INformation Systems

Flexible and scalable digital library search

M.A. Windhouwer, A.R. Schmidt, R. van Zwol,
M. Petkovic, H.E. Blok

REPORT INS-R0111 DECEMBER 31, 2001

ISSN 1386-3681

Flexible and Scalable Digital Library Search

Menzo Windhouwer [1]

Albrecht Schmidt [1]

Roelof van Zwol [2]

Milan Petkovic [2]

Henk Ernst Blok [2]

[1] CWI, Amsterdam, The Netherlands
[2] University of Twente, Enschede, The Netherlands

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT
In this report the development of a specialised search engine for a digital library is described. The proposed system architecture consists of three levels: the conceptual, the logical and the physical level. The conceptual level schema enables by its exposure of a domain specific schema semantically rich conceptual search. The logical level provides a description language to achieve a high degree of flexibility for multimedia retrieval. The physical level takes care of scalable and efficient persistent data storage. The role, played by each level, changes during the various stages of a search engine's lifecycle: (1) modeling the index, (2) populating and maintaining the index and (3) querying the index. The integration of all this functionality allows the combination of both conceptual and content-based querying in the query stage. A search engine for the Australian Open tennis tournament website is used as a running example, which shows the power of the complete architecture and its various components.

1998 ACM Computing Classification System: [H.2.4, H.3.1, H.3.7] Multimedia indexing
Keywords and Phrases: digital libraries, search engines, multimedia indexing, multimedia retrieval, information retrieval, conceptual modelling, feature grammars, feature detection, content-based video retrieval
Note: The project "Digital Media Warehouses" (DMW) is funded by the Telematics Institute.

INTRODUCTION

Unfortunately, the Internet is still, in the eyes of many of its users, what appears to be a chaotic organization of information sources. The state-of-the-art means for finding information are full text-based search engines, *e.g.* AltaVista and Google, and hierarchical indexes or directories, *e.g.* Yahoo. They constitute entry points based solely on the textual content of web pages. Still much research is needed to extend the capabilites of search engines to other media such as audio, image, or video. The first steps have been made, but the area is still very much in its infancy.

The challenge lies in the presentation orientation of the Internet: site designers usually have more semantic information than is visible on the Internet. For example: the designer of a tennis site knows that "Monica Seles" is the name of a female tennis player. Making this extra domain knowledge explicit would allow a search engine to restrict the range of possible semantic domains and to use more specific multimedia retrieval techniques. This knowledge could thus be used to semantically enrich both indexes and queries. The larger scale and unmoderate nature of the Internet as a whole impedes this approach. However, for intranets this approach still incurs high additional costs over deploying a 'standard' search engine restricted to a certain IP-domain.

The integrated database approach, described in this chapter, lowers the costs of developing specialised search engines. For the unlimited, Internet scale, domain it still uses well known textual retrieval techniques, but, instead of using a special purpose index structure, now transparently integrated into a DBMS. The proposed architecture consists of three levels: the conceptual, the logical and the physical level. The conceptual level schema enables by its exposure of a domain specific schema semantically rich conceptual search. The logical level provides a description language to achieve a high degree of flexibility for multimedia retrieval. The physical level takes care of scalable and efficient persistent data storage.

The role, played by each level, changes during the various stages of a search engine's lifecycle. This lifecycle consists of the following stages:

1. The initial phase is the (re)creation of the schema which underlies the search engine: describing both the domain specific data and the multimedia meta-data.

2. In the next stage this schema is populated with conceptual data and multimedia meta-data. The schema, the source data and the extraction algorithms may all change, so the stored data has to be maintained to keep its validity.

3. Concurrently to the maintenance stage, the search engine is used to query the Internet or intranet.

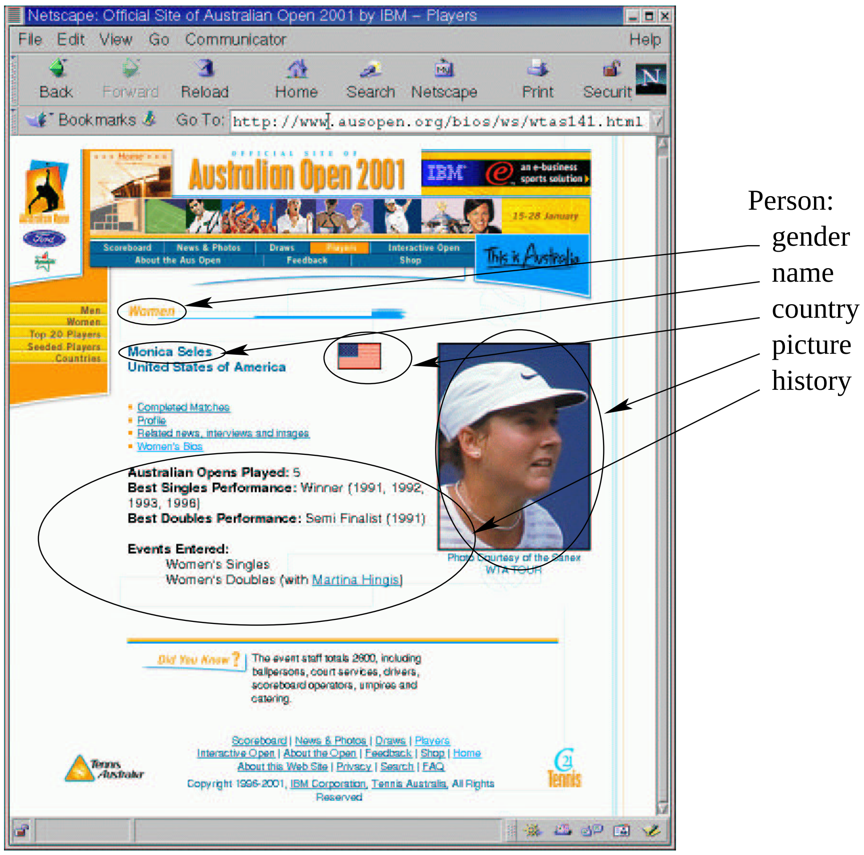
The integration of all this functionality allows the combination of both conceptual and content-based querying in the query stage. This integration is missing in traditional search engines. It gives users more powerful query primitives. For example, on the Internet scale, the user can ask the following query: *"show me all portraits embedded in pages containing keywords semantically related to the word 'champion' "*. In case of a more limited domain the developer can exploit the database advantages even further by applying specialised multimedia retrieval techniques to the data and, thus, create an even richer index. This allows the user to ask, in the tennis domain, for specific queries like: *"video shots in which Monica Seles approaches the net"*. The specialised video analysis needed for this can not be applied to the whole Internet, but is very well feasible for such a limited domain. The more detailed the index, the easier it is to provide users with concept-based search interfaces. At the physical layer the queries break down to structured database searches.

The remainder of the chapter is organized as follows. The next section presents a motivating example. It is used as a running example throughout this chapter. The system architecture section introduces shortly the three system levels. In the main part of the chapter describes the different system level roles, related to the lifecycle stages, in more detail. In the final sections global directions for (future) usage scenarios of the presented architecture and development trends for search engines in general are given.

MOTIVATING EXAMPLE

The Australian Open tennis tournament website [Ten01] is a good example of a site with a hidden semantical structure. Figure 1 shows a page of the site. Semantic concepts, *e.g.* gender and name, are clearly available in the source data used for this page. However, this semantic information is lost due to the translation of the source data into HTML, a presentation oriented format. As a result search engines can only see the page as a body of text and provide the ability to search for keywords in it. To find the history of Monica Seles, which is part of the biography page, the user has to come up with selective keywords, and hope that the information wanted is ranked high by the search engine. Alternatively, the query, could be formulated more precise using conceptual information, *i.e.* ask directly for the history of the player with name Monica Seles.

Apart from structural information, the site also contains multimedia fragments: audio files of interviews and even videos of tennis matches. As already stated in the introduction, formulating specific queries for these type of media is still a major research area. However, for specific domains, *e.g.* tennis, soccer or formula 1, good results are already possible. In such domains content-based retrieval of media items can be offered to the user, next to structural information. This enables them to issue queries like: *"Show me video shots of left-handed female players who have won the Australian Open in the past, and in which they approach the net."* This query


Netscape: Official Site of Australian Open 2001 by IBM – Players
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Securit
Bookmarks Go To: http://www.ausopen.org/bios/ws/wtas141.html
Australian Open 2001
IBM an e-business sports solution
15-28 January
Scoreboard | News & Photos | Draws | Players | Interactive Open
About the Aus Open | Feedback | Shop
This is Australia
Men
Women
Top 20 Players
Seeded Players
Countries
Person:
gender
name
country
picture
history

Women

Monica Seles
United States of America

- Completed Matches
- Profile
- Related news, interviews and images
- Women's Bios

Australian Opens Played: 5
Best Singles Performance: Winner (1991, 1992, 1993, 1996)
Best Doubles Performance: Semi Finalist (1991)

Events Entered:
Women's Singles
Women's Doubles (with Martina Hingis)

Photo Courtesy of the Sanex WTA TOUR

Did You Know? The event staff totals 2600, including ballpersons, court services, drivers, scoreboard operators, umpires and catering.

Scoreboard | News & Photos | Draws | Players
Interactive Open | About the Open | Feedback | Shop | Home
About this Web Site | Privacy | Search | FAQ
Copyright 1996-2001, IBM Corporation, Tennis Australia, All Rights Reserved

Figure 1: An annotated page from the Australian Open website

contains a mixture of structural information (the play hand is available in the players profile) and content-based information (shots showing a player approaching the net).

Throughout the description of the proposed system architecture the construction of a specialised search engine for the Australian Open website will be used as a running example. In the final sections the mixed query will indeed be answered. The next section shortly introduces the architecture of this system.

SYSTEM ARCHITECTURE

The proposed system architecture, as shown in Figure 2, can be divided into three abstraction levels. Independence between levels is used to transparently augment the source data with meta information and to make optimization decisions.

The whole system architecture is setup to model, populate, maintain and query the concept and content-based schema. Traditional search engines often favour specialised, Information Retrieval (IR) based, index structures (*i.e.* often inverted files [BYRN99]). This architecture is based on a DBMS (*i.e.* Monet [BK95]), which has by default generic support for schemas.

The following subsections introduce the various levels. More in-depth discussion of their functions and internal workings will be provided in the remainder of this chapter.

The conceptual level

The conceptual level focuses on limited domains of the Internet, *i.e.* intranets and large web-sites. The content provided on such domains is often highly related and structured. This aspect makes it feasible to determine a set of concepts, which adequately describe the content of the document collection at a semantic level.

The Webspace Method [ZA99] offers a methodology to model and search such a document collection, called a webspace. The Webspace Method defines concepts in a webspace schema using an object-oriented data model. This collection is stored as XML documents in the XML storage level of the global system architecture, see Figure 2. A strong correlation between the persistent documents is achieved, since the structure of each XML document describes (a part of) the webspace schema in turn. Actually each document contains a materialized view over the webspace schema; it contains both content and schematic information.

The webspace schema is used to formulate queries over the entire document collection. Novel within the scope of search engines and query formulation over document collections is that it allows a user to integrate information stored in different documents in a single query. Traditional search engines (*e.g.* AltaVista) are only capable to query the content of a single document at a time. Furthermore, using the Webspace Method specific conceptual information can be fetched as the result of a query, rather than a bunch of relevant document URLs.

Modeling data on the web is an active research area. Closely related to the Webspace Method is the Araneus project [MMA99] where existing database technology is applied to the web as well. The main difference with our approach is that we aim at combining concept-based search with content-based information retrieval, to come up with new query formulation techniques for data on the web. Others [AM98, CCD$^+$99], in line with XQuery [W3C01], use the structure of an XML document as their model. This allows them to search for patterns and structure in the XML data, but does not allow them to formulate content based queries. In [FG00, HTK00], about XIRQL and searching text-rich XML documents with relevance ranking, the focus is on XML and information retrieval. But these approaches do not use a conceptual schema to capture the knowledge of a limited domain upfront, and integrate the IR model only partially.

The logical level

The webspace schema, as formulated at the conceptual level, describes the information directly provided by the developer. However, a wealth of meta-data exists, hidden inside (associated) data. For example an attribute in the webspace can contain a large body of free text or refer to a video. In both cases algorithms exist to extract meta-data, *e.g.* the language of the text or the individual video shots. Such an algorithm may produce new information which enables the execution of another algorithm: forming a pipeline or chain of algorithms. For example: first the video shots are extracted, keyframes are found, and as a final step, some shots are labeled as closeups of a person.

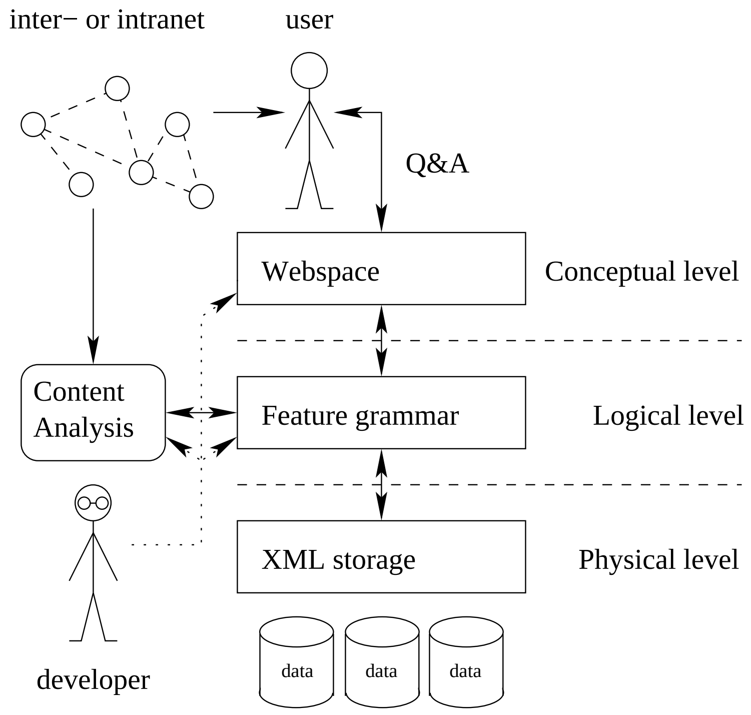


Figure 2: The global system architecture

By storing this meta-data the retrieval process can be enriched with content-based facilities. Opposed to the conceptual data, which exists mainly in the DBMS, the stored meta-data forms an index to external data (*i.e.* the raw multimedia data). As both conceptual data and meta-data are stored in the same DBMS, we will, throughout this chapter, also refer to the DBMS as index or meta-index.

As feature extraction algorithms may form a pipeline or chain, they have to be executed in the right order to populate the meta-index. To manage this order efficiently a high level description of the dependencies is needed. The Acoi system [KNW98, WSK99] provides the framework to specify this description and to use it to create, fill, and maintain the meta-index. The description is specified in the *feature grammar language*, which forms the system's core. A feature grammar describes the relationships between meta-data items and detectors (*i.e.* meta-data extraction algorithms) in a set of grammar rules.

Several other systems have proposed high level descriptions of the algorithm and data relationships. In the Mirror system [VEK98, Vri99] algorithms are encapsulated in daemon (CORBA) objects. The daemons issue a *get_work* query to the database, extract the meta-data for the returned objects and then issue a *finish_work* query to commit their results.

Another framework is MOODS [GYA97]. MOODS is based on the extension of an object oriented schema with semantic objects. The unique aspect of a semantic object is its processing graph. In a processing graph algorithms are linked together. The developer defines a split point in the processing graph. The first part is executed data-driven when a new object instance is inserted into the database. The last part is executed demand-driven during query processing. Additionaly, the object can be extended with inference rules, which can be checked during query processing.

The main novel aspects of feature grammars are the approach to maintenance and the role of contextual knowledge. Adding one individual daemon is easy in Mirror, but adding a pipeline of daemons is much harder. This due to the fact that all context knowledge is embedded in the *get_work* query. Each new daemon in the pipe has to check if all the previous daemons have already been executed. This affects the generality of the algorithm's implementation.

In contrast to MOODS, where each semantic object has its own processing graph, the system contains only one processing graph: the feature grammar. The grammar specifies both the order of algorithms and the inference rules, and these can be mixed freely. Having one processing graph enables the easier construction of a scheduler: which analysis dependencies and uses them to control the index maintenance.

Feature grammars have a sound theoretical basis to analyse the algorithm dependency structure and to construct efficient systems from it.

The physical level

The conceptual data and the multimedia meta-data has to be stored persistently. The physical mapping of this data, which has on its way through the previous levels been translated into a set of XML documents, should be optimized for fast query response. This involves not only the mapping of the structure and content of the XML documents into an relational system, but may also lead to fragmentation of the data or indexes over several database servers.

The literature provides numerous ways to map XML documents (or, more generally, semi-structured data) to database instances (see [BDHS96, GW97, KM00, AQM$^+$97, Sof00, SYU99, STH$^+$99, FK99, DFS99]). Most approaches the authors have come across fall into the following categories:

- *DTD-less* mappings: No additional information is needed to map the document or documents to a database instance.
- Mappings that *require* a DTD. Some mappings use a DTD to derive a storage structure for documents. As a consequence, all the documents must conform to the DTD.
- Mappings that require a description in a more *complex data-definition language* such as XSchema or one of those described in [BC00].

Another important criterion according to which we can categorize mappings is the question whether a mapping may change the database schema or not:

- *Document-dependent* mappings generally imply a global database schema. Since the schema depends on the document, it may be large and change when the database is updated. In pathological cases the schema can grow so large that managing the meta-information in the database becomes a severe bottleneck.
- *Document-independent* mappings usually maintain a heap on which all documents are stored.

The authors designed and implemented a mapping which is DTD-less to preserve generality and document-dependent for ease of implementation and execution speed in their test bed. This approach proved useful since the dynamic nature of feature grammars calls for an extremely flexible storage method. DTD dependence would be a show-stopper for many applications.

The following sections will describe the roles of these shortly introduced levels during the lifecycle of the search engine.

MODELING THE INDEX

The first stage of a search engine's lifecycle is the development of the initial index model. As already indicated in Figure 2, the developer does not have to model all the system levels: the focus is on the upper levels.

Conceptual modeling

In the Webspace Method [ZA00a] concepts are modeled in an object-oriented fashion. The webspace schema models the concepts in terms of classes, attributes of classes, and associations over classes. Together the concepts give a semantic description of the content available in a webspace. Each document then forms a materialized view over the webspace schema: describing a part of the webspace. Within a document web-objects are defined along with the relations between them, forming instantiations of classes and associations from the webspace schema.

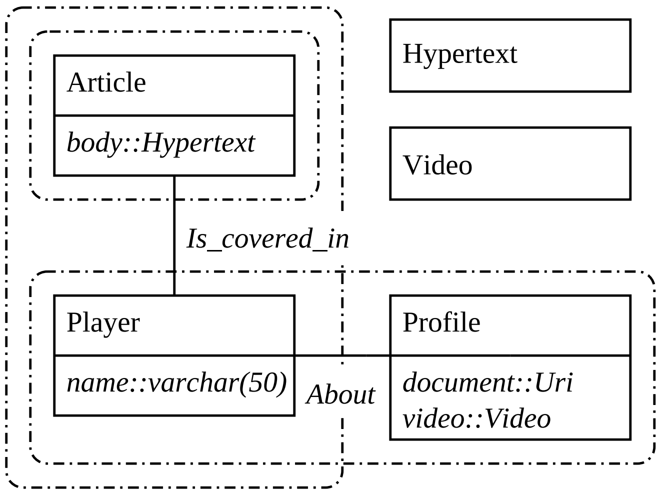


Figure 3: Fragment of a webspace schema.

Figure 3 shows a fragment of the webspace schema for the Australian Open webspace. Five class concepts are defined there: `Article`, `Player`, `Profile`, `Hypertext`, and `Video`. Furthermore the figure contains some attribute concepts (`body`, `name`, `document`, and `video`), and two association concepts (`Is_covered_in`, and `About`). The slashed boxes show what concepts are used within the three kinds of documents that appear in the webspace. The overlap of concepts used in different documents provides the necessary conditions for conceptual search over a webspace, and possibly combines information stored in several documents into one query. For the integration with content-based information retrieval we allow the conceptual schema to be extended with all kinds of multimedia types (*i.e.* text, images, video or audio). Each conceptual web-object can refer to various multimedia objects.

While the webspace schema resides at the semantical level of a webspace, the document level of a webspace is usually formed by a collection of XML documents, which form materialized views over a webspace schema. When a webspace is setup from scratch the author will create the documents using a specialized webspace authoring tool [ZA00a]. The tool guides the author through the entire design process. However, if a webspace is based on an already existing document collection, a reengineering process can be invoked. The process extracts the relevant data from the (HTML-)documents on a website, and stores it in XML-documents, which form a correct view over the webspace schema. The documents for the Australian Open search engine are generated in this manner, using a special purpose feature grammar.

Logical modeling

When the multimedia types, *e.g.* `Hypertext` and `Video`, in the conceptual model are known, the developer can start with the construction of a feature grammar. To have a clear understanding how a feature grammar works, we will have a closer look at the formal background of the feature grammar language.

The formal basis of feature grammars is the theory of context-free grammars [Lin97]. A context-free grammar G is defined as a quadruple $G = (V, T, S, P)$, where V is the set of variables, T the set of terminal symbols, $S \in V$ is the special symbol called the start variable and P is the set of productions. A feature grammar is basically a context-free grammar with some extensions related to a special set of variables called detectors [SWK99]. A detector is a variable bound to a feature extraction algorithm. A feature grammar is thus defined as a quintuple $G = (V, D, T, S, P)$, where D is the set detectors, and the start variable can be either a normal variable or a detector: $S \in (V \cup D)$.

The production rules are the heart of the grammar: they specify how the grammar transforms one token string to another, and through this they define a language associated with the grammar. A production rule consists of a left-hand and a right-hand side: $x \rightarrow y$. The left-hand side consists of one symbol: $x \in (V \cup D)$. The

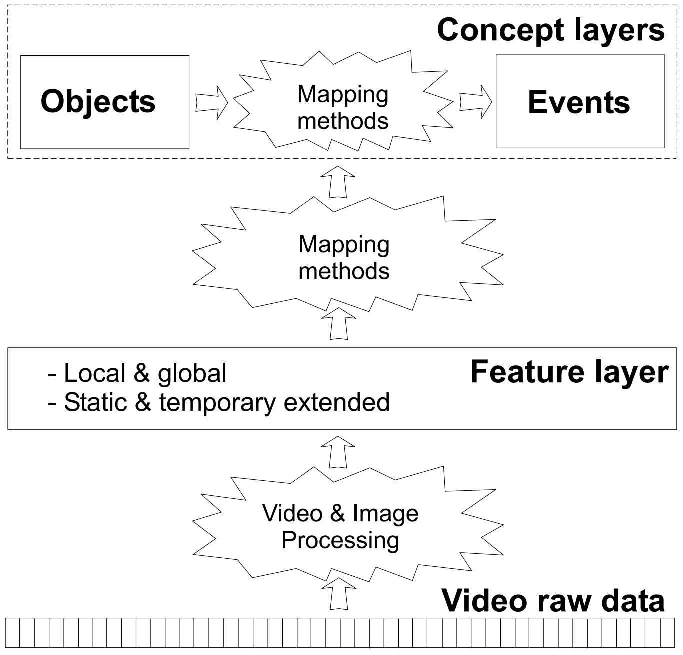


Figure 4: The Cobra video data model.

right-hand side contains a sequence of symbols: $y \in (V \cup D \cup T)^*$.

Several rules may contain the same symbol at their left-hand side. These rules are considered alternatives. To compress the grammar specification an extended notation can be used to combine several of the alternatives into one grammar rule. The feature grammar language uses regular right part grammars [LaL77] as the extended notation.

The rules associated by their left-hand side to a detector describe the output of this detector. The required input of a detector is part of its declaration. Those input tokens are specified as paths into the parse tree. These paths can only refer to preceding symbols. This adds a limited amount of context sensitivity to the grammar.

In the following section we will see how relevant meta-data can be extracted from a tennis video. Then we will return to the feature grammar language and see how it can be used to model this extraction process.

Tennis video modeling and analysis In order to explore video content and provide a framework for automatic extraction of semantic content from raw video data, we propose the *COntent-Based RetrievAl* (COBRA) video data model [PJ00]. The model is independent of feature/semantic extractors, providing flexibility by using different video processing and pattern recognition techniques for that purposes. At the same time it is in line with the latest development in MPEG-7, distinguishing four distinct layers within video content (see Figure 4): the raw data, the feature, the object, and the event layer. The object and event layers consist of entities characterized by prominent spatial and temporal dimensions respectively.

In the remainder of this section we will focus on meta-data extraction in the particular domain of tennis videos. First, we briefly describe tennis video segmentation and shot classification. Then, we move to player segmentation and tracking. Finally, we present the features that are extracted from the player shape and discuss different techniques for event recognition.

A typical video of a tennis match consists of different shots. The algorithm that segments the video into different shots is implemented as a segment detector. The shot boundaries are detected using differences in color histograms of neighboring frames. For each shot, we extract its dominant color. The dominant color that occurs most frequently is supposed to be the tennis court color. By analyzing the dominant color of all shots, our segmentation algorithm is generalized to work with different classes of tennis courts without changing any

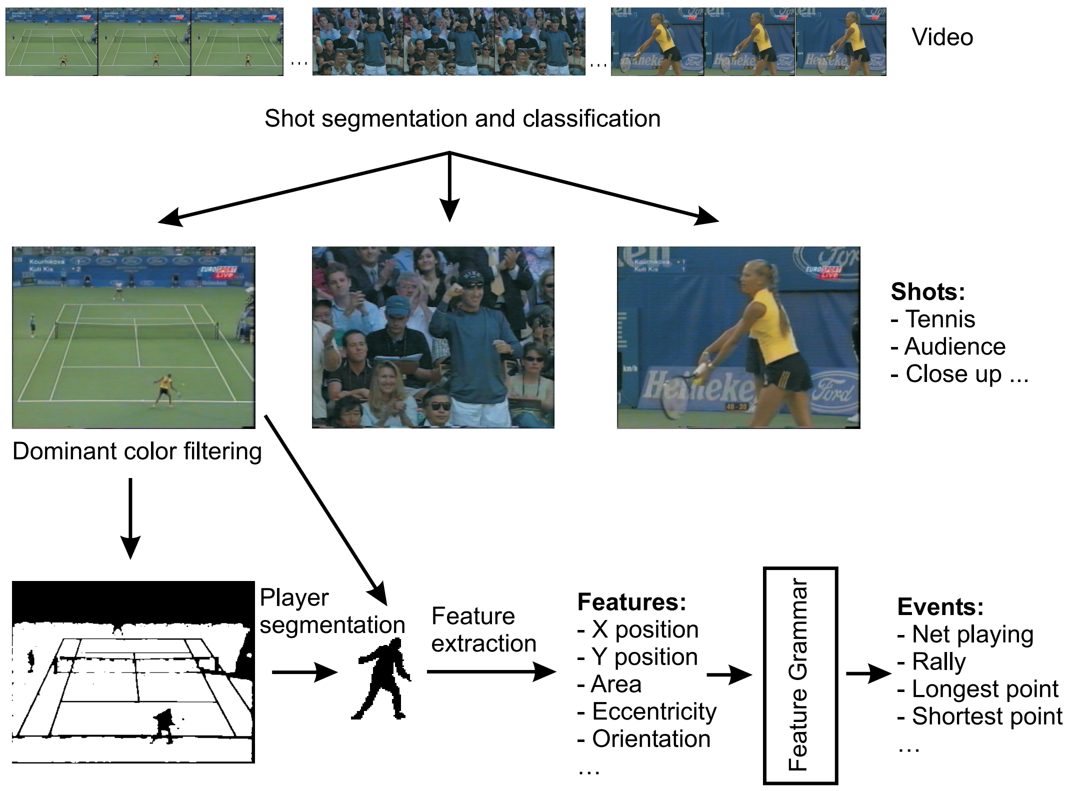


Figure 5: Tennis video analysis.

parameters in our algorithm.

The same algorithm encapsulates shot classification. As shown in Figure 5, the algorithm classifies shots in four different categories: tennis, close-up, audience, and other. Selection of the video shots containing a tennis court from others is necessary, since we are mostly interested in finding playing events. The court shots are recognized based on dominant color, as explained. A shot is classified as a close-up, if it contains a significant amount of skin colored pixels. For the classification, we also use entropy characteristics, mean and variance.

Player segmentation and tracking is done by the tennis detector. Using estimated statistics of the tennis field color, the algorithm does the initial quadratic segmentation of the first image of a video sequence classified as a playing shot (for details see [PJZ01]). In the next frames, we predict the player position and search for a similar region in the neighborhood of the initially detected player.

In the same algorithm we extract features characterizing the shape of the segmented player's binary representation. Having the specific case of the human figure in this particular application, we extract special parameters trying to maximize their informative value. Besides the player's position, we extract the dominant color, and standard shape features such as the mass center, the area, the bounding box, the orientation, and the eccentricity.

As a video is a temporal sequence of pixel regions at the physical level, it is very difficult to explore its semantic content, *i.e.* detect and recognize video events automatically. Very often, different models and techniques are required for event understanding. Furthermore, as it is shown in the literature, different techniques are more suitable for different kinds of events.

To provide automatic extraction of concepts (objects and events) from visual features mentioned above, the COBRA video model is enriched with few extensions. Each extension encapsulates a different knowledge-based technique for concept inference.

First, the model is extended with object and event grammars. These grammars are aimed at formalizing the descriptions of high-level concepts, as well as facilitating their extraction based on spatio-temporal reasoning.

```
 1: %start      MMO(location);
 2:
 3: %detector   header(location);
 4: %detector   header.init();
 5: %detector   header.final();
 6:
 7: %detector   video_type primary == "video";
 8:
 9: %atom       url;
10:
11: %atom url   location;
12: %atom str   primary;
13: %atom str   secondary;
14:
15: MMO         : location header mm_type?;
16: header      : MIME_type;
17: MIME_type   : primary secondary;
18: mm_type     : video_type video;
```

Figure 6: A fragment of the video feature grammar.

The syntax of rules is described in [PJ01], where they were implemented within the query engine. Here we implement them within the feature grammar using detectors, as we will see later in the chapter.

As the model provides a framework for stochastic modeling of events, other possibilities are to exploit the learning capability of *Hidden Markov Models* or *Dynamic Bayesian Networks* to recognize events in video data automatically (see for an example [PJ01]).

In the following section we will return to the feature grammar language and construct a feature grammar for the Australian Open search engine. This grammar forms an instantiation of the COBRA framework for this specific domain. We will focus on the video content abstractions introduced, but the grammar is extensible for other multimedia types and even other query domains.

Tennis video feature grammar The first step in the extraction process is to retrieve the multimedia object from its location, and to decide if it is indeed a video. The part of the feature grammar doing exactly this is shown in Figure 6. In the following paragraphs feature grammar language constructs will be explained using this grammar.

Production rules The lines 15 to 18 contain the production rules, while the preceding lines declare certain properties and abilities of the symbols found in these rules. Notice the extended notation used for the optional symbol `mm_type` in the `MMO` rule.

Start symbol Line 1 states that `MMO` is the grammar's start symbol. The detectors produce the tokens consumed by the parser on-the-fly, however, optionally a minimum set of tokens has to be available to start this process. The arguments to the start declaration specify this minimum set. In this case the set contains only the `location` of the multimedia object.

Detectors The first detector declaration we find at line 3. This declaration specifies that `header` is a detector which needs as input a `location`. The arguments of a detector declaration consist of paths, which always refer to available nodes in the parse tree. Furthermore, detectors come in two types: blackbox and whitebox. The `header` detector is an example of the first type. This means that the feature grammar developer has to implement the algorithm, for example in the programming language *C*. Only the input and output of the implementation are known: there is no information on the internal workings, hence the type name: blackbox detector. For the `header` detector this implementation contacts the `HTTP` server, specified by the `location`, and gets the header information for the object. From this information it filters out the primary and secondary MIME type of the object, as required by the production rules (see line 17). These data items are returned in

```
20: %detector   xml-rpc::segment(location);
21: %detector   xml-rpc::tennis(location,begin.frameNo,
   end.frameNo);
22:
23: %detector   netplay some[tennis.frame](
24:               player.yPos <= 170.0
25:             );
26:
27: %atom flt   xPos,yPos,Ecc,Orient;
28: %atom int   frameNo,Area;
29: %atom bit   netplay;
30:
31: video       : segment;
32: segment     : shot*;
33: shot        : begin end type;
34: begin       : frameNo;
35: end         : frameNo;
36: type        : "tennis" tennis;
37: type        : "other";
38: tennis      : frame* event;
39: frame       : frameNo player;
40: player      : xPos yPos Area Ecc Orient;
41: event       : netplay;
```

Figure 7: Another fragment of the video feature grammar.

the token stack of the parser, the parser can validate them against the production rules and move them into the parse tree. More information on the parser internals will be given in the description of the next lifecycle stage.

Special detectors Detectors may need specific initialization or finalization, *e.g.* for allocating and freeing memory. Lines 4 and 5 contain the declaration of special `init` and `final` detectors which handle the W3C WWW library [W3C00]. The `init` detector is called the first time the parser encounters the specified symbol. If the initialization has been successful the `final` detector will be called when the parser finishes. Next to these special detectors two others exist: `begin` and `end`. They are called every time the specified symbol is encountered.

Whitebox detectors In line 7 we find an instantiation of the second type of detectors: the `video_type` whitebox detector. In contrast to a blackbox detector the complete specification of a whitebox detector is part of the feature grammar. This specification takes the form of a boolean predicate over the information in the parse tree. In this specific declaration it checks if the primary MIME type is video.

Atoms For the terminal symbols in the grammar their Abstract Data Type (ADT) can be specified. This is done by the atom declarations in line 11 to 13. Line 9 shows the declaration of a new ADT (*i.e.* the `url`), which should be supported by the lower system levels.

This small grammar shows the basic ingredients of a feature grammar: a start symbol (`MMO`), variables (*e.g.* `MIME_type`), terminals (*e.g.* `location`) and detectors (*e.g.* `header`). To make the developer's life easier this basis has been extended with shortcuts and additional functionality. While we have already seen several extensions, more will be used for the modeling of the tennis specific video processing (see Figure 7). Notice that this is just a partial feature grammar, not all features described in the discussion of the tennis video analysis are shown.

External detectors As described in the previous section the video is first segmented into shots. They are classified as a tennis shot or other type of shots, *e.g.* on the basis of the tennis court detection. This segmentation step is embedded in the `segment` detector, declared in line 20. Instead of linking the *C* code into the parser, as is the case with the `header` detector, this detector is implemented externally (and may even run on a different machine). To contact the external implementation the XML-RPC protocol is used, indicated by the prefix `xml-rpc::`. Code for the protocol instantiation is generated. Several other connection protocols for external detector implementations are supported: from plain system calls to using distributed objects through CORBA.

Literals The `segment` detector outputs the begin and end frame numbers of the shots and a string indicating the shot type. Using this type information and the literals in the alternative rules for the `type` symbol the right alternative can directly be validated. In the case of a tennis shot it will choose the rule in line 32 and call the `tennis` detector. This detector is once again implemented as a remote procedure that finds the tennis player in each video frame of the tennis shot. The player's position, together with additional features are returned to the parser.

Quantifiers The `event` rule contains extra concepts which can be derived from the basic player features. One concept is specified by the `netplay` whitebox detector. It shows the use of the *some* quantifier to determine if the player approaches the net in at least one frame of this shot. The other quantifiers (*i.e. all* and *one*) are also supported.

As announced before this grammar is easily extensible. New multimedia types can be (and indeed are) added by providing alternative rules for the `mm_type` symbol. Furthermore, if the `segment` detector would be able to recognize soccer shots, an alternative `type` rule could trigger a whole sequence of soccer specific detectors. During the description of the indexing stage we will return to these kind of model changes and see how we can efficiently support them.

This particular grammar is a domain and implementation specific instantiation of the COBRA framework. Blackbox detectors are used to translate the raw video data into features. The mapping methods from the feature layer into the concept layers are implemented by a mixture of black- and whitebox detectors.

Although the feature grammar language has more constructs (*e.g.* modules, structure sharing by references) which make the model even more powerful, those are not discussed in this chapter. The presented model is already powerful enough to support the tennis video abstraction process.

POPULATING AND MAINTAINING THE INDEX

Using the schemata and algorithms developed, the system tools can be used to populate the index with the desired conceptual structures and the related meta-data, and thus enable the user to issue his or her queries. The same tools are able to maintain the index when the schemata and algorithms evolve over time.

Conceptual indexing

In the indexing phase, a crawler retrieves the source documents from a webspace. During this process, the conceptual information stored in these documents is extracted. This is done by the *web object retriever*, which reconstructs the web-objects, and the relations among them, stored in the documents, given the corresponding webspace schema. Next, for each web-object its multimedia items are handed over to the logical level (see the next section), where they are analyzed further. The web-objects are, in the form of XML documents, passed on to the persistent storage level. In [ZA00b] this process is described in more detail.

Logical indexing

The conceptual level passes its multimedia references on to the logical level. This multimedia information is augmented with meta-data, and stored in the meta-index. Creating and maintaining the meta-index is done by exploiting the dependencies in the feature grammar. This is done in two basic ways: data- and demand-driven. We use the first approach to populate the meta-index, while a combination of both approaches is used to localize changes and incrementally maintain the data in the index. Lets start with the population phase.

Feature Detector Engine The data-driven approach is handled by a special parser, the *Feature Detector Engine* (FDE), generated from the feature grammar. Before we have a look at the inner workings of this parser we will shortly recall some parser theory.

A parser explains a sentence, w, which consists of a sequence of tokens, through its grammatical derivation. This means that the parser proofs that the sentence is a member of the language generated by the feature grammar G. Formally this language is describes as the set $L(G) = \{w \in T^* : S \overset{*}{\Rightarrow} w\}$. The result of the parser is a comprehensive description of the productions used in the parsing process: the parse tree. This parse tree contains all the tokens found in the input sentence placed in their hierarchical context.

The current FDE implementation uses a recursive descent algorithm (for a discussion of parser algorithms see [GJ98]). This means that the FDE works top-down and left-to-right by trying to prove that the start symbol of the grammar is valid. While doing this the FDE manages a stack of tokens (the input sentence), a parse tree, and a set of feature detectors. Tokens are matched against the production rules and move from the stack to the parse tree. Upon its way through the production rules the FDE encounters the detector symbols and executes their associated algorithms. The algorithms produce new tokens which are pushed on the token stack. In the end the parser proves the start rule valid, in which case the parse tree can be dumped as an XML-document, or invalid.

To support backtracking, the FDE needs to maintain several versions of the token stack. Simple copying of stacks places a high burden on both memory consumption and CPU time. However, many copies share the same suffix of tokens. Those suffixes can be shared and thus limit the resource consumption, *i.e.* a suffix tree is created. This is done in the same manner as the reuse of stack prefixes in [Tom86].

The FDE can be used to fill the initial version of the meta-index. But, the real benefit of a feature grammar shows when the feature detector algorithms change and the index has to be updated. To support index maintenance, the feature grammar can then be used with a different execution paradigm.

Feature Detector Scheduler Opposed to the FDE, which currently uses a strictly data-driven paradigm, the *Feature Detector Scheduler* (FDS) uses the feature grammar also in a demand-driven manner. Based on the dependency graph, deduced from the grammar rules, the FDS can localize the effects of the evolutionary changes, and trigger incremental parses. An incremental parse means that new branches are added to an existing parse tree or its old branches are updated. The main goal of this process is to prevent the regeneration, and the associated calls to detectors, of the complete parse tree.

Figure 8 shows a dependency graph fragment corresponding to the feature grammar fragment in Figure 6. The node types in the graph correspond to the basic symbol types of a feature grammar. There are several types of dependencies between the nodes, indicated by different edge types:

1. The first one is the *sibling dependency*: these edges connect nodes which appear together in the right-hand side of a rule. The `header` symbol appears together with `location` and `mm_type` in a `MMO` rule. The siblings influence the validity of each other as the rule as a whole should be succesfull: *e.g.* `header` depends on `location` and vice versa.

2. The *rule dependency* describes the dependency between the left-hand and right-hand symbols of a production rule. The left-hand symbol depends on the validity of the last obligatory symbol (*i.e.* the last symbol with a lower bound greater then zero). So `MMO` depends on the validity of `header` and not on the validity of `mm_type`, as it is optional.

3. The last dependency is due to the, optional, parameters of a detector: the *parameter dependency*. For example: the `header` detector needs the `location` as input.

All incremental changes to the grammar can be traced back to a change in a detector or the minimum token set. If, for example, new atoms are added to the grammar they have to be produced by one of the detectors in the grammar or provided as initial tokens to the FDE.

The impact of changes in a detector implementation is indicated by a version. Such a version consists of three levels. The lowest level indicates a *correction revision*. Such a revision will not lead to invalidation of any nodes in the current stored parse trees, so the FDS does not have to take any action. Changes of the next level, the *minor revisions*, will lead to invalidation of the partial parse trees. However, the data may still be used to answer queries. Those revalidations are scheduled with a low priority. High priorities are used for invalidations caused by *major revisions*. In these cases the changes are so severe that the stored data has become unusable.

The invalidation of nodes in the parse tree, and the scheduling of their revalidation, is handled by the FDS using the following steps (assuming that the `header` detector implementation has changed):

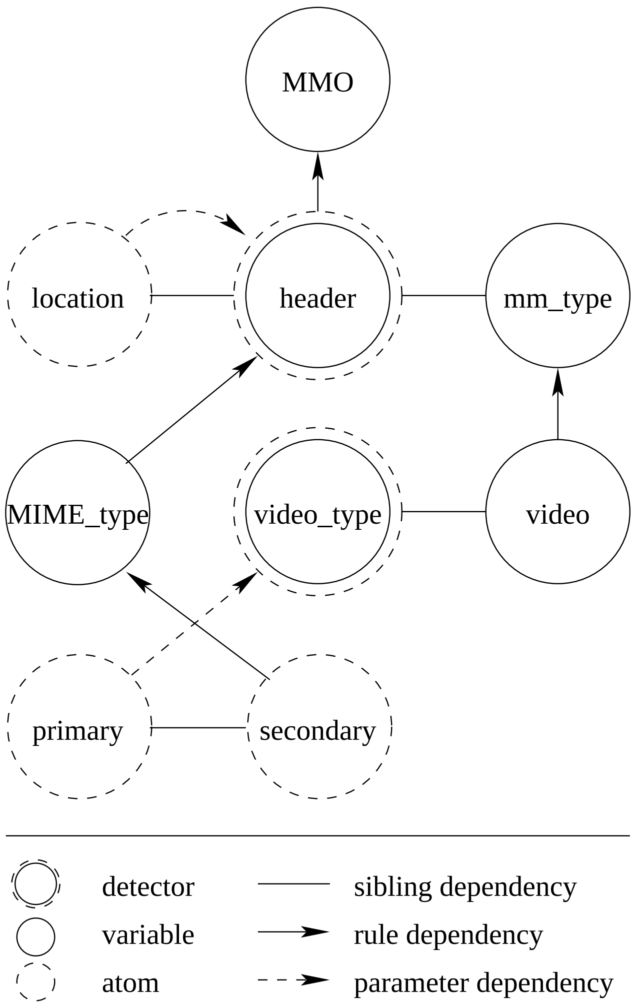


Figure 8: A fragment of the video dependency graph.

```
<image key="18934" source="http://.../seles.jpg">
  <date> 999010530 </date>
  <colors>
    <histogram> 0.399 0.277 0.344 </histogram>
    <saturation> 0.390 </saturation>
    <version> 0.8 </version>
  </colors>
</image>
```

Figure 9: Example document

1. The FDS will invalidate all partial parse trees which have an instantiation of a `header` symbol as root. This will involve `header`, `MIME_type`, `secondary` and `primary` nodes, as can be derived by following the rule and sibling dependencies downward. For these parse trees incremental parsing processes are scheduled, which can be handled by the FDE. The followup action of the FDS is determined by the result of such a process.

2. If the subtree is still valid, the parameter dependencies are checked for modification. If there has been a modification the dependent detector needs to be revalidated. If, for example, the `primary` MIME type has changed the `video_type` detector will become invalid.

3. If the subtree has become invalid, the rule and sibling dependencies are followed upward to the first detector or start symbol which is marked invalid. The FDS will repeat the whole procedure for these invalid symbols.

Next to schema changes the source data may change. The FDS uses a special detector associated to the start symbol to determine if the complete stored parse tree has become invalid due to changes of the source data, in which case the parse tree will be regenerated.

Physical indexing
Both the conceptual and logical level pass on their data in the form of XML documents. The physical level takes care of persistently storing the documents in such a way that insertion of new documents, incremental updates of them and retrieval (over multiple documents) are efficient. To do so the structure of the XML documents is mapped to a database schema.

XML to database schema mapping XML documents are commonly represented and thought of as syntax trees. We first recall some of the usual terminology needed to work with XML documents. In the sequel, **string** and **int** denote sets of character strings and integers and **oid** a set of unique object identifiers. We can now define an XML document formally (a/b denotes that b is a child element or descendant of a, $a[b]$ means that b is an attribute of a, see [W3C98b] for details): An *XML document* is a rooted tree $d = (V, E, r, label_E, label_A, rank)$ with nodes V and edges $E \subseteq V \times V$ and a distinguished node $r \in V$, the root node. The function $label_E : V \rightarrow \mathbf{string}$ assigns string labels to nodes, *i.e.* element denominations. $label_A : V \rightarrow \mathbf{string} \rightarrow \mathbf{string}$ assigns pairs of strings, attributes and their values, to nodes. Character Data ([P]CDATA) are modeled as a special attribute of $cdata$ nodes, $rank : V \rightarrow \mathbf{int}$ establishes a ranking to allow for an order among sibling nodes.

Figure 9 shows an XML fragment; Figure 10 displays the corresponding tree (arrows indicate XML attribute relationships, straight lines XML element relationships).

Before we discuss techniques how to store a tree as a database instance, we introduce the notion of *associations*. They are used to cluster semantically related information in a single relation and constitute the basis for the Monet XML Model; the aim of the clustering process is to enable efficient scans over semantically related data, *i.e.* data with the same element ancestry, which are the physical backbone of declarative associative query languages like SQL. A pair $(o, \cdot) \in (\mathbf{oid} \times \mathbf{oid} \cup \mathbf{oid} \times \mathbf{string} \cup \mathbf{oid} \times \mathbf{int})$ is called an *association*.

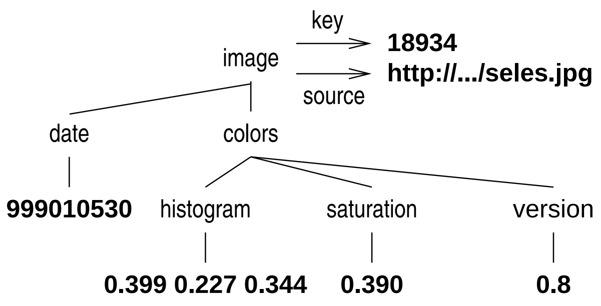


Figure 10: Syntax tree

The different types of associations play different roles: associations of type $\mathbf{oid} \times \mathbf{oid}$ represent parent-child relationships. Both kinds of leaves, attribute values and character data are modeled by associations of type $\mathbf{oid} \times \mathbf{string}$, while associations of type $\mathbf{oid} \times \mathbf{int}$ are used to keep track of the original topology of a document. For a node o in the syntax tree, we denote the sequence of labels along the path (vertex *and* edge labels) from the root to o as $path(o)$.

Paths describe the position of the element in the graph relative to the root node and we use $path(o)$ to denote the *type* of the association $(\cdot, o)$. The set of all paths in a document is called its *Path Summary*, which plays a central role in our query engine. The main rational for the path-centric storage of documents is to evaluate the ubiquitous XML path expressions efficiently; the high degree of semantic clustering achieved distinguishes our approach from other mappings (see [FK99] for a discussion). Our approach is to store all associations of the same 'type' in one *binary relation*. A relation that contains the tuple $(\cdot, o)$ is named $R(path(o))$. We can now define the mapping.

Definition 1 *Given an XML document* d*, the* Monet transform *is a quadruple* $M_t(d) = (r, \mathbf{E}, \mathbf{A}, \mathbf{T})$ *where* r *remains the root of the document and*

$$\begin{aligned}
\mathbf{E} &= \bigcup_{(o_i,o_j,s)\in\tilde{E}} R(path(o_i)/s)\langle o_i, o_j\rangle, \\
\mathbf{A} &= \bigcup_{(o_i,s_1,s_2)\in label_A} R(path(o_i)[s_1])\langle o_i, s_2\rangle, \\
\mathbf{T} &= \bigcup_{(o_i,i)\in rank} R(path(o_i)[rank])\langle o_i, i\rangle\big)
\end{aligned}$$

Encoding the $path$ to a component into the name of the relation achieves a significantly higher degree of semantic clustering than implied by plain data guides [GW99]. In other words, we use $path$ to group semantically related associations. A direct consequence of the decomposition schema is that we do not need to cope with irregularities induced by the semi-structured nature of XML, which are typically taken care of with NULLs or overflow tables [DFS99]. The next subsection will deal with the machinery we need to convert documents to Monet format and bulkload them efficiently. Note that we are able to reconstruct the original document given its Monet transform $M(d)$, *i.e.* for an XML document d there exists an inverse mapping M_t^{-1} such that d and $M_t^{-1}(M_t(d))$ are isomorphic.

A discussion of the inverse mapping can be found in [SKWW00]. The Monet transform enables an object-oriented perspective, *i.e.* object as node in the syntax tree, which is often more intuitive to the user and is adopted by standards like the DOM [W3C98a]. Particularly in querying, approaches that bear strong similarities with object-oriented techniques have emerged. Given the Monet transform, we have the necessary tools at hand to reconcile the relational perspective with the object-oriented view: An *object* o is a set of strong and

```
<image key="18934" source="http://.../seles.jpg">
<image><date>
<image><date>" 999010530 "
<image></date>
<image><colors>
<image><colors><histogram>
<image><colors><histogram>" 0.399 0.277 0.344 "
<image><colors></histogram>
<image><colors><saturation>
<image><colors><saturation>" 0.390 "
<image><colors></saturation>
<image><colors><version>
<image><colors><version>" 0.8 "
<image><colors></version>
<image></colors>
</image>
```

Figure 11: Path sequences in the example document

weak associations $\{A_1\langle o, o_1\rangle, A_2\langle o, o_2\rangle, \dots\}$. This perspective is used when we need to DOM-like traversals or run edit-scripts against the database.

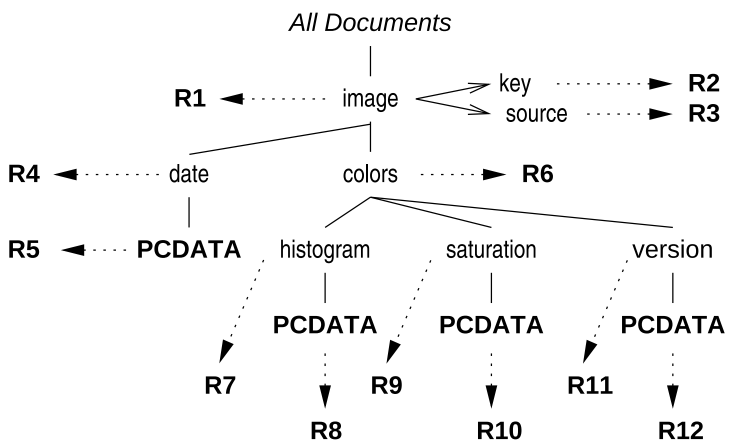
Populating and maintaining the database There are two basic notions of interest that we are going to discuss in this section: populating a database from scratch, *i.e.* bulk load, and incremental insertion of new data into an already existing database. However, we use the same technique for both cases. Let us consider an example first.

There are two standard ways of accessing XML documents: (1) a low-level event-based, called SAX [Meg01], which scans an XML document for tokens like start tag, end tag, character data *etc.*; user supplied functions are called on encountering for each type of token. The advantage of the SAX parsers is they only require minimal resources to work efficiently. (2) There is also a high-level DOM interface [W3C98a] which provides a standard interface to parse trees of complete documents. In terms of resources, the memory consumption of DOM trees is much higher, linear in the document size; thus, it may happen that large documents exceed the size of available memory. We propose a bulk load method that has only slightly higher memory requirements than SAX – $O($*height of document*$)$ – but still keeps track of all the contextual information it needs and which would otherwise only available through a DOM interface. Thus, the memory requirements of the bulkload algorithm we use are very low as it does not materialize the complete syntax tree to generate insertion statements.

Since Monet XML stores complete paths, the bulk load routine need to track those paths. We do this by organizing the path summary as a *schema tree* which we use to map efficiently paths to relations. Each node in the schema tree represents a database relation and contains a tag name and reference to the relation. Figure 11 shows the path sequences generated by combining the SAX events of the parser and a stack.

We can now attach OIDs to every tag when we put it on the stack. This way, we are able to record all path instances in the documents without having to maintain a syntax tree in (main) memory – an advantage that lets us process very large amounts of documents in relatively little memory. The function that performs the actual insertion is $insert(R, t)$ where R is a reference to a relation and t is a tuple of the appropriate type. A first naïve approach would thus result in the following sequence of insert statements:

1. $insert(sys, \langle o_1, image\rangle)$
2. $insert(R(image[key]), \langle o_1, \text{“}18934''\rangle)$
3. $insert(R(image[source]), \langle o_1, \text{“}http://.../seles.jpg''\rangle)$
4. $insert(R(image/date), \langle o_1, o_2\rangle)$



R1: /image **R3:** /image[source] **R5:** /image/date/PCDATA
R2: /image[key] **R4:** /image/date ...

Figure 12: Schema tree of example document

5. $insert(R(image/date/pcdata), \langle o_2, \text{“}999010530''\rangle)$

6. ...

Note that this sequence of insert statements requires us to hash the *complete* path to a relation name. By exploiting the hierarchical structure of the schema tree we can do much better. We can do away with much of the hashing if we keep track of the context, *i.e.* the current node, in the schema tree: when we encounter a start tag, we look at the sons of the current context. There are now two cases: (1) we find a son that represents the tag, or, (2) there is no son that represents the tag. In the first case, we simply push the son on the stack, thus making it the current context, and store the OIDs in the relation that is associated with the son. If we don't find a child node that represents the tag, then the path does not yet exist in the database. In this case, we create a node and the respective relation and continue processing with the newly created node as in (1). If we encounter an end tag we 'pop' the stack twice, *i.e.* pop both the start and corresponding end tag.

We note that we can easily extend the bulkload procedure to records *extents* of elements, *i.e.* the textual position of a start tag and its corresponding end tag. We can use the extent mechanism to implement a multi-attribute schema for documents which come along with a DTD by reserving slots for every $1:1$ parent-child relationship specified in the DTD and flushing tuples once the end of their extent is reached. Note that this mechanism allows us, as we promised, to convert documents independent of the presence of a DTD and of the previous database content. This feature connects nicely to the extensibility of Feature Grammars which call for incremental updates.

The described mapping functions well for the general retrieval case, however, for specific query types, *e.g.* full text retrieval, specific accelerators can be hooked in.

Optimization support for full text retrieval One multimedia type in the webspace schema is `Hypertext`. Optimization of full text retrieval is hooked in the system by adding triggers for the text type in the physical level. We support a variant of the $tf \cdot idf$ ranking model, derived from the well founded probabilistic retrieval model of [Hie98]. The $tf \cdot idf$ model is well known in the area of IR research. As described in [VW99] we transparently integrate the necessary relations into our database:

term index A binary relation `T(term-oid, term)` mapping terms on oids, *i.e.* the vocabulary. Note that the terms to be stored in this relation actually will be the corresponding stems. Stop terms are expected to be filtered out.

document index A binary relation `D(doc-oid, doc-url)` which maps the known documents/web objects (in particular their URLs) on oids. This relation actually is not special to the full text part of the document collection, but is used as a global notion of the entire document collection known to the system.

document term list For each pair of document and term oids in the collection, a pair oid is listed in the ternary relation `DT(doc-oid, term-oid, pair-oid)`.

term frequency A binary relation `TF(pair-oid, tf)` listing the number of times the pair with the given oid occurs in the collection (*i.e.* the number of times a certain term occurs in a given document). Note that this relation can be derived from the `DT`-relation.

inverse document frequency We list the idf for each known term in a binary relation `IDF(term-oid, idf)`. The idf of a term is defined as $\frac{1}{df}$, where df represents the number of documents in which a term occurs. Note that this relation can be derived from the `TF`-relation, too.

The incremental full text indexing process is started every time the XML storage manager has parsed a certain number of document bodies. The document-term pairs are by then listed in `DT`. The stemmed and stopped terms have been stored in `T` and the document URLs are listed in `D`. Using these three basic relations the `TF` and `IDF` relations are updated incrementally.

Since the `TF` and `DT` relations are prone to grow huge, even when compression techniques are applied, we horizontally fragment these relations. The fragmentation criterium we use is derived from the `IDF` relation. Since terms with a high idf (and therefore a low df) are expected to be more significant to the ranking of a document that contains that term, compared to terms with a low idf, we fragment on descending idf. Note that the less interesting lower idf terms typically are the most computationally expensive terms (their high df means they have many related tuples in the `TF` relation). Moving these less interesting but more expensive terms to the end of the fragment set allows us to exploit this knowledge later on during query optimization. Next to this horizontal fragmentation on idf we distribute the `TF` (and corresponding `IDF` tuples) over several database servers, by assigning parts on a per-document basis to the available hosts. Again, we can benefit from this later on during query optimization to achieve almost perfect shared nothing parallelism which facilitates (almost) unlimited scalability.

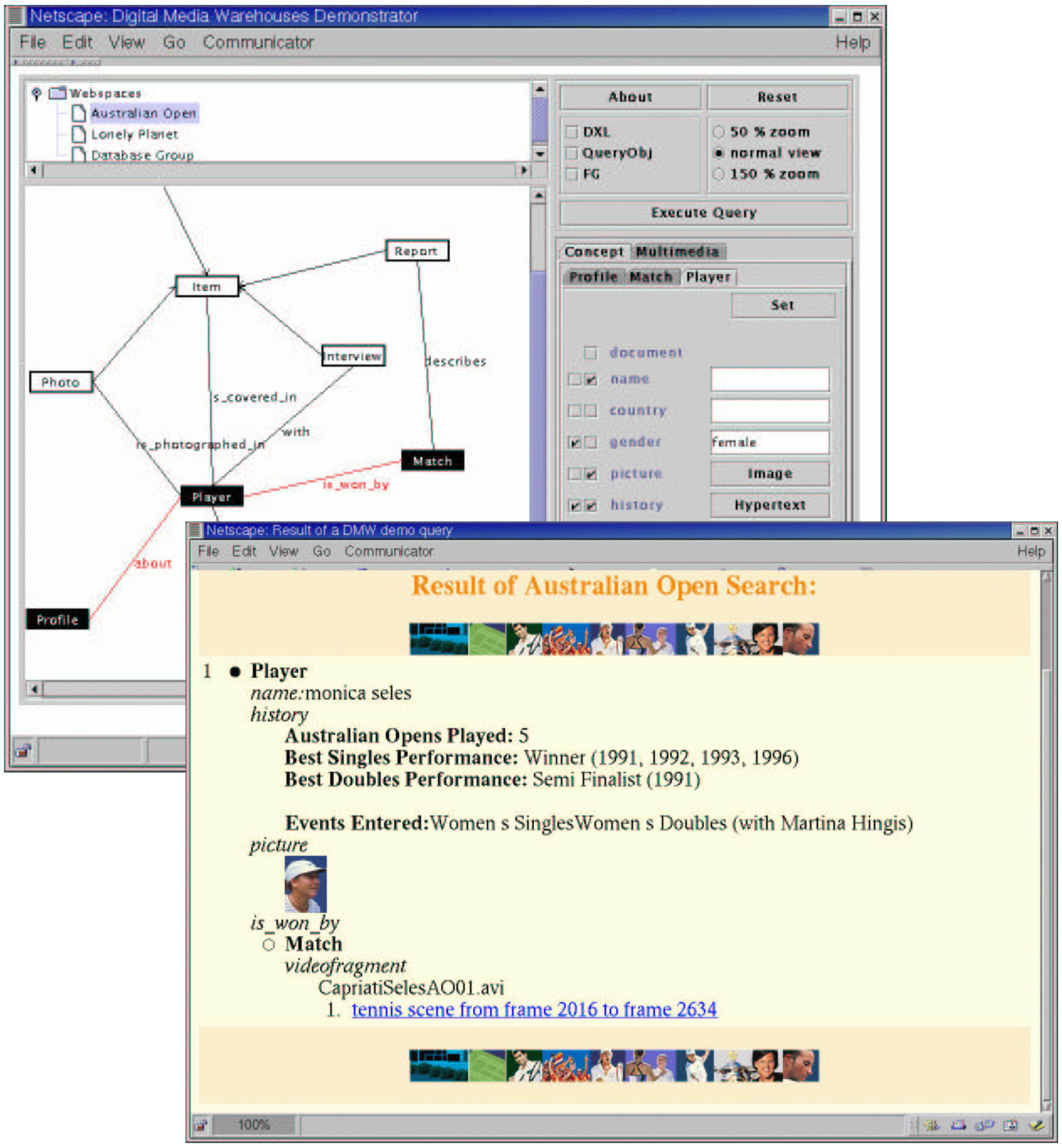
QUERYING THE INDEX

This section describes, again per focus area, the query aspects of our system. The system supports integrated querying, using the conceptual structure and the meta-data as described before. As shown in Figure 2 the user interacts with the conceptual level.

Here comes out the main advantage of using conceptual modeling for data from a limited web environment. It allows us to query a webspace by its schema. This introduces new query formulation techniques for web-like environments, so far only available within the database environment.

Given the Australian Open webspace one can formulate queries like: *"Show me video shots of left-handed female players, who have won the Australian Open in the past, and in which they approach the net."*. To support end-users to compose such queries by hand a graphical user interface visualizes the webspace schema, and allows a user to easily formulate his information need. In [BWZ$^+$01, ZA01] this interface is described. Figure 13 shows the interface for formulating this specific query.

The phrase *"who has won the Australian Open in the past"* has been turned into a free text search on the word *"Winner"* in the `history` attribute of the `Player` class, which is of the multimedia type `Hypertext`. The `netplay` event, which was defined in the video feature grammar (see Figure 7), is used to determine which shots match the phrase *"approach the net"*. Combined with the structural information from the webspace schema the answer obtained is shown in Figure 13.



Result of Australian Open Search:

1 ● **Player**
*name:*monica seles
history
Australian Opens Played: 5
Best Singles Performance: Winner (1991, 1992, 1993, 1996)
Best Doubles Performance: Semi Finalist (1991)

Events Entered: Women s SinglesWomen s Doubles (with Martina Hingis)
picture

is_won_by
○ **Match**
videofragment
CapriatiSelesAO01.avi
1. tennis scene from frame 2016 to frame 2634

Figure 13: An example query and its answer.

```
1: html    : title? body? anchor* ;
2: body    : &keyword+;
3: anchor  : &MMO embedded link? alternative?;
4: keyword : word;
```

Figure 14: A fragment of the Internet feature grammar.

Under the hood of the system the query is translated into an XML representation, which in its turn is translated into the query algebra of the storage engine. During this translation statements using the optimization hooks, like implemented for full text retrieval, are inserted.

Use of the optimized full text retrieval support The central database server, which first pushes the terms, *e.g. "Winner"*, though the stemmer and stopper, matches it against the `T` relation to reduce the content query portion to oids in the system's vocabulary space.

The generated query tells the query optimizer that we are only interested in the 10 best ranking results that match the selection clauses. Since we know that the IR meta-data has been distributed on a per document basis, we can push this top-10 request to the distributed nodes, along with the query terms oids.

On each distributed node the query optimizer might use top-*N* optimization techniques to determine the best way to compute the local top-10 (see [Blo00, BCBA01]). Note that both database top-*N* optimization techniques (*e.g.* [DR99, CK98]) and IR top-*N* optimization techniques (*e.g.* [Bro95]) can be exploited here. Next to the typical IR cut-off techniques we are working on a quality model that allows the query optimizer to estimate the quality degrade resulting from a-priori ignoring fragments with lower idf [BHC+01]: IR is inherently uncertain allowing other probabilistic query optimization tricks than those which typically can be used in exact match querying (see [DR99]).

After having computed and aggregated the necessary $tf \cdot idf$ values, each distributed node returns a result of the form `RES(doc-oid,rank)` to the central DBMS. The central node merges the top-10 rankings into a large ranking. This master ranking is combined with the other result parts of the query and the final top-10 is computed.

Note, that it is up to the query optimizer whether the ranking should be unlimited and the results merged afterwards or the ranking should be restricted to only a limited domain. For example, if one is only interested in articles about the Australian Open tennis tournament from a certain author, this might be (from a query optimization point of view) a very interesting a-priori restriction of the ranking candidate set.

This ends the discussion of the various roles the system levels play during the lifecycle of the search engine. The final sections of this chapter contain discussions on future work, using this architecture, future trends for search engines in general, and conclusions.

FUTURE WORK AND TRENDS

The system architecture, presented in this chapter, is implemented and tested on our running example: a search engine for the Australian Open website. This limited domain showed a clear use of the conceptual level and specific multimedia extraction algorithms. Other case studies have been based on the Lonely Planet and a computer science faculty websites. However, the system is applicable to the Internet as a whole. Either by replacing the specific webschema by a very generic, and thus not so semantically rich one, or by giving the user the possibility to use a direct interface on top of the logical level (see for example the Acoi website [Cen01]). Furthermore, the system can be applied to non-internet related collections, *e.g.* the digital library of a museum [NWP+01].

A fragment of the production rules of a feature grammar for the Internet is shown in Figure 14. The rules for HTML pages show how the hierarchical structure of the grammar can be turned into a graph using references to the start symbol. In this way the linking structure of the web is modeled.

Of course this grammar can contain multimedia extraction algorithms as detectors. For example: a photo/graphic classifier for images [ASF97], language detection for HTML pages [TNO01], face detection [LH96] etc. This would allow queries like: *"show me all portraits embedded in pages containing keywords semantically related to the word 'champion' "*.

As the examples already indicate the meta-data extracted from the multimedia objects is, in this case, of a generic nature. However, by using a feature grammar, a better grip on the Internet context of a multimedia object is possible. For example: when the content of a webpage is classified as a sports topic, rules in the grammar can be used to steer the processing of videos embedded in the page, towards sport specific detectors (*e.g.* the discussed tennis video analysis).

A similar close connection can be realized between the information at the conceptual level and the logical level. From the webspace schema a feature grammar can be derived, containing references to, for example, the `MMO` start symbol. In this way detectors could refer back to information comming from the conceptual level, enabling thus the use of even more specific detectors.

By providing a conceptual schema the user can not only use concepts previously hidden in the HTML page, but also the semantic links between pages. Recent research provides more general knowledge about the nature of this link structure of the web [Kle00, DKM$^+$01]. Search engines, like Tomea [Ask01] already does, will use this knowledge to provide the user with better ranked or grouped query results. In this way they implicitly use the semantic value of the links.

The use of contextual knowledge is becoming more important, many search enigines provide facilities to search for images, and some even for video and audio, based on keywords found on the pages which embed or link to these images. This a start of the use of multi-modal queries, which use the meta-data of multimedia objects found in the "neighborhood" of the searched objects.

So, while in our system architecture the schema is made explicit for a specific domain, more search engines are starting to implicitly exploit the semantic knowledge of the Internet by using both generic structure and context. Those generic approaches will never be able to provide the user with the powerful query primitives a specific engineered conceptual schema provides. Initiatives related to Semantic Web technology aim at making this knowledge explicit in a larger context, *i.e.* by combining topic related websites using onthologies, and thus be able to combine their source data for generic purposes.

CONCLUSION

In this chapter, we have described a system architecture, which overcomes the limitations of the current generation of digital library search engines, by combining conceptual search and content-based retrieval. This combination offers more sophisticated query primitives to the users, allowing them to formulate semantically rich queries. The architecture knows three levels: the conceptual, logical, and physical level.

The highest level uses the Webspace Method to describe a conceptual schema. This schema places the source data in a specific domain, and enables thus the conceptual searches.

Attributes in the webspace schema can be of a multimedia type. On the logical level, feature grammars are used to augment instances of these multimedia types with meta-data. This meta-data enables the user to use content-based retrieval methods in his or her queries. This data is produced by extraction algorithms, called detectors. Those detectors and the data they produce are managed by a feature grammar. Feature grammars are not only able to fill the meta-index, but can also maintain it efficiently.

Both the data described by the webspace schema and the feature grammar need to be stored persistently. Both levels maintain their information as XML documents. The path-based storage schema is optimized to store and (incrementally) update the documents in a fast and efficient way.

Next to the main levels the described system provides special hooks, which lead to two key system properties: flexibilty and scalability.

At the logical level detectors can be plugged into the system, which implement all kinds of multimedia feature extraction algorithms. The system architecture contains both a parser and a scheduler which, on the basis of a feature grammar, handle the change management of these algorithms. This provides the system the flexibility to maintain the search engines index incrementally.

The scalability of the system is due to the lowest layer where special optimization measures can be hooked in to speed up query processing by, for example, distributing the query workload over several database engines.

The final conclusion is that the described system architecture is well suited for developing specialised, flexible and scalable digital library search engines.

ACKNOWLEDGEMENTS
We wish to thank Arjen de Vries, Niels Nes and Martin Kersten for numerous discussions of the various topics related to the presented system and their work on miЯRor, a content-based multimedia database, and (early versions of) the Acoi system tools.

This chapter combines work from the Advanced Multimedia Indexing and Searching (AMIS), and Digital Media Warehouses (DMW) research projects.

References

[AM98] G.O. Arocena and A.O. Mendelzon. WebOQL: Restructuring documents, databases and webs. In *proceedings of Fourteenth IEEE International Conference on Data Engineering (ICDE98)*, 1998.

[AQM+97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[ASF97] Vassilis Athitsos, Michael J. Swain, and Charles Frankel. Distinguishing photographs and graphics on the world wide web. In *Workshop on Content-Based Access of Image and Video Libraries*, Puerto Rico, June 1997.

[Ask01] Ask Jeeves, Inc. *Tomea Search from Ask Jeeves*. `http://teoma.com/`, 2001.

[BC00] A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. *SIGMOD Record*, 29(1):68–79, 2000.

[BCBA01] Henk Ernst Blok, Sunil Choenni, Henk M. Blanken, and Peter M. G. Apers. A selectivity model for fragmented relations in information retrieval. CTIT Technical Report Series 01-02, CTIT, Enschede, The Netherlands, February 2001.

[BDHS96] P. Buneman, S. B. Davidson, G. G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[BHC+01] Henk Ernst Blok, Djoerd Hiemstra, Sunil Choenni, Franciska de Jong, Henk M. Blanken, and Peter M.G. Apers. Predicting the cost-quality trade-off for information retrieval queries: Facilitating database design and query optimization. In *Tenth International Conference on Information and Knowlegde Management (ACM CIKM '01)*. ACM SIGIR/SIGMIS, ACM Press, November 2001. To appear.

[BK95] Peter A. Boncz and Martin L. Kersten. Monet: An Impressionist Sketch of an Advanced Database System. In *Basque International Workshop on Information Technology, Data Management Systems (BIWIT'95)*. IEEE Computer Society Press, July 1995.

[Blo00] H. E. Blok. Top N optimization issues in MM databases. In Stefan Conrad and Holger Riedel, editors, *Proceedings of the EDBT 2000 Ph.D. Workshop*, pages 74–77. EDBT, March 2000.

[Bro95] Eric W. Brown. Execution Performance Issues in Full-Text Information Retrieval. Ph.D. Thesis/Technical Report 95-81, University of Massachusetts, Amherst, 1995.

[BWZ+01] H. E. Blok, M. Windhouwer, R. van Zwol, M. Petkovic, P.M.G. Apers, M.L. Kersten, and Jonker

W. Flexible and scalable digital library search. In *Proceedings of the twenty-seventh International Conference on Very large Data Bases (VLDB'2001)*, Rome, Italy, September 2001.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[CCD$^+$99] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. Xml-gl: a graphical language for querying and restructuring XML documents. In *proceedings of the International World Wide Web Conference (WWW)*, pages 1171–1187, 1999.

[Cen01] Centre for Mathematics and Computer Science (CWI). *Acoi home page*. `http://www.cwi.nl/~acoi/`, 2001.

[CK98] Michael J. Carey and Donald Kossmann. Reducing the Braking Distance of an SQL Query Engine. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *Proceedings of the 24th VLDB Conference*, pages 158–169. VLDB, Morgan Kaufmann, 1998.

[DFS99] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing Semistructured Data with STORED. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 431–442, 1999.

[DKM$^+$01] Stephen Dill, Ravi Kumar, Kevin McCurley, Sridhar Rajagopalan, D. Sivakumar, and Andrew Tomkins. Self-similarity in the web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 69–78, Rome, Italy, 2001.

[DR99] Donko Donjerkovic and Raghu Ramakrishnan. Probabilistic Optimization of Top N Queries. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *Proceedings of the 25th VLDB Conference*, pages 411–422. VLDB, Morgan Kaufmann, September 1999.

[FG00] N. Fuhr and K. Grossjohann. Xirql – an extension of xql for information retrieval. In *Proceedings of the ACN SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

[FK99] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[GJ98] Dick Grune and Ceriel Jacobs. *Parsing Techniques - A Practical Guide*. Vrije Universiteit, Amsterdam, 1998.

[GW97] R. Goldman and J. Widom. Dataguides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 436–445, 1997.

[GW99] R. Goldman and J. Widom. Approximate DataGuides. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, Jerusalem, Israel, 1999.

[GYA97] James Griffioen, Raj Yavatkar, and Robert Adams. A framework for developing content-based retrieval systems. In Mark T. Maybury, editor, *Intelligent Multimedia Information Retrieval*. The AAAI Press and The MIT Press, 1997.

[Hie98] Djoerd Hiemstra. A Linguistically Motivated Probabilistic Model of Information Retrieval. In Christos Nicolaou and Constantine Stephanidis, editors, *Proceeding of the second European Conference on Research and Advanced Technology for Digital Libraries (ECDL'98)*, pages 569–584. Springer-Verlag, 1998.

[HTK00] Y. Hayashi, J. Tomita, and G. Kikui. Searching text-rich xml documents with relevance ranking. In *Proceedings of the ACN SIGIR 2000 Workshop on XML and Information Retrieval*, Athens, Greece, July 2000.

[Kle00] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 2000.

[KM00] M. Klettke and H. Meyer. XML and Object-Relational Database Systems - Enhancing Structural

Mappings Based on Statistics. In *International Workshop on the Web and Databases (WebDB)*, pages 63–68, 2000.

[KNW98] M. L. Kersten, N. Nes, and M. A. Windhouwer. A Feature Database for Multimedia Objects. In *ERCIM Database Research Group Workshop on Metadata for Web Databases*, pages 49–57, Bonn, Germany, May 1998.

[LaL77] W. R. LaLonde. Regular right part grammars and their parsers. *Communications of the ACM*, 20(10):731–741, 1977.

[LH96] Michael S. Lew and Nies Huijsmans. Information theory and face detection. In *Proceedings of the International Conference on Pattern Recognition*, pages 601–605, Vienna, Austria, 1996.

[Lin97] Peter Linz. *An Introduction to Formal Languages and Automata - Second Edition*. Jones and Bartlett Publications, 1997.

[Meg01] D. Megginson. SAX 2.0: The Simple API for XML. `http://www.megginson.com/SAX/`, 2001.

[MMA99] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of xml. *IEEE Data Engineering Bullettin, Special Issue on XML*, September 1999.

[NWP$^+$01] F. Nack, M. A. Windhouwer, E. Pauwels, M. Huijberts, and L. Hardman. The Role of High-level and Low-level Features in Semi-automated Retrieval and Generation of Multimedia Presentations. Technical Report INS-R0108, CWI, Amsterdam, The Netherlands, June 2001.

[PJ00] M. Petkovic and W. Jonker. A framework for video modelling. In *proceedings of Intl. Conf. on Applied Informatics*, Innsbruck, Austria, February 2000.

[PJ01] M. Petkovic and W. Jonker. Content-based video retrieval by integrating spatio-temporal and stochastic recognition of events. In *proceedings of IEEE Intl. Workshop on Detection and Recognition of Events in Video*, Vancouver, Canada, 2001.

[PJZ01] M. Petkovic, W. Jonker, and Z. Zivkovic. Recognizing strokes in tennis videos using hidden markov models. In *proceedings of Intl. Conf. on Visualization, Imaging and Image Processing*, Marbella, Spain, 2001.

[SKWW00] A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *International Workshop on the Web and Databases (WebDB)*, pages 47–52, Dallas, TX, USA, 2000.

[Sof00] Software AG. *Tamino – Technical Description*. Available at `http://www.softwareag.com/tamino/technical/description.htm`, 2000.

[STH$^+$99] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the International Conference on Very Large Data Bases*, pages 302–314, 1999.

[SWK99] A. R. Schmidt, M. A. Windhouwer, and M. L. Kersten. Feature Grammars. In *Proceedings of the International Conference on Systems Analysis and Synthesis*, Orlando, FL, USA, August 1999.

[SYU99] T. Shimura, M. Yoshikawa, and S. Uemura. Storage and Retrieval of XML Documents Using Object-Relational Databases. In *Database and Expert Systems Applications*, pages 206–217. Springer, 1999.

[Ten01] Tennis Australia. *The Australian Open Homepage*. `http://www.ausopen.org/`, 2001.

[TNO01] TNO Institute of Applied Physics. *DRUID project home page*. `http://dis.tpd.tno.nl/druid/`, 2001.

[Tom86] Masaru Tomita. *Efficient parsing for natural language*. Kluwer Academic Publishers, 1986.

[VEK98] Arjen P. de Vries, Brian Eberman, and David E. Kovalcin. The design and implementation of an infrastructure for multimedia digital libraries. In *Proceedings of the 1998 International Database*

Engineering & Applications Symposium, pages 103–110, Cardiff, UK, July 1998.

[Vri99] A. P. de Vries. *Content and Multimedia Database Management Systems*. PhD thesis, Centre for Telematics and Information Technology, 1999.

[VW99] Arjen P. de Vries and Annita N. Wilschut. On the Integration of IR and Databases. In *8th IFIP 2.6 Working Conference on Data Semantics 8*, 1999.

[W3C98a] W3C. Document Object Model (DOM). available at `http://www.w3.org/DOM/`, 1998.

[W3C98b] W3C. Extensible Markup Language (XML) 1.0. available at `http://www.w3.org/TR/1998/REC-xml-19980210`, 1998.

[W3C00] W3C. *Libwww - the W3C Protocol Library*. `http://www.w3.org/Library/`, 2000.

[W3C01] W3C. XQuery: A query language for XML. Technical report, World Wide Web Consortium (W3C), February 2001.

[WSK99] M. A. Windhouwer, A. R. Schmidt, and M. L. Kersten. Acoi: A System for Indexing Multimedia Objects. In *International Workshop on Information Integration and Web-based Applications & Services*, Yogyakarta, Indonesia, November 1999.

[ZA99] R. van Zwol and P. M. G. Apers. Searching documents on the intranet. In *proceedings of Workshop on Organizing Webspace*, Berkeley (CA), USA, August 1999.

[ZA00a] R. van Zwol and P. M. G. Apers. Using webspaces to model document collections on the web. In *proceedings of WWW and Conceptual Modelling(WCM00), in conjunction with ER2000*, Salt Lake City, October 2000.

[ZA00b] R. van Zwol and P. M. G. Apers. The webspace method: On the integration of database technology with information retrieval. In *In proceedings of CIKM'00*, Washington, DC., November 2000.

[ZA01] R. van Zwol and P. M. G. Apers. Complex query formulation over web-based document collections, 2001.