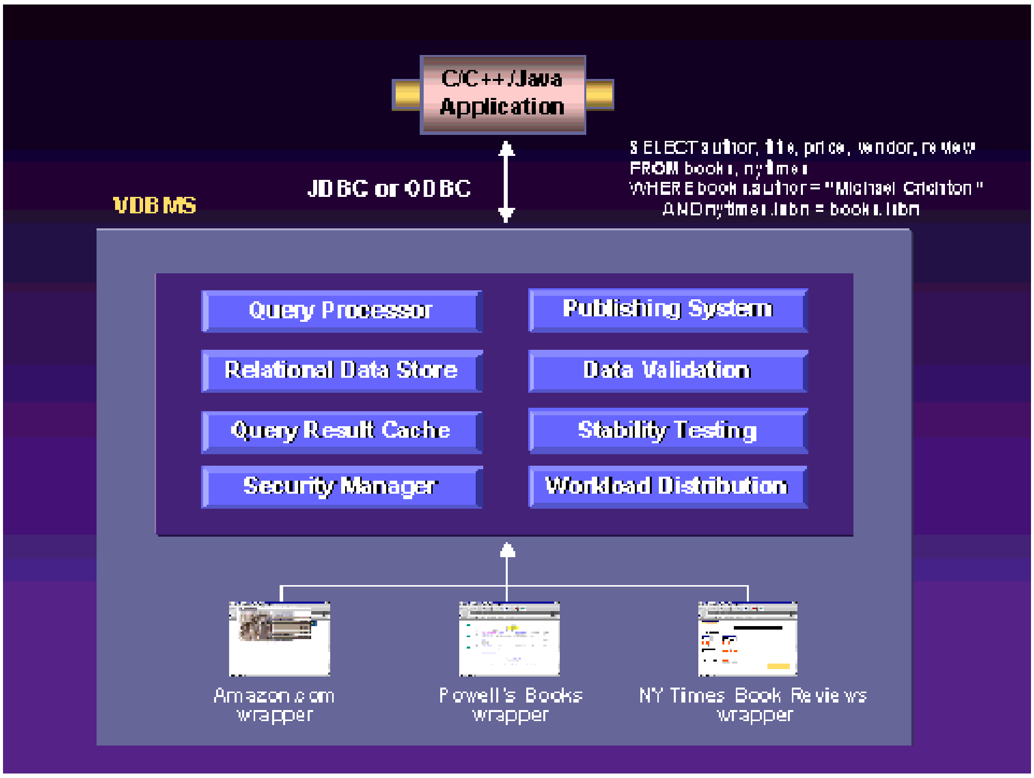


Figure 3. From standalone wrappers to an integrated Virtual Database

In addition, the *publishing system* can be set up to periodically create physical snapshots of virtual tables in a local *relational data store*. The VDBMS can also perform data validation tests that are more sophisticated than those at individual wrappers; an example is *stability testing,* which compares data against historical statistical trends and raises an alert if there is a large deviation.

2.2 The Virtual Database Management System (VDBMS)

Junglee's Virtual Database Management System (VDBMS) enables the creation and management of Virtual Databases such as the Books VDB. Figure 4 shows the components of the VDBMS.

Wrapper Development Kit (WDK).

The Wrapper Development Kit enables the rapid creation of wrappers for web sites, file systems, and other network data sources. The WDK is built around the notion of *wrapper frameworks*. A wrapper framework is a collection of classes and a programming idiom that together ease the creation of wrappers for a family of data sources. For example, the *Navigator* framework makes it simple to create wrappers for web sites: a few simple lines of Java code can capture the structure of a complicated web site, including sites whose pages are dynamically generated in response to filled out forms. This framework also captures relationships between hyperlinked pages in a web site and reflects the relationships in the contents of the virtual database tables corresponding to the site.

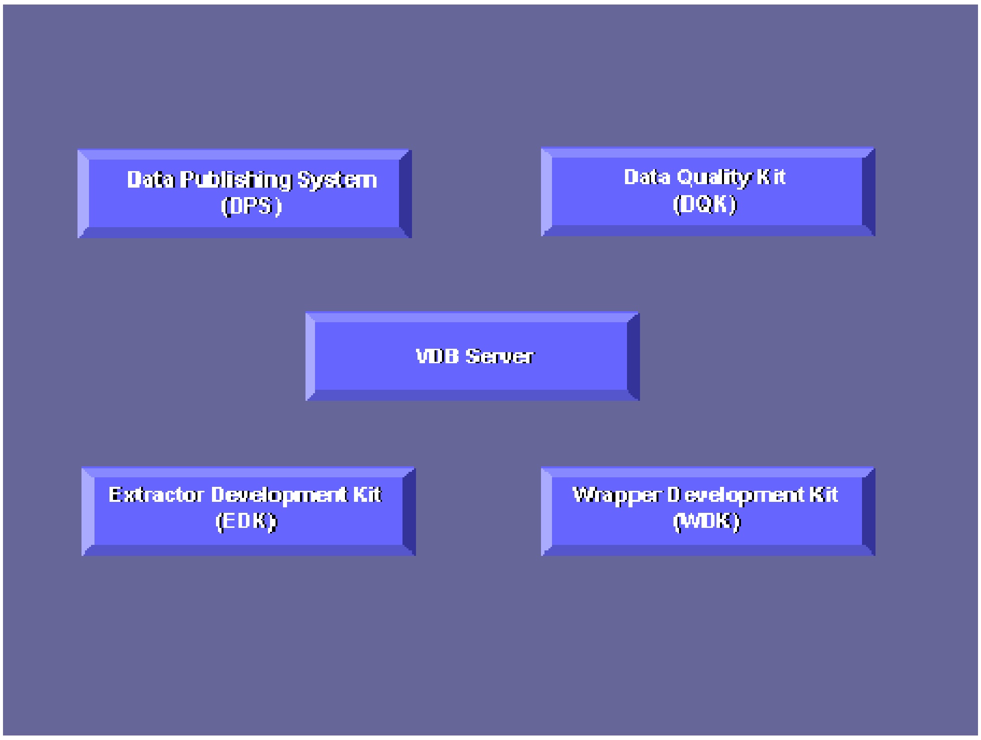


Figure 4. VDBMS components

Extractor Development Kit (EDK).

Data integration often involves extracting structure from unstructured textual data. For example, consider a newspaper web site that lists apartments for rent. The application needs a table with columns for features such as number of bedrooms, number of bathrooms, location, and rent. However, each apartment classified listing is typically a block of undifferentiated text. Extraction rules describe how to extract the required features from the text.

The *Extractor Engine* is an interpreter for Jel (Junglee Extraction Language), a language designed to express sophisticated text processing rules. Extraction rules rely on dictionaries of words and phrases with attributes. For example, a location extraction rule may use a dictionary that lists the names of cities and states in the US, together with common abbreviations for the names. The EDK includes *Dictionary Management Utilities* to create and manage such dictionaries.

Data Quality Kit (DQK).

The DQK deals with *data transformation* (also called *mapping)* and *data validation*.

Wrappers can make arbitrary data sources behave like relational data tables, but these tables are likely to have inconsistent schemas and vocabularies. As a simple example, consider a scenario where some wrappers export *salary* in *$/month*, others export it in *$/week,* and the application wants it in *$/year*. The required attribute name and unit conversions are handled by a *field mapping*. *Row mappings* create new table columns based on the values of other columns.

Virtual databases often deal with highly irregular data from sources outside the control of the VDB administrator, and subject to large-scale changes without notice. Therefore, ensuring data integrity is a key issue for VDBs. The DQK provides several varieties of data validation checks to ensure data integrity.

VDB Server.

The VDB Server exposes tables in multiple data sources as *virtual tables* in a single Virtual Database (VDB). The VDB Server supports full RDBMS functionality over virtual tables, including view definitions and query processing across sources. The VDB Server supports the JDBC and ODBC APIs and is administered via the browser-based *VDB Console.* It includes a relational data store, which is a commercial-strength RDBMS. The data store can be used to warehouse snapshots of virtual tables for rapid access, and also to store other physical data tables that are used by the database application. The VDB Server implements a security model that hides the details of authentication with individual data sources.

Data Publishing System (DPS)..

The DPS allows administrators to use virtual tables in two modes: *dynamic* and *warehoused.* In dynamic mode, a query to a virtual table results in one or more queries to the underlying data sources. In warehoused mode, the DPS is set up to periodically refresh a snapshot of the virtual table created in the local relational data store, and queries to the table go to the local store. A VDB can contain both kinds of tables, and a single query can span dynamic and warehoused data.

Warehousing is appropriate for slowly-varying data that changes on a daily or weekly basis, while dynamic querying is appropriate for rapidly-varying data, such as stock quotes, or when the data cannot be warehoused because of copyright restrictions. For example, the Books VDB might set up the Amazon and Powell's tables to be dynamically queried (since bookstores tend to change prices at unpredictable intervals) and the NY Times Reviews data to be warehoused weekly in the local data store.

3. The VDB At Work: Real-World Applications

Junglee has applied VDB technology in several key domains: Employment Classifieds, Consumer Shopping, Real Estate, and Apartment listings. The Canopy family of products uses VDB technology to integrate data from a potentially unlimited number of World Wide Web sites, enabling consumers to harness the power of the Web and make more informed decisions.

3.1 Online Recruitment

The *JobCanopy* VDB application integrates job listings from over 700 data sources, including employer web sites, flat files, and legacy data feeds. The schema for this VDB includes 31 attributes of interest to employers and jobseekers, including job title, job category, job location, and contact information. The data sources are scoured each week to ensure that the information is always fresh. Listings from different employers are normalized to have the same set of fields and the same vocabulary. The JobCanopy product is accessible from the web sites of several major newspapers and online media companies, including The Wall Street Journal Interactive Edition, The Washington Post, The San Jose Mercury News, Classifieds2000, and Westech Virtual Job Fair.

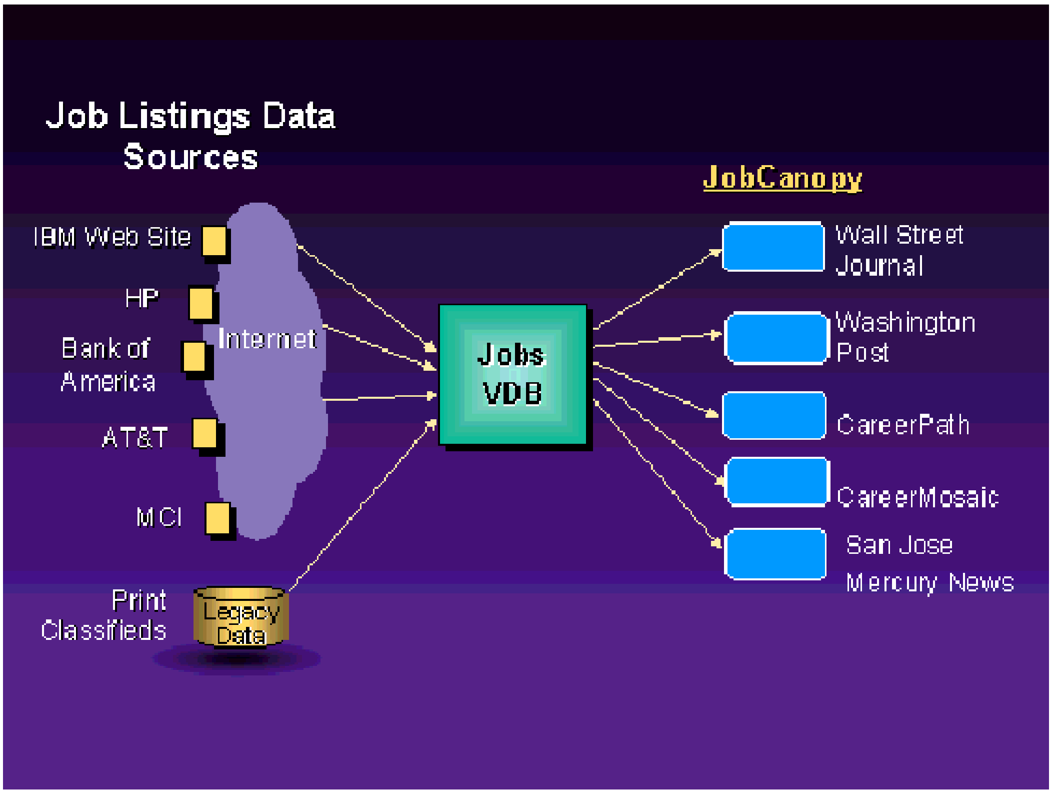


Figure 5. JobCanopy Application

3.2 Web Commerce

The *ShopCanopy* VDB application allows comparison shopping over 40 merchants in 8 categories, including Books, Music, Computer Hardware, and Consumer Electronics. ShopCanopy is deployed on the Yahoo! Visa Shopping Guide web site.

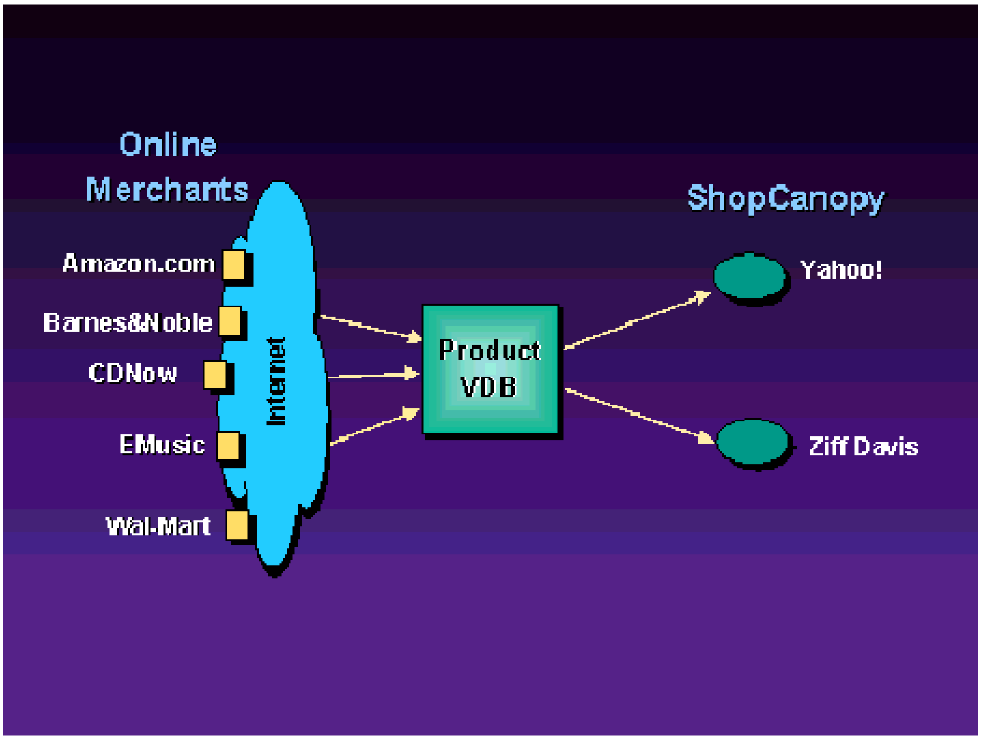


Figure 6. ShopCanopy Application

The ShopCanopy application brings together buyers and sellers online to create marketplaces on the Web. ShopCanopy allows consumers to easily access and compare product and pricing information from merchants simultaneously, and then link to a specific merchant's site to make a purchase. For example, individuals looking for a specific book can go to the book category on the *VISA ShoppingGuide by Yahoo!*, type a book title, such as *Runaway Jury*, in the title field, and in seconds see a list of more than 25 versions of the popular John Grisham novel, offered by several different Internet book stores within the guide, in paperback, hard cover, or audio, with prices from $4.79 to $26.95. Users can also see at a glance which merchants have the book in stock. After making a selection, buying the book is just a click away.

4. Conclusion

VDB technology enables rapid deployment of applications with at least one of the following characteristics:

- Large numbers of data sources
- Data sources are autonomous, i.e., there is no centralized control
- Data sources can have a mixture of structured and unstructured data

The World Wide Web, and most Intranets, have all these characteristics. VDB technology converts the Internet into a database and transforms the World Wide Web.

Typing Concepts for the Web as a Basis for Re-use

Max Mühlhäuser, Ralf Hauber, Theodorich Kopetzky
Department of Telecooperation, Johannes Kepler University Linz, A-4040 Linz, Austria
e-mail: (max|ralf|theo)@tk.uni-linz.ac.at

Abstract: nodes and links i.e. HTML documents and URLs are the basic building blocks of the Web. Much has been done to enrich the structure, re-usability, and functionality of these individual building blocks. But *the* Web can also be regarded as a collection of loosely connected set of rather self-contained hyperwebs. Few efforts have concentrated on augmenting the power of hyperwebs; we believe that typing concepts for such hyperwebs are a key to re-use, structure, and usefulness of Web-based information. The GUTS approach for hyperweb typing will be introduced in this paper, its value for the re-use of Web-based information will be discussed.

Motivation: Web Advancements Overemphasize the "Atoms"

Nodes and links (together with anchors) are the atomic constituents of hypermedia documents - for the Web, the most important such constituents are HTML pages (or HTML documents) and URLs. "The Web" can be regarded as a world-wide coherent set of such nodes and links, growing almost in anarchy without rules. This paper concentrates on a structural entity that lies in between the atomic constituents and the global transitive closure that we call "the Web": a meaningful, coherent set of nodes and links. Before hypermedia went global with the Web, the notion of hypermedia documents as self-contained sets of nodes and links was much more common than the idea of links that would lead from one such document to another (note that, e.g., the term "Web site" refers not only to a *physical* location i.e. computer in the Internet where "some nodes and links" reside: quite often a "site" is viewed as a *logically* correlated set of nodes and links, too). We will refer to such a logically coherent set of nodes and links as "a hyperweb" in the remainder, in contrast to "the Web" as the set of all hyperwebs (note that hyperwebs may contain hyperwebs in a hierarchical manner).

In the context of the workshop, we want to advocate typing concepts for hyperwebs which cover both structural and logical aspects. We believe that the re-use of Web information can be sufficiently improved using such concepts, where re-use may refer to the issue of finding information by query or navigation or to the issue of incorporation of Web information into augmented hypermedia; by the latter we mean systems which are built on top of hypermedia-based structured information (such as, for instance, hypermedia-based learning systems, software engineering environments, decision support systems, and many more).

Hypertext Re-Use and Typing Concepts

A) Atoms: Typing for the atomic constituents of hypermedia has been around for quite some time. Even in HTML, there is a primitive form of implicit typing. For nodes, such implicit typing is given via media type and file format (cf. different graphics and video file types); for links, it is given via target node types (local/remote document, "mailto:", etc.). In the context of semantic networks, it is common to classify the atoms into (usually disjunct) node and link classes; such classification can be viewed as an explicit, user-defined variant of the implicit typing described before. Attributes - supported in many hypermedia systems - may be viewed as another primitive form of typing; in HTML, the <meta> tag can provides a means for defining attribute names and values. More advanced hypermedia systems support user-defined types of nodes and links; for some of them, typing is based on the object-oriented paradigm i.e. associated with particular, type-specific operations (methods). User-defined types are often used for modeling the application domain. In open hypertext systems, finally, typing of anchors is crucial for integrating applications (think of, e.g., email subjects, calendar entries, source code lines, etc.).

Benefits for Re-Use: What are the benefits - in the re-use context - if the atomic constituents of hypermedia are augmented by such typing concepts? As the current emphasis of the Web community on meta data indicates, types serve for classification of elements and facilitate indexing and retrieval of information (and hence, re-use). Typing in the programming language sense (e.g., the object-oriented typing mentioned above) helps to integrate nodes and links into larger contexts in a seamless and consistent way. As a simple example, one might imagine a Web-wide definition of node types which visualize the dynamics of a part of the Internet (performance monitors, visualizations of communication protocols etc.). Any Internet user might then re-use such nodes by combining them into a comprehensive monitor since operations like ?set_update_interval? or ?start_visualisation? were uniquely specified.

B) Hyperwebs: On the atomic level of nodes and links, authors and users of hypertext are concerned with individual units of information (nodes) and individual "hops" between adjacent such units. The hyperweb level adresses the more holisitic question of how to "glue together" (or, respectively, how to consume) a set of nodes which "belong together". On this level, node types and link types are assembled into meaningful larger structures. A primitive typing concept for the hyperweb level can be imagined to consist of classifications of nodes and links plus triples "(node-class)--<link-class>à (node-class)", describing how different classes of nodes may be interconnected via (different classes of) links. A next step of sophistication might be the one of entity-relationship-diagrams, where single and multiple in- and outbound connections are discerned. Yet more sophistication than this is desired: as to the "glueing together", hyperweb types should provide means for specifying the "building plan" for the corresponding hyperweb types. This includes "chains" of nodes and links, several links types originating from the same node type, and many other local and non-local rules and constraints. With respect to re-use, it must be possible to map hyperweb types onto existing parts of in the Web, see below.

Benefits for Re-Use: Again, the question of benefits with respect to re-use may be raised. These benefits fall into two different categories:

1. Re-Use of Hyperweb Types: whatever "knowledge" becomes materialized in a hyperweb type can be re-used each time it is "inherited" by a hyperweb instance.
2. Re-Use of Nodes and Links: hyperweb instances may not only be authored from scratch; they may also become "overlays" for existing nodes and links, just like meta data may be added to HTML pages.

Major benefits are listed below:

- Consistent Structure of the Hyperweb. For instance, authors of learning material (who might re-use information found in the Web) might be "forced" to accompany each concept (node) with at least two examples (nodes of type example)
- Common Look & Feel. The re-use of hyperweb types leads to recurring structures; e.g., an organization may enforce a certain hyperweb style for the personal sites (i.e. the individual home pages plus the hyperwebs which originate from there) of its employees, thereby making all such sites look "familiar"; this reduces cognitive overhead for anyone browsing through these pages
- Improved Indexing and Retrieval. Hypertext queries combine structural aspects (of hyperweb structures) with content aspects (relating to contents of nodes). Respective systems cannot rely on any type information. Meta data for nodes and links may in the future improve precision with respect to contents, but this precision can be much further improved if the user who formulates a query can make assumptions about the structural aspects.

- Improved Sophistication of Applications. The "knowledge" materialized in hyperweb types may relate to application functionality. E.g., learning systems may rely on instructional strategies which prescribe navigational patterns to be used as a learner "travels" through a hyperweb, compiling the learning material. Corresponding navigation rules, updates of the "learner model" etc. may be expressed as part of the hyperweb, instead of being "programmed into" each individual hyperweb instance i.e. learning material. Re-Using the hyperweb type considerably reduces the authoring effort required. This effect relates not only to the re-use of non instantiated hyperweb types, but also to nodes and links which have been "overlaid" by hyperweb instances for micro-level instructional aspects. Such micro-level hyperwebs may be re-used in different learning material.
- Consistent and Easy Incorporation into Composite Hyperwebs. If different hyperwebs follow the same construction principle and offer common functionality (as defined in the corresponding hyperweb type) then their common re-use is obviously much easier. This aspect will be elaborated in the example given in chapter 4.

Related Work

Up to now, the Web community has concentrated on augmenting the power of nodes and links rather than the power of hyperwebs. Many observations back this claim, some of which are listed below:

- HTML as a markup language has a large number of tag and element types which concern the internal structure of individual pages i.e. nodes, but only few which concern hyperwebs (URLs in essence)
- There is no standard on the Web for describing a hyperweb
- Most Web authoring tools are not much more than word processors (which "happen to" use HTML as their output format), often just augmented with the notion of URLs; exceptions like NetObject Fusion™ support site management much more in the "physical" sense mentioned above than in the "logical" sense.
- Metadata, too, reflect logical characteristics of node contents (such as author, language, etc.) much more than characteristics of hyperwebs.

While the Web (considered as a particular kind of hypermedia system) is by far the most wide-spread such system, even a de-facto-standard, it is by far *not* the most advanced system. As a consequence, several of the most interesting contributions to the field have been made for other hypermedia systems than the Web. A short excerpt of such contributions - independent of whether they are Web-compliant (cf. [MM97]) or not - is given below. These contributions relate to fields as diverse as hypermedia authoring/design support, generics and dynamics in hypermedia, database schema approaches, and structural queries.

- DeVise [GHM_94] is a hypermedia system built at the University of Aarhus which complies with the so-called Dexter Model, an architecture which clearly separates (node) contents from (hyperweb) structure. DeVise supports "composites" as a higher-level abstraction for a set of nodes and links; as to typing support, composite types specify different variants of composites (as to what it is that the components have in common) and restrict the types of components that are valid; they do not, however, specify in detail "how to put together" "how many" components "of what type" - we will call these aspects "construction", "restriction" and "element typing").
- Trellis [FS89] and [SF89] is an approach for mapping the navigational behavior of hyperwebs onto PetriNets. It was the first approach to addressing the aspects "construction" and "element typing". However, since the approach was based on the so-called backus-naur-form BNF used

to express programming language syntax, Trellis could only specify linear parts of hyperweb structures, not graphs.

- The object-oriented hypermedia design method OOHDM [SRB96] is an attempt to map nodes and links onto objects and relations; OOHDM emphasizes "element typing" in the above-mentioned terms: object-oriented design concepts are used for expressing *types* of objects (i.e. classes) and relations; the construction / restriction aspects mentioned above are rather neglected.
- All approaches mentioned above were centered around structured development of hypertext documents. As to structured information acquisition, one has to consider WWW query systems in particular. Systems such as [WebSQL]suffer from the lack of structural information available in hyperwebs. WebSQL and others can only offer nodes, links, and anchors as the principle constituents of a hyperweb: it cannot be expected that any typing information about nodes and links is available on the Web in general, let alone that a common type system be used, let a alone that structural information about the corresponding hyperweb is available. While it is most likely too optimistic to expect the use of common hyperweb types throughout the Web, it is both desirable and imaginable that organizations and "communities" may adhere to such hyperweb types.
- The HyperTree approach [STH97] has many goals in common with the approach described here. HyperTree makes a distinction between the graph-like structure of an actual hyperweb and a tree-shaped conceptual level "above" this graph. We regard a single conceptual level as too limiting and believe that tree-like structures can only be regarded as a special case of hypermedia. In addition, HyperTree does not attempt to introduce common type systems together with corresponding semantics for specific application domains.
- PreScripts [Richartz96] were developed with the same goals as the approach described here but were restricted to hyperweb types (i.e. did not support consistent types and semantics). The PreScript approach was only slightly related to the Web. The approach presented here takes over many ideas from PreScripts but adds Web compliance, many detailed enhancements, and the concept of ontologies as described further below.
- Along with the recent advancements around HTML, meta-data come into focus. The W3C note about XML-data [Lay98] and the TR-WD about RDF [LaS98] point into the direction mentioned as "triples" under B) in the last section; apart from the restricted functionality of such approaches as discussed, the ones mentioned here are intended for links and nodes which describe nothing but meta-data i.e. no contents; an extension towards general HTML contents can be easily imagined, it has however not been thoroughly discussed yet.

In summary, there have been considerable achievements in the attempt to provide design support for hypermedia; the most promising ones are based on type concepts of what we call hyperwebs in this paper. Most such developments relate to hypermedia systems *other* than WWW. Even worse, the considerable achievements made are contrasted by a rather moderate state of commercially available authoring tools for WWW (with a minor exception being the "site management" support given by systems like NetObject Fusion™. The most general and most adequate representation of hyperwebs has been found to be a "graph" of nodes and links.

THE Guts APPROACH

The Guts approach (generic unified typing system) described here leverages off multi-year research at the hypermedia group of the first author. It is based on two principle approaches:

- The PreScript approach to hypermedia typing concepts mentioned in the section above was adopted to the WWW context. The resulting system, called *WebStyle*, is implemented in Java and features full HTML conformance.

- In addition, individual concrete *ontologies* are introduced based on WebStyle. They cater for different facets both of the hyperweb development lifecycle and of alternative application domains, approaches, and components.

Using learning systems as an example, the lifecycle may be supported, e.g., via ontologies for instructional analysis, for instructional design, for domain analysis, and so on. Different alternative approaches may be offered, for instance, ontologies that express rather traditional instructional concepts and rather advanced ones. The key to understanding Guts is its way of representing knowledge. In the teaching context, *knowledge* means *content* of courses together with information *about* the entities involved in the teaching-learning situation, for example content, courses and learners. The latter kind of knowledge is called *meta-information*.

Principal mechanisms for knowledge representation and inference as used in GUTS were thoroughly studied the fields of *semantic networks* and *graph grammars* (see for example [Sowa91] and [Rozen97], respectively). As advocated earlier, the basic underlying data structure is the *Graph*. Our extended notion of a Graphs - called *WebStyles* - comprise a grammar for expressing *static* (syntactic, structural) and *dynamic* (semantic, navigational) aspects.

WebStyles

WebStyles are based on previous work about "generic and dynamic aspects of hypermedia" [Richartz96]. They consist of three parts: generic nets, procedures, and rules.

Generic Nets: Generic nets are the core of hyperweb typing in that they describe the essential construction rules for all hyperweb instances that adhere to a considered type. A unique characteristic wrt. the state of the art is the fact that generic nets can themselves be considered hyperwebs. This means that, in order to cope with the generic net aspect of WebStyles, users do not have to learn entirely different paradigms such as PetriNets or algebra. ON the other hand, generic nets exploit some of the functionality found in graph grammars (cf. [Rozen97]) without directly exploiting the burden of their formalisms.

Being hyperwebs, generic nets consist of nodes and links. At the degree of detail discussed in this section, three basic kinds can be distinguished for both nodes and links: mandatory ones, optional ones, and a repetitive kind which will be discussed further below. These kinds are of course orthogonal to the node and link types assigned in the application context. The instantiation of hyperweb types can be considered an evolutionary process which may take the whole lifetime of a hyperweb (cf. "live" documents or hyperwebs representing software, configurations, etc.), Therefore, the type description and the instance "live together", so that instantiated nodes and links can gradually "populate" a generic web. All elements described up to now are depicted in figure 1.

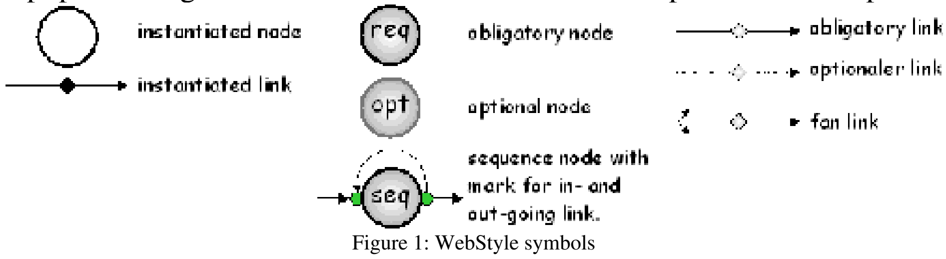
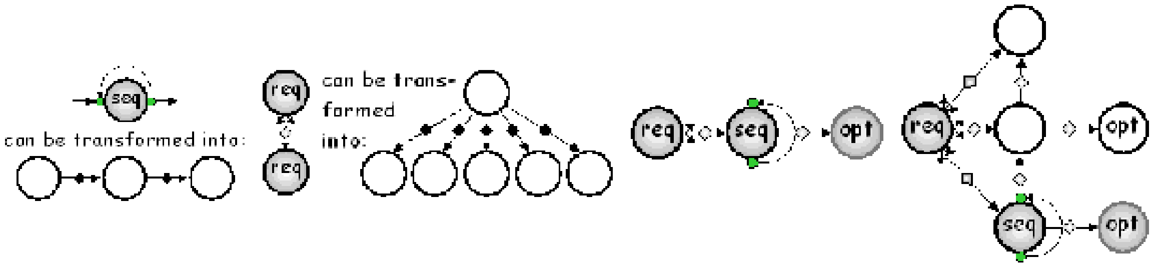


Figure 1: WebStyle symbols

Transformation in generic nets: Apart from the obvious transformations - instantiating mandatory nodes and links, instantiating or erasing optional nodes and links - two interesting transformations remain from above: instantiating "sequence nodes" and "fan links". The following figures help to clarify how nodes and links can be transformed.



a) sample transformation of a sequence node

b) sample transformation of a fan link

c) a simple generic net

d) the generic net from (c) after partial transformation

Figure 2: WebStyle transformations

As figure 2a) indicates, sequence nodes are transformed into "chains" of nodes. Thereby, the link type inbound to the sequence node is inserted between the node instances, and the application-specific type associated with the sequence node is assigned to the node instances themselves.

In figure b) a possible transformation of a fan link is shown. Obviously, fan links expand into "bunches" of links originating from a common source node. The node and link types are assigned as given in the generic net in the obvious way.

Figure c) depicts a section of a generic net consisting of two types of nodes and two types of links. By applying a number of transformation steps, the net shown in d) can be constructed. The example shows that fan links and sequence nodes can be combined in a useful manner: here, the fan link expands into a bunch of links that lead from the initiating node to all the nodes in the chain which results from the sequence node. The optional node outbound to the sequence node gets replicated for all elements of the chain (and has been erased for the topmost node, instantiated for the other two). Note:

Further kinds of nodes/links: Apart from the above-described kinds, alternatives and meta-nodes are supported**.** Alternative nodes and alternative links are used by the author of a hyperweb to offer a choice from different possibilities during construction. Meta-nodes are used to build recursive and hierarchical structures; thereby, hyperwebs are represented as a single (meta-) node at the next higher level of abstraction. They can help to model complex sub-nets and can be used, e.g., to build tree-like structures. For more detailed descriptions cf. [Richartz96].

Attributes: In the above example according to figure 2d), smart readers might reckon why nodes outbound to a sequence node are replicated for all nodes in the resulting chain - quite as well, a single copy (of the node marked as "opt" in the above example) might be kept, with the corresponding link outbound from each node in the resulting chain pointing to that single copy; this is in fact an option, controlled by a specific attribute associated with the links present in generic nets. More generally spoken, each WebStyle object has general attributes, like its node/link type, and more specific attributes, like lower bound and upper bound. The bounds for example are used by the transformations and define how many nodes or links can be instantiated.

Procedures and Rules: Besides default procedures (like *isTraversible* which tells if an object may be traversed) user defined procedures and rules may be attacheto nodes and links. These procedures and rules may influence the construction of a net even more (e.g. by constraining it) and may influence the navigation in such a net, too.

Ontologies

Knowledge representation involves *classifying* the ?things? to be represented, e.g. «Mars» is a «planet», «next» is an «order relation», «is a» is a «genus-species relation». Ideally the *classes* (concepts, types, the terms inside french quotes «?») are explicitly written down and put in relation with each other. This is called a theory, conceptualization, or, as is fashionable, an *ontology*. (Ontology as a part of philosophy is the study of being, or, the basic categories of existence. With the indefinite article, the term "an ontology" is often used as a synonym for a taxonomy that classifies the categories or concept types in a knowledge base [Sowa91], p. 3)

There are ontologies for ?everything?. For instance, in instructional design, if one wants to use Gagné?s events of instruction [GBW92], one could define an ontology containing «gain attention», «indicate goal», «recall prior knowledge», «present material», «provide learning guidance», etc. Or, to be able to talk in terms of Reigeluth?s elaboration theory [Reigel87], one needs «fact», «concept», «principle», and «procedure».

In these examples we did not consider any relations and formalization of semantics. If one tries to work out these aspects, it soon will be evident that something crucial is missing: *How* could such an ontology be defined? In which language? Our approach here is to define a kind of "bootstrapping" ontology which built in. Guts? *representation ontology* is rich enough to capture the computational content of new, user defined ontologies. It comprises

- the objects «object», «theory», «abstraction», «type», and «rule» and
- the relations («relation», «genus-species», «instance», «composition», «equivalence», «order», «derivation», «functional», and «context».

Together with this representation ontology (i.e. the above-mentioned "basic, built-in" ontology that is used to define other ontologies), WebStyles can be considered as a specialized representation language (cf. KIF [GF92] with its so-called "frame-ontology"; note, however, that Sowa mentions that "the structure of a knowledge representation language depends critically on its ultimate goal" ([Sowa91] p. 157), and since WebStyles and KIF differ in purpose, their flavor, appearance and computational properties are different - although WebStyles could be easily mapped to KIF and back).

A sample re-use case with Guts

To take an arbitrary re-use case for Web-based information, we want to imagine the following: A department of philosophy at a university has texts of relevant philosophers on-line. They want to prepare these texts for re-use by students. A common kind of task assignment lets students create their own philosophical texts, incorporating lines of thought and arguments as developed in the above-mentioned texts.

Now, in order to prepare the on-line texts for re-use, they are to be structured based on semantic networks which describes rhetoric spaces. A corresponding hyperweb type is created; it is based on ideas which have been traditionally used to formalize argumentation. The central concepts are "issues, positions, and arguments (IPA)" for the macro-level (cf. [McCall90] for an elaborate version) and the Toulmin argumentation space for the micro level (cf. [Toul58] for the original reference).

The typing for the entire rhetoric space concept can be best described in a modular way, using a top-level hyperweb and several meta-nodes (as mentioned, these are again hyperwebs). One such meta-node, called issue-space, is depicted in the figure below. It covers the subset of a rhetoric space which deals with a single issue and describes the following reasoning (the terms "node" and "node type" are blurred below for the ease of formulation):

- a "problem" or subject to be rhetorically treated is called an "issue" in IPA. The hyperweb (meta-node) depicted shows how such as issue is to be coped with in the rhetoric space
- for a given issue, several positions may be taken (positions can, e.g., be regarded as potential solutions to a problem)
- in a rhetoric discourse, arguments are collected which support or object to a position. A given argument may support several positions, it may also support one position and, at the same time, object to another one, etc.
- as to the WebStyle, the above is described in a hyperweb where a fanlink goes from a (single, mandatory) "issue" node to one or more position nodes. From each position node, a fanlink indicates that one or more "argument" nodes may be reachable (each one either supporting the position or objecting to it).
- Each argument must be backed with a "datum" i.e. a fact or an axiomatic, undebatable statement. From there, a chain of logical conclusions (links of kind "so") leads from argument to argument (in the sense of a logical deduction) until the argument is reached which finally objects to or supports the position.
- as to the WebStyle, the above is expressed as a "datum" node leading to a sequence node "argument" which ends at the "argument" node which in turn responds directly to a position.
- being a meta-node, "issue-space" is embedded into the next higher hierarchy level; this embedding is based on the fact that issues may depend on one another and that certain positions may suggest the treatment of a separate issue; accordingly, there are "depend" links leading to issues outside the meta-node and a "suggests" links leading from a position to issues outside. In the meta-node, only the ports for these links are defined and depicted, their use "outside" must be defined on the next level.

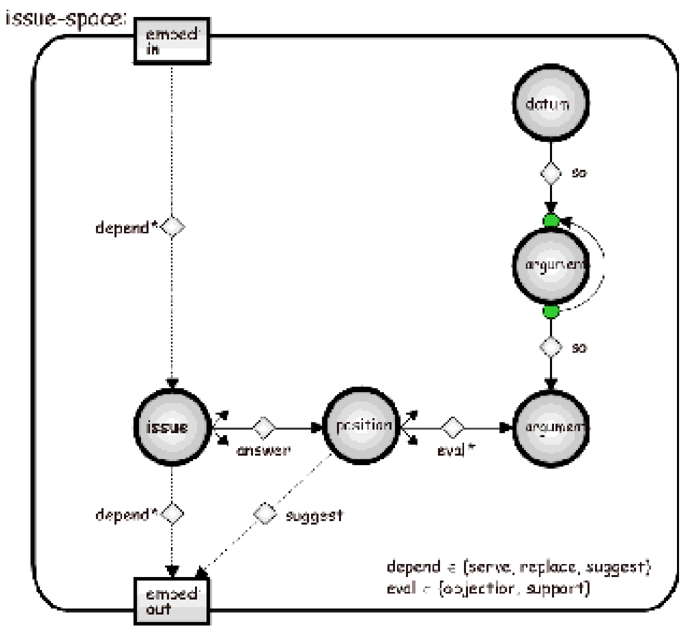


Figure 3: WebStyle example "issue-space": generic net for a meta-node

Note that in the above example, we have only considered the "generic net" part of WebStyles for the sake of brevity, neither procedures/rules nor the GUTS ontology/typing aspects. The latter have already come into play since the functionality of WebStyle generic nets is defined using the "bootstrap" typing mentioned above.

In addition, there are aspects of the generic net which we do not want to offer a graphical representation for; the reason is that we want to keep the graphical representation simple, covering the important aspects; additional aspects shall be expressed literally. To take an example, for the last item listed above one might want to make sure that a position may only suggest a "new" i.e. different issue, not the one to which it is an answer. The corresponding type rule reads as follows:

```
// i1:Issue --[a:Answer]--> p:Position i2:Issue i1<>i2
// ----------------------------------------------------------------
// p --[s:Suggest]--> i2
```

The above is to be interpreted as "if there is an answer link a between issue i1 and position p *and* if i2 is a *different* issue, then a ?suggest? link s may be created between p and i2. Note that this constraint may be expressed on the higher level of abstract by "forbidding" that an outgoing ?suggest? link of an issue-space may be looped back to its incoming port (in this case, the constraint can in fact be expressed graphically), but the hyperweb type author might as well want to keep this constraint together with the issue-space hyperweb by using the above-mentioned type rule.

Instantiation: reworking the on-line documents now means to create a hyperweb instance which conforms to the type described above; whenever a node (of type issue, argument, position, datum) is created, it has to be mapped onto a portion of the existing HTML document. This iterative process of link-creation, node-creation and content-mapping is driven by the generic web which remains attached to the instance. For each (node/link) creation step, the user has to refer to a "current" node. The core of the system, the so-called "chain-algorithm", then checks which expansions are valid for the current node in the current context. It is this chain-algorithm which enforces conformance to the WebStyle (generic net) and to the semantics of generic nets (i.e., of optional and mandatory nodes and links, sequence links, etc.)

Comparison: Even with the above small example, the reader should have been able to get an idea about relations and constraints that can not or hardly be expressed with other concepts. Meta data embedded in HTML can not even express concepts like fan links; approaches based on triples as mentioned earler can neither express sequence nodes at all nor the effect in the above example that only the last "argument" node in a chain may be an "answer" to a position, etc.

Benefit: Following the example from above, the following benefits become evident. If an on-line philosophical text is re-worked using a WebStyle as was partly described above, several different advantages can be exploited. For one, students can easily create rhetoric spaces and resulting philosophical texts re-using the uniquely represented documents and issue-spaces (which might have been authored in different centuries!). Using an elaborate generic net with additional type rules, the re-working itself may be tightly "controlled" to conform to the WebStyle author?s intends. And with rules and procedures added properly, sophisticated navigation and presentation support for the final reader may be prescribed (in a re-usable form) in the WebStyle; to cite just two examples:

- imagine a unique way of laying out issues, corresponding positions and (objecting and supporting) arguments on the screen,
- imagine further a set of rules which describe an "interactive learning mode"; there, some elements in every argument chain (i.e. chain leading from a "datum" to the final argument which supports or objects to a position) might be explicitly hidden until the student has

written down a suggestion about how s/he would fill in these"holes" with arguments; thereafter, the corresponding links (and thus, argument nodes) become accessible so that the student can verify his or her proposal

Note that the detailed meaning and use of rules and procedures is left out here for the sake of brevity. Please note, too, that the use of ontologies has only been treated superficially for the same reason; for this part, the reader may refer to [HTM98].

WebStyles Implementation and Status

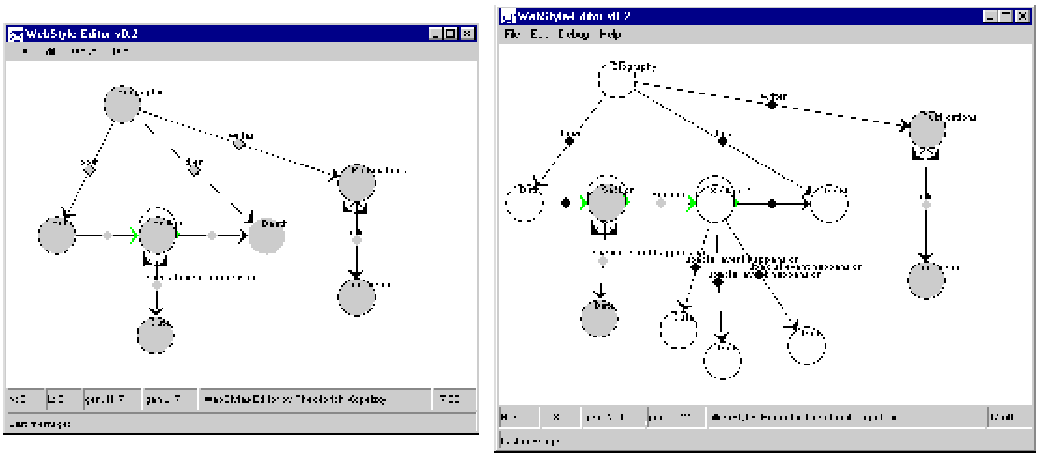
Throughout the past year, two different prototypes have been developed. The first prototype was implemented using JavaScript 1.1. JavaScript was chosen because of the following reasons: a) it integrates very well with HTML-pages and b) it enables users to build dynamic HTML on the client side. The JavaScript prototype was capable of dealing with nodes and links, and even implemented the above-mentioned chain-algorithm on a very basic level. The most serious drawback of the prototype was the lack of a proper user interface, so a lot of activities had to be done by hand.

Meanwhile Java was chosen as language for the next prototype. The Java prototype features graphical editing of WebStyle nets (this includes manipulation of the graph structure and the objects) and implements the complete chain-algorithm.

The prototype is conformant to Dexter hypermedia reference model [HS_94] as far as WWW-compatibility allows, and is divided into two main parts: the user interface, basically represented via the Java class WebStyleEditor, and the WebStyle engine. The engine manages the hypermedia objects and provides a well-defined interface. A Prolog interpreter used to evaluate the rules associated with generic nets.

In order to demonstrate the prototype, a small part of a learning-related domain is modeled with the editor. Figure 5a shows a generic net which models the main characteristics of a biography, consisting of an overview node *Biography*, a node for the birth and an optional node for the death of a person. In addition, a biography contains some major sections (modeled as a sequence node) which can be mapped to major periods in one's life, for example. Each section may have *n* dates. Furthermore a person may write publications which have *n* dates of their own.

In figure 5b the net is shown after some transformations: some nodes and one fan link have been instantiated.



a) the starting net b) the net after some transformations

Figure 5: Screenshots of the prototype

Currently work on the prototype is going on. It will be possible in the near future to export HTML pages linked according to the WebStyles. As soon as that is possible, a more substantial evaluation of the system is planned.

References

[FS89] R. Furuta, P. D. Stotts. Programmable browsing semantics in Trellis. Proc. ACM Hypertext '89, pp. 27-42

[GBW92] R.M. Gagne, L.J. Briggs, W.W. Wager. Principles of Instructional Design. 4th Edition , Hbj College & School Div, 1992

[GF92] M. R. Genesreth, R. E. Fikes et al. Knowledge Interchange Format, Version 3.0 Reference Manual, Computer Science Department, Stanford University, Technical Report, 1992

[GHM_94] K. Grønbæk, J.A. Hem, O.L. Madsen, L. Sloth: Cooperative Hypermedia Systems: A Dexter-based architecture. CACM 37, 2 (Feb. 1994), pp. 64-75. cf. http://www.daimi.aau.dk/~kgronbak/DEVISE/index.html

[HS_94] F. Halasz, M. Schwartz, The Dexter hypertext reference model. CACM 37, 2 (Feb. 1994), pp. 30-39.

[HTM98] R. Hauber, T. Kopetzky, M. Mühlhäuser. Lifecycle Support for Hypermedia Based Learning. Proc. Ed-Media 98, AACE Conference on Educational Hypermedia and Multimedia, Freiburg, Germany, June 1998 (to be presented).

[LaS98] Lassila, O., Swick, R. Resource Description Framework Model and Syntax. http://www.w3.org/TR/WD-rdf-syntax

[Lay98] Layman, A. et al. XML-Data. http://www.w3.org/TR/1998/NOTE-XML-data-0105

[McCall90] R. McCall et al. Phidias: A PHI-based Design Environment Ingerating CAD Graphics into dynamic Hypertext. Proc. ECHT '90, INRIA France, Nov. 90, Cambridge University Press, pp. 152-165.

[MM97] A. Mendelzon, T. Milo. Formal Models of the Web, Prof. ACM Database Systems, Tucson, Arizona, June 1997.

[Reigel87] C.M. Reigeluth (Ed.). Instructional Theories in Action. Lawrence Erlbaum Assoc, September 1987

[Richartz96] Martin Richartz. Generik und Dynamik in Hypertexten. Shaker Verlag, Aachen 1996 (in german)

[Rozen97] G. Rozenberg. Handbook of Graph Grammars and Computing by Graph Transformation: Foundations. World Scientific, 1997

[STH97] M. Salampasis, J. Tait, C. Hardy. HyperTree: A Structural Approach to Web Authoring. Software - Practice and Experience, Vol. 27(12), 1411-1426, December 1997.

[SF89] P. Stotts, R. Furuta. Petri-net-based hypertext: document structure with browsing semantics. ACM ToIS 7(1), pp. 3-29

[Sowa91] J.F. Sowa. Principles of Semantic Networks. San Mateo, 1991

[SRB96] D. Schwabe, G. Rossi, S. D. J. Barbosa. Systematic hypermedia application design with OOHDM. Proc. 7th ACM Hypertext '96, pp. 116-128

[Toul58] Toulmin, S. The Uses of Argument. Cambridge University Press, 1958

[WebSQL] University of Toronto, http://www.cs.toronto.edu/~websql/

The Role of Reactive Typography in the Design of Flexible Hypertext Documents

Rameshsharma Ramloll

Collaborative Systems Engineering Group

Computing Department

Lancaster University

Email: ramloll@comp.lancs.ac.uk

ABSTRACT

We introduce the concept of reactive typography with the aim of exploiting the nature of electronic text in the design of interfaces to a shared information space. The usefulness of reactive typography is gauged through a few simple experiments. We describe how reactive typography can be used as a medium for in-context data visualisation and argue how this investigation is related to the presentation issues of the product of mechanisms for dynamic document construction and re-use. This relationship is more prominent when considering web agent systems for data mining or relevant information extraction. The latter often provide flexibility at the expense of the user's sense of interaction control because their inference engines often fail to capture accurately the highly dynamic and context specific nature of reader needs. This leads to ?soft? issues such as how should agents be designed in order to inspire trust of usage. A step in that direction is to allow the said web agents to reflect their internal state of affairs at the level of information presentation. Reactive typography may have some contribution to make in that respect.

Keywords

Web agents, typography, awareness, data visualisation, and interaction control

INTRODUCTION

ArtLex [Delahunt 95], a lexicon of visual art terminology, defines typography as follows:

The design, arrangement, style, and appearance of type matter constitute typography. Among other things, students of typography learn about the uses of various type fonts, including serif and sans serif, capitals and lowercase letters, boldface, italic, and condensed type, letterspacing (kerning), point sizes, and the various factors affecting readability.

We define reactive typography as a particular kind of typography, which is sensitive to some parameters, selected by a designer. The computer display provides the necessary medium for reactive typography as anything that is displayed on a computer screen is refreshed at a high frequency and as such, is thus not static and liable to be modified meaningfully in real time. A long tradition of static ink on paper has influenced the way text is presented on computer screens. There is a need to exploit the inherent properties of this new medium by identifying new ways of presenting textual information.

Experimenting with reactive typography

The experiments involve the study of simple strategies aimed at enabling the public real time awareness of peer reader views about an online document. This mechanism occurs in arguably two distinct steps. The first is the gathering of the reader?s response and the second is the in-context annotation of the text with information representing the response. Previous researchers [Röscheisen et al 95] have already investigated group or shared annotation techniques and this work does not bring any new contribution in that respect. What is of interest to us here, from a communication designer's point of view, is how to least obtrusively tap data from users, and how to best present it without disrupting the readability of the document or imposing heavy task loads on those who wish to access to this extra piece of information.

Experimental set-up

The chosen online documents are about some controversial topic in order to solicit as many responses as possible from its readers.

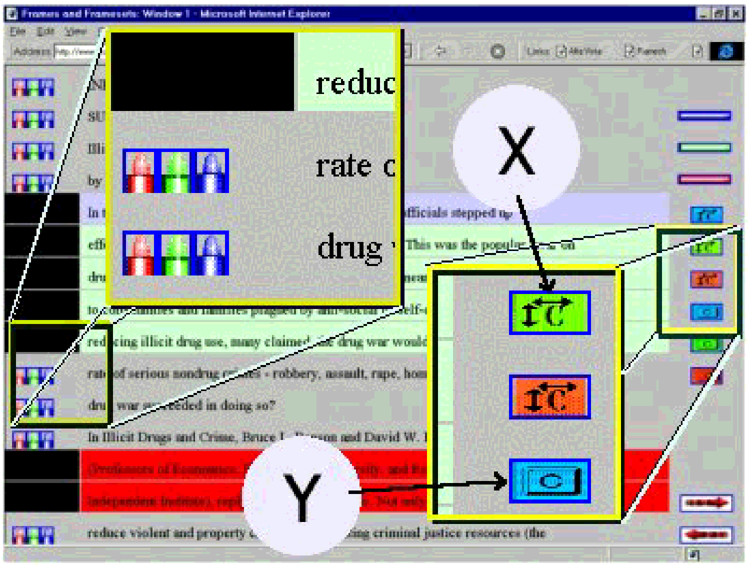


Figure 1: The icons on the left margin are clicked to input user response while those on the right allows access to different visualisations.

In experiment A, ?marker? icons to metaphorically represent ?highlighters? are provided on the left margin of the document (Figure1). Each marker represents a particular response of the reader about the point of view expressed in the sentence or part of the sentence as the case may be. The red marker is to be clicked if the reader disagrees with the view expressed, the green marker, if the reader agrees with it and finally, the blue one, if the reader is uncertain or entertains a 'could go either way' opinion about it. The readers are of course free to decide when they want to express their opinions, very much in the very same way as highlighters are used with ink on paper. On the right margin, readers are provided with visualisation buttons which when clicked would present them with the same text but this time augmented with a different typographical layout that represents certain specific parameters about the community of readers. For example clicking on the green ?c? icon (labelled X) produces a document with the size of the characters representing the degree of consent of the online reader community (Figure 2). Clicking on the blue ?c? button (labelled Y), on the right margin will display the background of each sentence with a blue intensity representing the degree of uncertainty of the community about the views expressed in the document (Figure3). In so doing, the reader is able to gauge the opinion of the community of readers at a glance and to quickly identify the areas of consent, dissent or uncertainty.

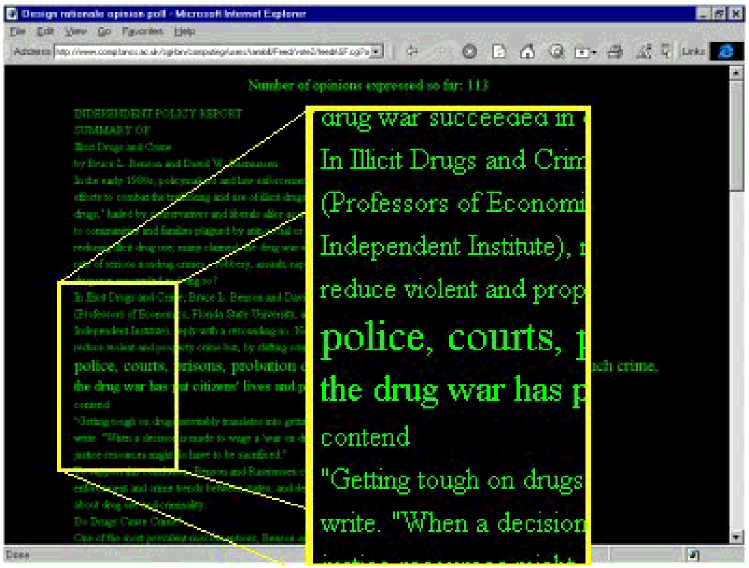


Figure 2: The size of the fonts represents the degree of consent of the online community of readers (e.g. the greater the size, the higher the degree of consent).

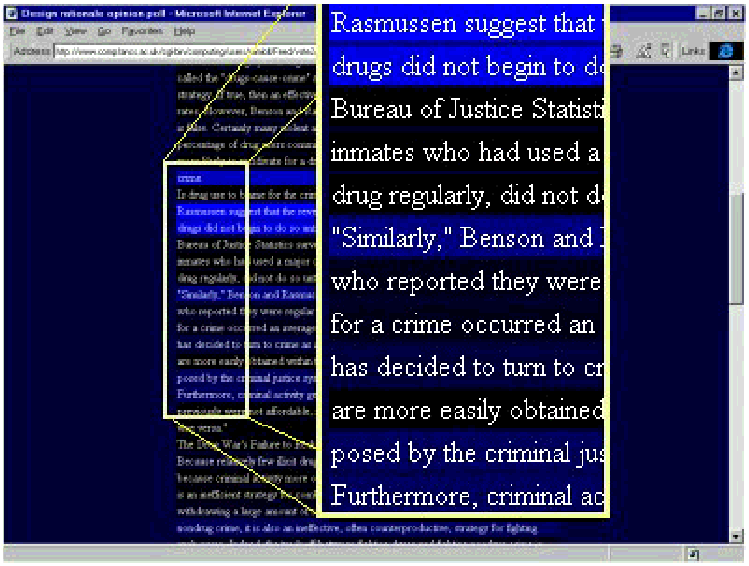


Figure 3 : The blue background colour represents the degree of ?uncertainty? about the views expressed.

In a different experiment B, readers were able to express their views about a given paragraph by clicking on ?thumbs up?, ?thumbs down? or ?no idea? icons. They were also able to visualise the data either by annotating the typographical layout directly or through graphs. For example, in this experiment the background colour expressed the most popular view about the corresponding paragraph.

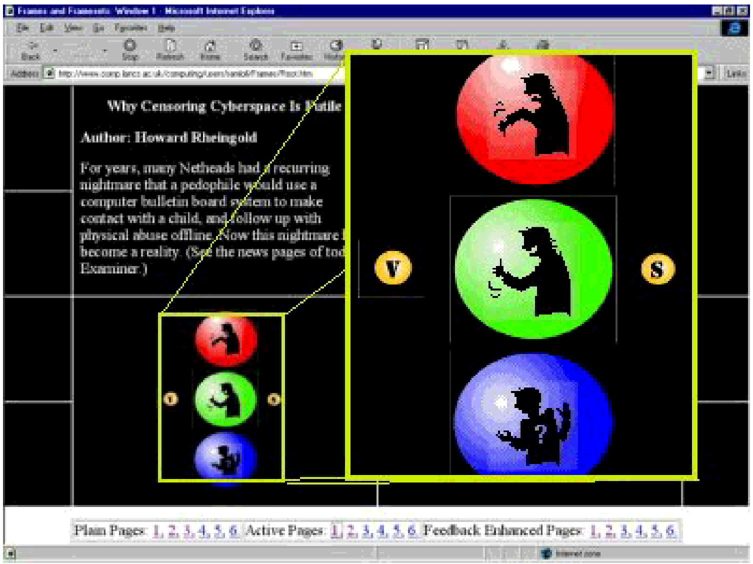


Figure 4: In the Active Pages mode, readers are able to register their views for each paragraph by clicking on the three ?thumbs down?, ?thumbs up? and ?don?t know? icons. Clicking on the ?v? and ?s? button allows access to feed back visualisations.

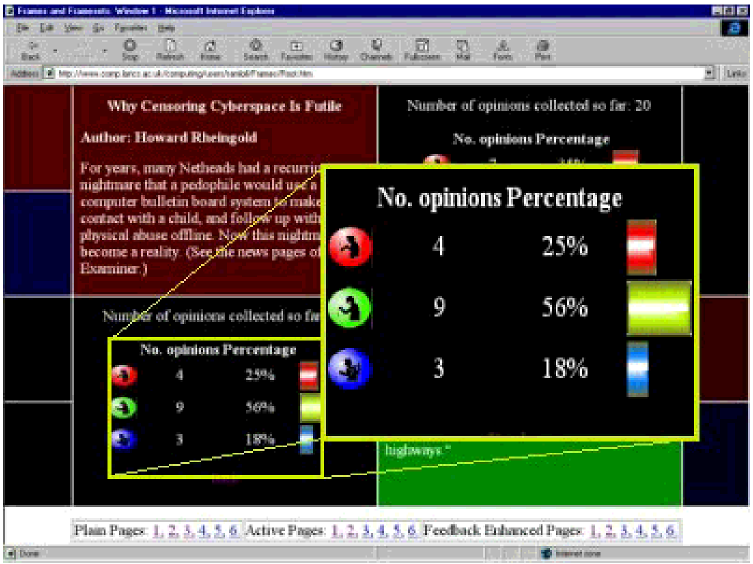


Figure 5: The reader is also provided with the option of visualising graphically the response of the community of readers.

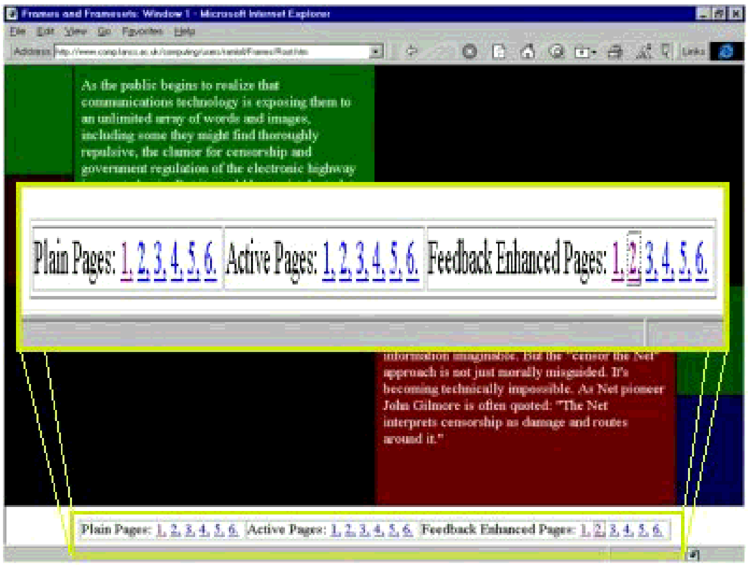


Figure 6: Readers who are not interested in voicing their opinions or who just wish to gauge the response of the online reader community can simply access the feedback enhanced pages.

We realise that there is definitely a lot of scope for improving the granularity of response capture and the visualisations offered but this is out of the scope of this paper as our concern is simply to get some insights into the implications of reactive typography on interface design.

Observations and issues raised

Most of the readers of the online documents were curious to find out what their peer readers thought about its contents. The ubiquitous availability of this information sometimes encouraged readers to go back to the original online document and to voice their opinions, which they haven't before. Most readers showed a preference for the in-context augmentation of text with the extra information rather than displaying it as graphs as is the case in one visualisation option in the second experiment.

Some concerns were raised about not allowing a person to give his her feedback more than once. However, checking if someone has already given his or her opinion already should be done unobtrusively. Any attempt to force readers to register explicitly before expression opinions may in some circumstances dampen any zeal for self-expression. Also opinions do change and ideally, there should be a mechanism that will update the feedback data of a given reader so that at any time it is only the latest feedback that is stored unless there is some special reason that makes the study of opinion shifts useful.

How do these observations inform the design of the presentation component of information filtration agent systems?

The above experiments show that a few simple design strategies can increase the sense of copresence, albeit in an abstract way, among readers of online documents. However, it is obvious that in all the experiments we have been dealing with, simply involves the visualisation of information annotating some text and that the processes that support this visualisation per se do not qualify as agents for the following reason. While they respond in a timely fashion to changes in their environment, are autonomous, and temporally continuous, they fail in one of the most prominent properties of agent systems, namely their lack of a goal oriented or pro-active character [Franklin et al 96]. After putting this possible source of confusion out of the way, we proceed to describe how reactive typography may be useful in the design of the information presentation of web agents for data mining or relevant information extraction. To begin, we revisit an argument usually levelled against such agents. It has often been argued that a pro-active agent may potentially rob the user of her sense of control because of the fact that a piece of software that take care of the information filtering process often has an internal logic which is most of the time hidden. In the following, we show how reactive typography may help to tackle this problem. Even though it is difficult for this approach to expose this internal logic just mentioned, it can certainly express the reliability of the output of the agents concerned.

Let us assume the existence of an agent, which is supposed to help a given user to quickly access Midi files, according to his musical taste, from a given Midi file repository. The web agent may be designed in such a way that when the user reaches the repository site, only the files that match his taste are always displayed. While the agent is certainly successful in helping the user attain the 'right' targets, which have been determined, based on data previously gathered from the user, it is arguably not very flexible for the following reasons. There is always a time lag between the time data was collected and the actual time of interaction so that the solution offered by the agent may not always match the current requirements of a given user. Displaying the files according to some criteria based on some previously collected data, such as an interest profile, denies the user of having access to the knowledge of other Midi files which may indeed fit his current taste or mood. An obvious solution to this problem can involve emphasising in some way, the names of those files that might suit the taste of the user while still presenting her with the other options. A wide variety of visualisation techniques such as fish eye views with multiple foci or a range of typographical techniques exists to achieve this effect. Since, the other files which according to the agent would not be of interest to the user, are still visible in some way, even if they are less prominent, the user can still change his mind or satisfy his immediate need to explore other musical styles. It seems that these cosmetic issues at the level of the information presentation have a non-negligible effect on the sense of control that users experience when being ?assisted? by a web agent. Thus as far as possible, web agents should act as prompters or advisers for interaction rather than wholly acting on behalf of the user. Reactive typography enables a ?soft? perusal filtering of the information presentation which fuzzifies the output of the agents concerned perhaps as a means to reflect their possible internal accuracies.

Similarly, agents responsible for the reader customisation of online newspapers can exploit reactive typography by identifying appropriate typographical parameters that will allow the automatic spatial layout of articles according to reader interests. Thus, one would expect such a newspaper to have prominent spacious headline articles matching what would most interest the reader and at the other extreme, little snippets in small fonts about events of a lesser degree of interest. Another application scenario that illustrates the importance of reactive typography as a useful ?soft? information-filtering stratum is as follows. Often, systems such as ComMentor are designed with the aim of contributing to a culture of widely commenting and reviewing [Röscheisen et al 95]. If this goal is indeed achieved, shared structured in-place annotations [Röscheisen et al 95] are bound to populate on line shared documents. Proper design choices have to be made so that numerous structured in place annotations do not interfere with readability. Reactive typography may provide a possible solution by having for example, annotated text augmented in real time according to the number or content of annotations so that just by glancing at the augmented text, the reader is able to guess the nature of the underlying annotations.

CONCLUSION

The suitability of reactive typography as a medium for ubiquitous user related data visualisation and its relevance to the presentation of the output form a class of agents is explored. We explain how augmenting automatically an existing document with relevant information can achieve multiple browsing contexts. It also seems that embedding information directly in the typographical layout can act as an unobtrusive and useful ?soft? information-filtering step in the process of advising rather than directing a user to a desired information source. One advantage of this technique is that it helps to maintain the user's sense of control about information access and interactions. This technique may thus provide a progressive step towards a possible solution to the common lack of user control criticisms usually levelled against web agents. Reactive typography is an area where communication design professionals, typographers, visualisation researchers and web agent designers can collaborate in order to design useful and ?live? interfaces to the World Wide Web.

The web interfaces in this paper were built using HTML, javascript and perl and the visualisations were largely constrained by the narrow range of typography parametrisations possible in the hyper text mark-up language used. There is thus a pressing need for specialised services for the realisation of the kind of typographical behaviour mentioned in the paper. We hope that research efforts aimed at providing web designers with the necessary tools to manage the typographical layout at the level complexity that we suggested, will gain more prominence in the near future.

ACKNOWLEDGEMENTS

I wish to thank the referees for their numerous comments to improve the focus and readability of this paper.

REFERENCE

M. Delahunt (1995). ArtLex, a lexicon of visual art terminology , http://www.aristotle.com/sskystorage/Art/ArtLex.html

Stan Franklin and Art Graesser (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.

Martin Röscheisen, Christian Mogensen, and Terry Winograd (1995). Beyond Browsing: Shared Comments, SOAPs, Trails, and On-line Communities. Paper presented at the Third International World-Wide Web Conference in Darmstadt, Germany, April 1995.

Using Constraints for Flexible Document Layout

Alan Borning
University of Washington
http://www.cs.washington.edu/homes/borning

Richard Lin
Monash University
http://www.cs.monash.edu.au/~rlin

Kim Marriott
Monash University
http://www.cs.monash.edu.au/~marriott

Peter Stuckey
University of Melbourne
http://www.cs.mu.oz.au/~pjs

Introduction

A major limitation of current internet publishing technology is insufficient layout control. We want flexible and powerful ways to specify layout, but these layout specifications must accommodate a wide variety of media and viewers. A rigid static layout, such as that used in classical publishing, is inadequate in the context of electronic viewing. In contrast to print publishing, the capabilities of the media used for viewing are not known in advance. Different browser types, monitor sizes and ratios, colour or black-and-white monitors render it virtually impossible to design pages that look good on all device types. The situation becomes more complicated when interactive resizing of windows and frames, printing, or communication conditions such as varying bandwidths are taken into consideration. Not only is it necessary to take into account the viewing media capabilities, the viewer's needs and capabilities also need to be considered. For example, font sizes and colours need to be alterable to fit the requirements of sight-impaired viewers. On top of this, the layout of documents that are created dynamically, such as those that display database or search engine query results, can only be predetermined to a small degree. In the ideal case their layout should be generated automatically based on an abstract style specification.

Constraint based-specification provides a general approach to this problem. Constraints have been used for many years in interactive graphical applications for such things as specifying window and page layout, specifying relationships among parts of a drawing, specifying animations, maintaining consistency between application data and a view of that data, maintaining consistency between multiple views, and representing physical laws in simulations. They allow the designer to specify *what* are the desired properties of the system, rather than *how* these properties are to be maintained. Constraints also make it easy to specify *partial information* about an attribute, instead of a specific value, thus making it easier to combine specifications from multiple sources.

It is thus natural to consider constraint-based tools to aid in authoring web documents. We have built a prototype system (described more fully in [Borning 97a]) that allows web authors to employ constraints to specify page layout, including figure placement and column widths and spacing. Some of these constraints will be requirements that must be satisfied, while others may be preferences of different strengths that can be relaxed if need be. In addition, authors can use several *constraint style sheets* to specify alternate sets of constraints to be used under different circumstances, for example, for a one versus a two-column layout. The conditions under which a style sheet is applicable are, of course, also specified as constraints.

Constraints may be imposed by the viewer of the document as well as by the author. These constraints may arise from the capabilities of the system which renders the document or the preferences or constraints of the human viewer. Like those of the author, some of the viewer's constraints can be preferences as well as requirements. The final appearance of the document is thus in effect the result of a *negotiation* between the author and the viewer -- where this negotiation is carried out by solving the set of required and preferential constraints imposed by both parties.

The internet's underlying client/server model implies two possible system architectures. In one model, both the web authoring tool and the web browser can perform runtime constraint solving. The web author uses the solver while constructing and testing the pages and applets, while the viewer uses a different solver on the viewing machine to determine the final page layout by solving the combined constraints from the author and viewer. In this case a compact representation of the constraints, along with the content of the page, additional layout information and applets, is shipped over the network for each page. In addition, the runtime solver must be shipped once and saved on the viewer's machine. In the alternative model, the web author again uses a powerful runtime constraint solver, but a more restricted set of constraints is available to the viewer. Using *projection* [Harvey 97], the authoring tool compiles a Java program representing a plan for satisfying the author's constraints and the predetermined kinds of constraints that the viewer may impose. This program is then shipped to the viewer's machine---a runtime constraint solver is not needed on the client side. A combination of the two approaches is also possible -- and in fact our prototype uses such a combination.

Constraint-Based Page Layout

Our approach is to use constraints to specify the core aspects of the design layout. The constraints capture the ''semantics'' of the design, those aspects that must hold true for the layout to be appealing. In our prototype, the designer can specify placement of the document elements using linear arithmetic equalities and inequalities. Such constraints allow easy specification of table, column, and image placement in a way that scales gracefully.

The following example, taken from [Borning 97a], explains the underlying idea. Consider the page layout shown in Figure 1a. We require that the text is arranged in two columns, that figures A and B are centered in the first and second columns respectively, and that the tops of the two figures are aligned horizontally. These layout constraints are captured by the following equalities and inequalities:

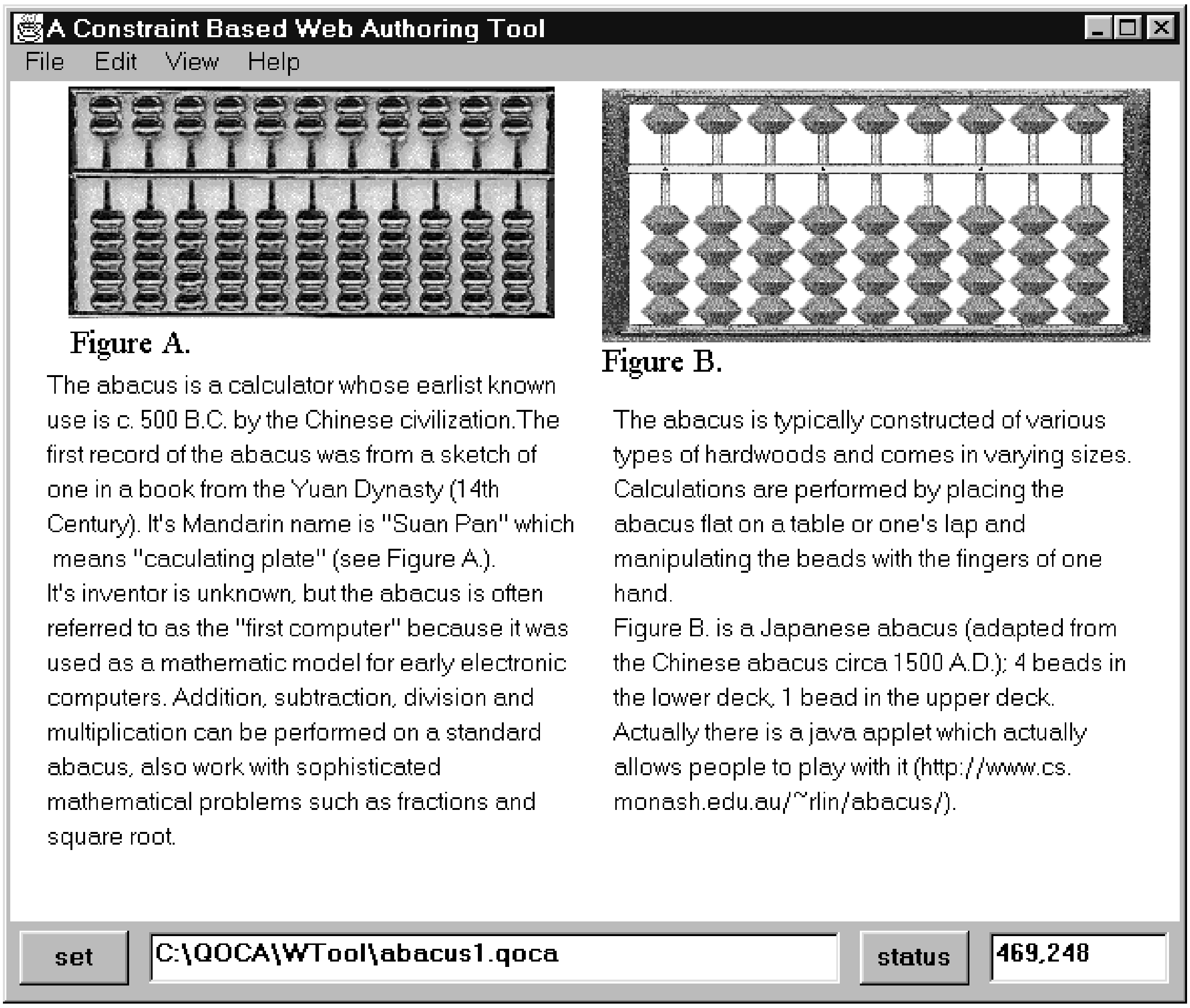


Figure 1a: Two Column output

(1) $PW = LG + MG + RG + CW_1 + CW_2$

(2) $CW_1 = CW_2$

(3) $LG = RG = 0.05 * CW_1$

(4) $MG = 0.7$ cm

(5) $FigA.midx = LG + 0.5 * CW_1$

(6) $FigB.midx = LG + CW_1 + MG + 0.5 * CW_2$

(7) $FigA.top = FigB.top$

(8) $FigA.width <= CW_1$

(9) $FigB.width <= MG + CW_2 + RG$

Constraint (1) states that the page width, PW, is the sum of the widths of the left, middle and right gutters LG, MG, and RG, and the two columns, CW_1 and CW_2 . Constraint (2) states that the two columns have equal width; (3) states that the left and right gutters are equal and are 1/20 of the width of the columns; (4) states that the middle gutter is of fixed size (0.7 cm); (5) states that the x value of the midpoint of Figure A is at the center of the first column; (6) states that Figure B is centered in the second column; while the last equality (7) enforces that the two figures are horizontally aligned. The inequalities (8) and (9) enforce that the columns are wide enough for

Figures A and B.

For a given value of the page width *PW* we can find a solution to the other variables that satisfies these constraints and that gives us a layout. For instance, if PW = 21.7 cm then $LG = RG = MG$ = 0.5 cm and $CW_1 = CW_2$ = 10 cm. Conversely, if PW = 42.7 cm then $LG = RG$ = 1.0 cm, MG = 0.7 cm and $CW_1 = CW_2$ = 20 cm. Note how the left and right margins scale with respect to the page size while the middle gutter has an absolute size.

This model is, however, a little too simple. In particular it does not allow the designer to state preferences for values. We therefore extend our model to allow the user to specify that an inequality or equality is preferred but not required, so that the constraint should be satisfied when possible but does not need to be. Constraint hierarchies [Borning 92] formalize such preferences. A constraint hierarchy consists of collections of constraints each labelled with a strength. There is a distinguished strength label *required*: such constraints must be satisfied. The other strength labels denote preferences. There can be an arbitrary number of such strengths, and constraints with stronger strength labels are satisfied in preference to ones with weaker strength labels. In our example, the equalities and inequalities given earlier are required constraints and we have used *weak* to label non-required constraints. Given a system of constraints and preferred values, the constraint solver must find a solution to the variables that satisfies the required constraints and which is as close as possible to the preferred values.

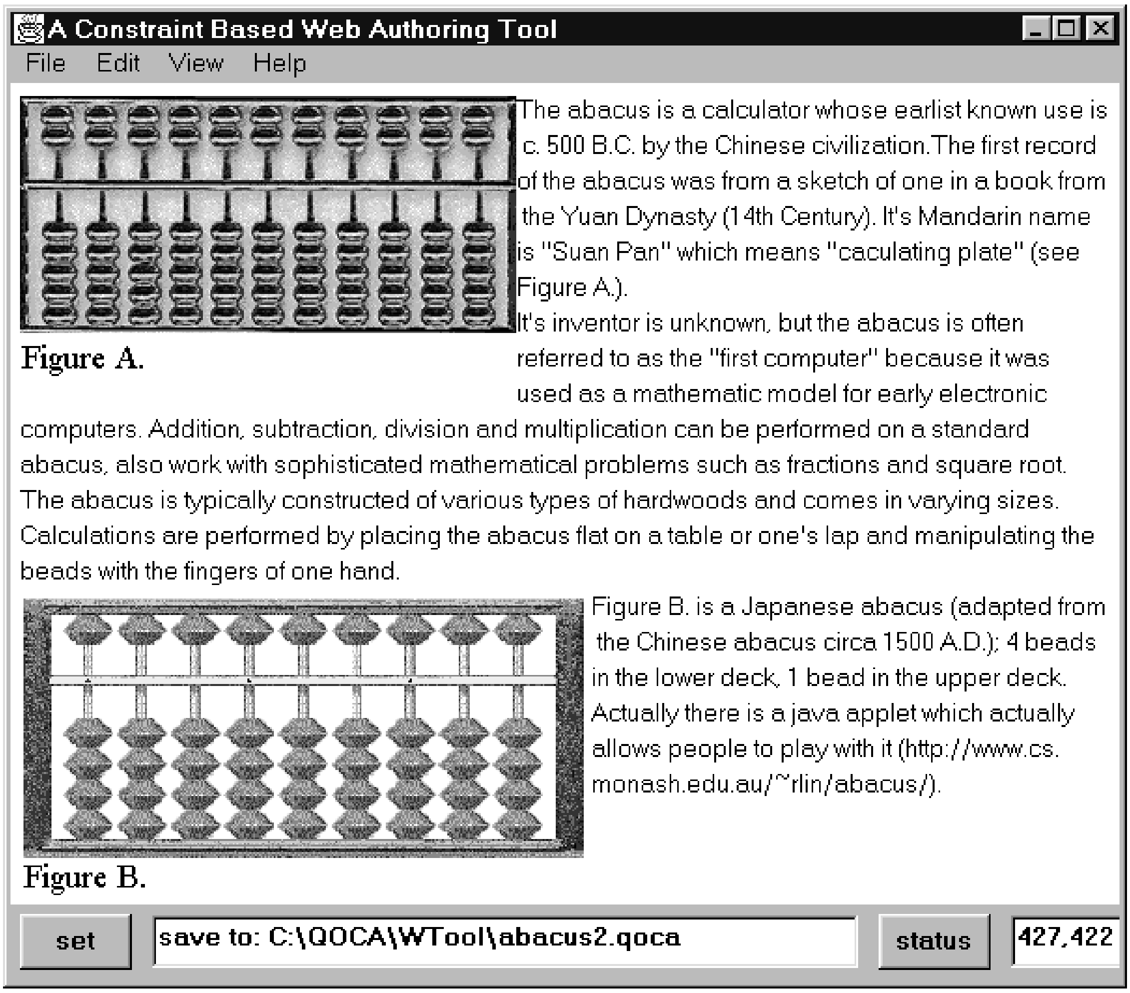


Figure 1b: Single Column output

In the previous example, we have required that the columns be wide enough for Figures A and B. If they are not, then this is not an appropriate layout. For instance, if the width of Figure A and B is 10 cm, then we cannot solve the constraints when the page width is less than 21.3 cm. In this case the designer might wish to use a single column with the example layout is shown in Figure 1b.

To accommodate such situations, our model allows the designer to provide multiple *constraint style sheets*. Each style sheet includes linear arithmetic constraints that define the layout of the design and that dictate when the design is appropriate. During manipulation by the viewer, the viewing tool will choose the appropriate style sheet and lay out the document subject to the constraints in the sheet. As the viewer changes the requirements, the document will be redisplayed using the current style sheet until the constraints are inconsistent with the viewer's desires. In this case the viewing tool will choose another style sheet for the document which is consistent with the viewer's constraints.

For instance, if the viewer of our example document originally displays the document in a window of width 28 cm, then resizes the window to 20 cm, the design will change from two column to one column. If the viewer now resizes it back to 28 cm, the design will change back to two column.

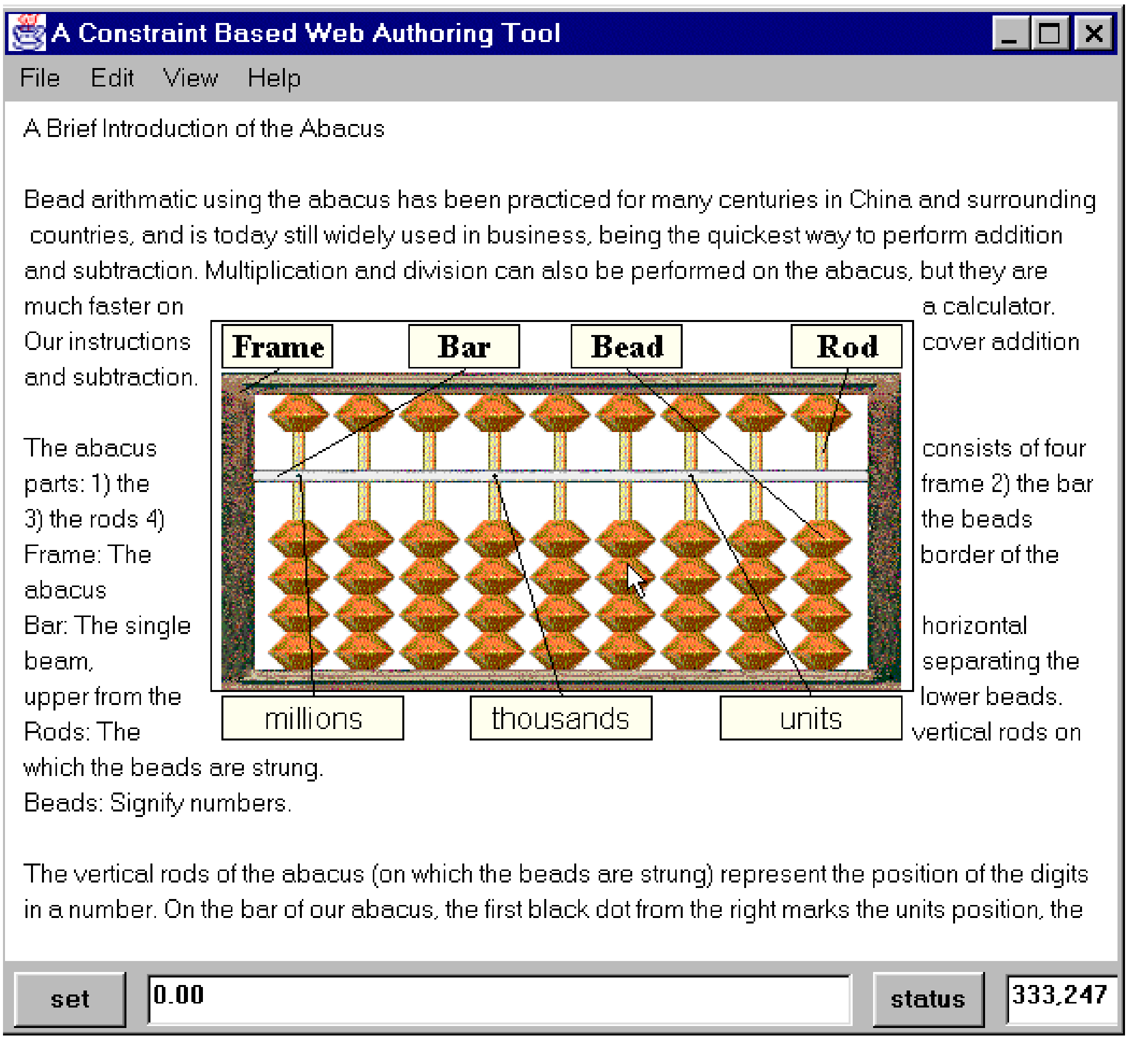


Figure 2a: Narrow page output

As a more complex example of constraint-based page layout, consider the web page shown in

Figure 2a and Figure 2b. These figures show two constraint style sheets, the first for a narrow page with one column layout and the second for a wider page with two column layout. In each, layout constraints ensure that the central abacus figure is centered and that the surrounding labels remain appropriately aligned as the window is resized or other edits performed. Each style sheet contains approximately 110 constraints.

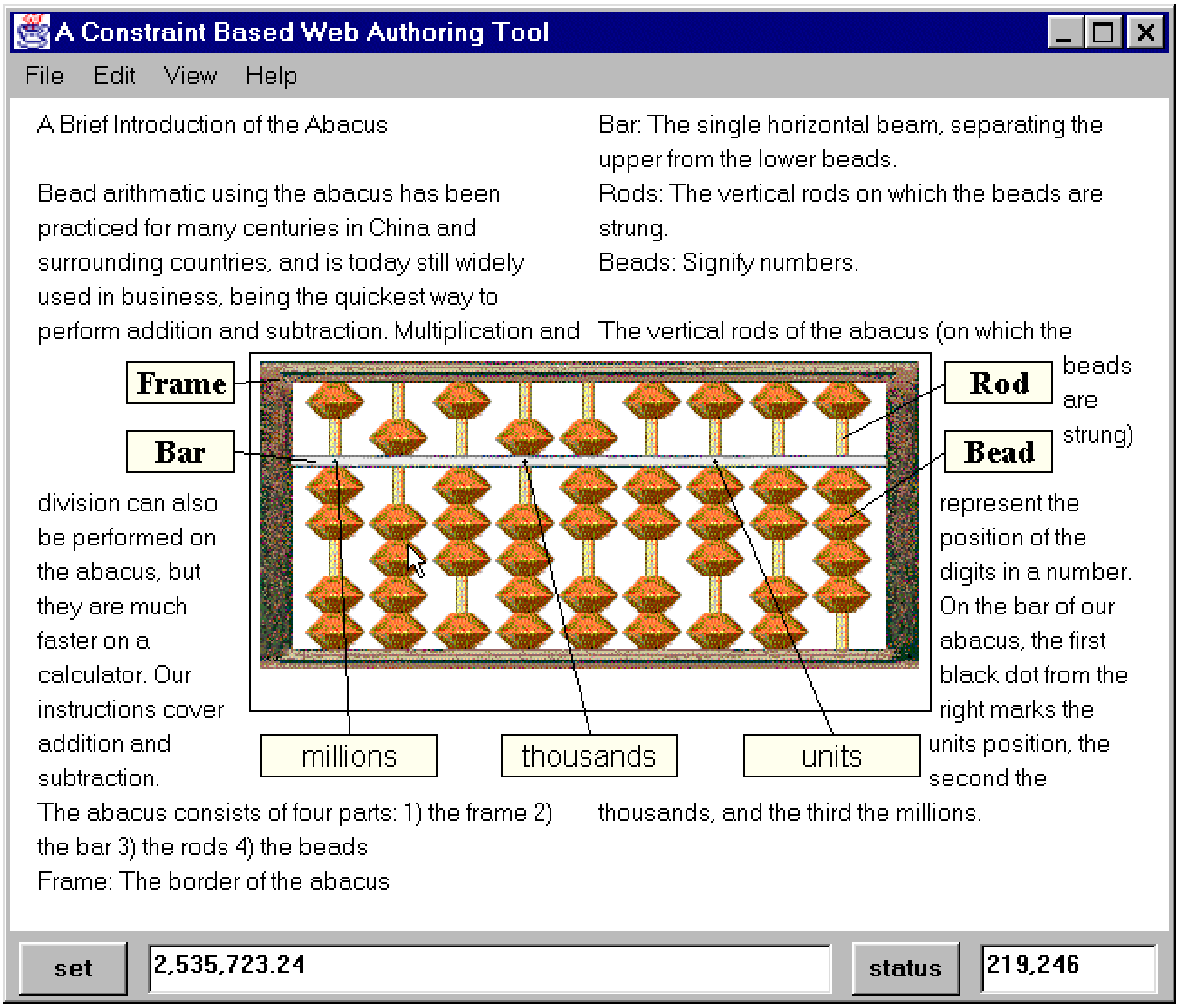


Figure 2b: Wide page output

Prototype System

Our prototype system consists of the document authoring tool, the viewing tool and the constraint solver. The constraint-based authoring tool is used by the designer to construct the constraint style sheets and document contents. Ideally the designer should not need to think in terms of arithmetic constraints or even be aware of the real nature of the constraints. At present pre-defined templates are used for this purpose, but further research is required. The viewing tool integrates constraints from the designer with those of the viewer, check which constraint style sheet is appropriate, resolve the constraints, and then display the document contents using the values from the solution to place elements in the layout.

The constraint solver is a key component of this architecture. It needs to support the following:

1. *Incremental constraint solving*, including incremental addition, deletion, and resolving of constraints for changing user inputs.
2. *Hierarchies of constraints* with required and preferential constraints.
3. *Efficient computation of a solution*, since it should not cause additional delay in retrieving a document. In addition, direct manipulation requires fast incremental computation of new solutions for good interaction.
4. *Geometric* and *non-geometric* constraints. Geometric constraints naturally arise in layout while non-geometric constraints arise when specifying non-geometric attributes of text and graphics such as font sizes, type and colour.

These are rather strong requirements. Indeed probably the main reason why constraints are not more widely used in computer graphics is that current constraint solvers do not come close to meeting these requirements. Fortunately, as part of the prototype implementation we have developed a Java solver that meets the first three requirements; we believe that with recent advances in constraint solving technology it will be possible to meet all of them. It provides fast incremental linear arithmetic constraint solving and solves the constraint hierarchy by translating it into a linear programming problem [Borning 97b].

Related Work

Cascading style sheets (CSS1 and CSS2) [Lie 96, Lie 98] partially address these issues by providing a number of new constructs for defining the layout of a page. For example, with CSS it is possible to specify font sizes, style attributes, colours, etc in a separate style sheet. Authors as well as viewers can define style sheets, and the final appearance of the document is determined by merging these style sheets into a single hierarchical style definition. The principal difference between CSS1 and constraint style sheets is that cascaded style sheets allow one to specify a particular value for a given attribute (or in some cases a percentage), while constraint style sheets allow more general constraints, i.e. partial specifications, to be given for these attributes. For example, a cascaded style sheet can include a rule specifying that the left margin of a layout element is a particular value. On the other hand, a constraint style sheet can include an arbitrary linear constraint on the left margin, which might constrain it to be less than twice some other value. (Constraining it to have a particular value is just a special case of the general constraint mechanism.) CSS2 moves further in the direction of general constraints, and includes such things as requirements that a floating box be placed as far as possible to the left in its enclosing box. Another difference is that we allow the appearance to change interactively as the viewer resizes the page. This means that the server must provide multiple style files, and the viewer must choose the style file that is compatible with the reader's requirements, and change this choice dynamically. In addition, we also support figure layout (although a similar extension to cascaded style sheets is expected [Lie 96] in future versions).

The `<table>` environment in HTML 3.0 can be viewed as providing certain constraints, including preferences as well as requirements, for example, desired cell width expressed either as an absolute quantity (in pixels) or as a percentage of the total table width; again, however, there is no general constraint capability.

Weitzman and Wittenburg [Weitzman 94] have investigated the use of relational grammars for document design. Their work is closely related to ours since in effect they use a grammar which details a class of constraint layout styles. However their interest is in specifying and recognizing layout styles rather than constraint solving. They only consider rather weak constraint solving techniques based on local propagation. Indeed, it seems rather natural to combine their work with ours.

Future Work

Our plans for future work include the following projects.

- We want to support a wider range of constraints for web authors and viewers to use. These will include constraints on non-numeric attributes such as fonts and colours.
- We are investigating extensions to CSS to allow more general kinds of constraints, in collaboration with Håkon Wium Lie and others.
- We plan to design and evaluate better user interfaces for the document authoring tool.
- We plan to investigate the constraint-based diagram layout.
- Another important application of constraints is to specify the behaviour of applets used in web pages. See [Borning 97a] for some examples. We plan to design a constraint-based animation authoring tool, which we hope will allow authors to construct animations more easily than using present techniques.

Bibliography

[Borning 92]
Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223-270, September 1992.

[Borning 97a]
Alan Borning, Richard Lin and Kim Marriott. Constraints for the Web. In *Fifth ACM International Multi-Media Conference*, pp. 173--182. Seattle, November 1997. http://www.acm.org/sigmm/MM97/papers/borning/constraints.html

[Borning 97b]
Alan Borning, Kim Marriott, Peter Stuckey, and Yi Xiao. Solving linear arithmetic constraints for user interface applications. In *Proceedings of the 1997 ACM Symposium on User Interface Software and Technology*, October 1997.

[Harvey 97]
Warwick Harvey, Peter Stuckey, and Alan Borning. Compiling constraint solving using projection. In *Proceedings of the 1997 Conference on Principles and Practice of Constraint Programming (CP97)*, October 1997.

[Lie 96]
Hakon Wium Lie and Bert Bos. Cascading style sheets, level 1. W3C Recommendation, December 1996. http://www.w3.org/pub/TR/REC-CSS1

[Lie 98]
Hakon Wium Lie and Bert Bos. Cascading style sheets, level 2. W3C Working Draft, 28 January 1998. http://www.w3.org/TR/WD-CSS2

[Weitzman 94]
L. Weitzman and K Wittenburg. Automatic generation of multimedia documents using relational grammars. In *Proceedings of 2nd ACM Conference on Multimedia*, 1994.

Table of Content

Themes, Program Committee	1
Program, Participants	3
Report on the Workshp	5

A Jumping Spider: Restructuring the WWW Graph to Index Concepts that Span Pages" *Curtis E. Dyreson*	9
An Architecture for Web Visualisation Systems *Petros A. Demetriades and Alexandra Poulovassilis*	21
Component Advisor: A Tool for Automatically Extracting Electronic Component Data from Web Datasheets, *Malu Castellanos, Qiming Chen, Umesh Dayal, Meichun Hsu, Mike Lemon, Polly Siegel, Jim Stinger*	31
Database Querying on the Worls Wide Web: *UniGuide*-An Object-Relational Search Engine for Australian Universities *Carlos F. Enguix, Joseph G. Davis, and Aditya K. Ghose*	39
The FRAMES Project: Reuse of Video Information using the World Wide Web *C. A. Lindley, B. Simpson-Young, and U. Srinivasan*	51
A personal Evolvable Advisor for WWW Knowledge base Systems M. Montebello, W.A. Gray, S.Hurley	59
Transforming the Internet into a Database *Anand Rajaraman*	71
Typing Concepts for the Web as a Basis for Re-use *Max Mühlhäuser, Ralf Hauber, Theodorich Kopetzky*	79
The Role of Reactive Typography in the Design of Flexible Hypertext Documents *Rameshsharma Ramloll*	91
Using Constraints for Flexible Document Layout *Alan Borning, Richard Lin, Kim Marriott and Peter Stuckey*	99