Acoi: A System for Indexing Multimedia Objects

Menzo Windhouwer windhouw@cwi.nl
Albrecht Schmidt albrecht@cwi.nl
Martin Kersten mk@cwi.nl

CWI
Kruislaan 413
NL-1098 SJ Amsterdam
The Netherlands

Abstract

In this paper, we present a system that combines independent feature detector programs with multimedia database technology to provide a semantic rich index to multimedia data items on the World Wide Web.

First, we introduce a grammatical framework, called feature grammars, which forms the indexing schema. Feature grammars are an extension of context-free grammars with active symbols (*e.g.* multimedia feature detectors) that may invoke feature detector programs. The Acoi system reads in the grammar, compiles it and executes it against a data source, *e.g.* a multimedia object. The derived parse tree is used as an index to this data source. Its structure closely resembles that of semi-structured (XML) documents.

Then, we present the architecture of our implementation on top of Monet, our extensible main memory database system. In this implementation, feature grammars are used as a description of the execution sequence of the indexing program. We show how the resulting parse tree can be stored efficiently in Monet. A SQL-like query language enables users to use the feature grammar as a schema for query formulation to retrieve both index values and the original data sources.

Throughout the paper, we illustrate the concepts with a running example of a grammar used for indexing HTML pages and multimedia objects on the World Wide Web.

Keywords: multimedia databases, feature grammars, multimedia indexing

1 Introduction

With the explosive growth of the amount of linked multimedia objects, *i.e.* the building blocks of the World Wide Web, the 'lost in cyberspace' problem aggravates. There are numerous approaches to alleviate the problem studied in the context of digital libraries [AR97, WAS98]. Topics include navigation and query formulation facilities, and database support for both of them. In this paper we focus on the database support for multimedia query facilities.

Searching for objects on the web is mainly supported by constructing a meta-index. Well known meta-indexes are the search engines, *e.g.* AltaVista, widely used to find specific HTML pages. These indexes are based on keywords extracted from the page description. The equivalent of a keyword for an image or audio object, however, is ill-defined. Depending on the type of user queries, they range from color histograms or pixel values, to real world concepts, like "this image shows a desert landscape". Building an useful meta-index for multimedia objects depends on the ability to support such a multitude of views, from low level features to real world concepts, on the same object.

Supporting multiple views is also needed to describe the capabilities of the data. Multimedia data comes in a diversity of multimedia types, formats and sources; not all of them will support the same set of features and concepts.

Our vision is to develop a framework which supports building a search facility with the same simplicity as AltaVista, but is geared towards multimedia data. In this paper we describe the design and system architecture of this system. Its fundamental framework uses a declarative specification language, called *feature grammars*. Using such a language is important, because it allows us to use the same framework for several purposes. It provides a schema and orientation for an user issuing queries or browsing the multimedia index database. Moreover, it provides an abstract specification of the behavior of the program responsible for the physical indexing of the multimedia objects.

Feature grammars also meet the requirements for building an useful meta-index. Because they have the ability to express multiple indexing views for one multimedia type, thus enabling different views depending on its format or source on the web. Another salient feature of this declarative language is the seamless integration of third-party feature extraction modules into the agent, which allows the needed support for a wide range of features and concepts.

The rest of this paper is organized as follows. Section 2 outlines the concept of feature grammars, which form the theoretical foundation of our system. The following section (Section 3) takes a closer look at the system architecture and its realization. In Section 4, the application of the system in the context of multimedia searching on the web is described. We conclude with a review of related research and indicate topics for future work.

2 Feature grammars

Feature grammars were introduced in [KNW98] and their foundations are outlined in [SWK99]. Their introduction was motivated by the desire to provide a grammatical framework for the specification and implementation of an autonomous multimedia indexing agent, as well as a concise model for query resolution.

We choose context-free grammars as the basis of the specification. On the one hand, they provide concise descriptions of a large class of objects; on the other hand, one can easily derive a parser from their structure. Furthermore, context-free grammars have another desirable property if we associate their structure with concept trees: we can regard the hierarchies of symbols in the parse tree as semantic categories and, therefore, use them as a knowledge base to resolve queries.

Before describing the special properties of a feature grammar, we will have a look at its formal definition. A feature grammar G is a quintuple $G = (V, D, T, S, P)$, where V are the variables, D a set of special variables called detectors (which will be described in the following paragraphs), T the terminal symbols, $S \in (V \cup D)$ is the start symbol and $P \subseteq (V \cup D) \times (V \cup D \cup T)$ are the production rules. Figure 1 shows a sample feature grammar with the following instantiations:

$$
\begin{aligned}
V &= \{web\_object, web\_body, anchor, surround, before, after\} \\
D &= \{web\_header, page\_type, web\_page\} \\
T &= \{url, content\_type, title, section, word, alt\} \\
S &= \{web\_object\} \\
P &= \{web\_object \rightarrow url\ web\_header\ web\_body?, web\_header \rightarrow \ \ldots\}
\end{aligned}
$$

This simple feature grammar shortly describes the structure of HTML pages as used in the World Wide Web. This example grammar will be used, in this and following sections of the paper, to describe how we make use of context-free grammars and extended them.

The most prominent extension of feature grammars to context-free grammars are grammar rules whose left-hand side symbols are programs, called *detectors*. These programs interrupt the parsing process by reading from the same input stream as the parser. They also write their output back onto this input stream, which then again is consumed by the parser and evaluated against the right hand side of the detector rule. In the example (see Figure 1), the output of the *web_header* detector must return a token that matches *content_type*.

```
%atom          str url, content_type, title, section, word, alt;

%detector      web_header(url);
%detector      page_type
                 select true
                 from web_object
                 where content_type = "text/html";
%detector      web_page(url);

%start         web_object;

web_object  →  url web_header web_body?;
url         →  "http://([^:/]*)(:[0-9]*)?/?(.*)";
web_header  →  content_type;
web_body    →  page_type web_page;
web_page    →  title? anchor*;
anchor      →  web_object surround?;
surround    →  section? before alt? after;
before      →  word*;
after       →  word*;
```

Figure 1: Feature grammar for a simple indexing agent

In [SWK99] we proved that extending a context-free grammar with these kind of detectors does not change the basic properties of a context-free grammar, so a feature grammar is still a context-free grammar. Which means that using a compiler-compiler this feature grammar can be translated into an executable parser. This parser acts as an agent which starts a conventional parsing process of an object until a detector occurs in a grammar rule. When a detector rule is evaluated the corresponding program is executed.

Detector programs produce the meta-index information needed for two main retrieval categories: content-based and concept-based retrieval [AGJ97, OS95]. Content-based retrieval aims at retrieving multimedia objects by values, so called features, which are directly derived from the raw object data. Concept-based retrieval aims at using real-world concepts for retrieval. To achieve this goal objects are annotated with the concepts. In the best case the concepts can be completely derived from the low level features using a declarative rule. In other cases human intervention may be needed to annotate the objects manually.

To fit the different needs of these retrieval techniques the following detector types are available:

black-box detectors are written in a programming language, which gives the developer the means to access the data source in any way he or she finds appropriate and to compute the desired features or even ask an user to provide some information; an example of a black-box detector is the *web_header* detector;

white-box detectors are part of the feature grammar specification and consist of queries on the (partially) built parse tree, these queries derive new information from known feature values or concepts and therefore describe meta-concepts; the example grammar contains the white-box detector *page_type*.

An additional type of detectors takes over the role of lexical analyzers in the parsing process, they are called atom detectors. In this case regular expressions are used to describe valid token values for the specific atom (*e.g.* the url rule in Figure 1). If the atom is too complex to be described by a regular expression (like images, MIDI files etc.) a black-box detector can be used to validate the value.

The use of feature grammars enables us to describe the relations between multiple feature values, concepts and the original multimedia data sources. However as identified in the introduction, the feature grammar does not only need to support a wide range of feature values and concepts, but needs also the ability to describe the support of these features and concepts by a data source.

Context-free grammars already provide a partial solution to this problem. They allow several alternative right hand sides to describe one non-terminal, however, only one of these alternatives can be true. In the case of feature grammars this is not the case. Each alternative describes a different, but valid, view on the multimedia object. Alternatives can be used in feature grammars not only to describe different logical views (a low level feature value view and a higher level concept view of the same object), but also different views depending on data source or format (a GIF view or a JPEG feature view). So alternatives in context-free grammars are exclusive, while they are inclusive in feature grammars. In both cases the extended notation, in this specific form also called *regular right part grammars* [LaL77], can be used to group some of these alternatives into one rule. This is illustrated in the *web_page* rule in the example grammar.

A parsing process on basis of feature grammars containing these kind of alternative rules will result in a parse tree, which closely resembles the graphs used to describe *semi-structured data* [FLM98]. One characteristic of semi-structured data is that information maybe incomplete. The alternatives in a feature grammar can lead to this information incompleteness: *e.g.* some *web_pages* don't contain a *title*. Another characteristic is that they have an inherent structure, *i.e.* the structure can be derived from the data itself. In the case of feature grammars the structure is not derived from the data, but is prescribed by the grammatical rules. But the resemblance of the parse trees to semi-structured data is large enough to allow us to apply a plethora of techniques to browse, visualize and store them (*e.g.*, see [GW99], [xFr99]).

Building and maintaining a database of parse trees for indexing purposes is not a static process. The structure of the meta-index with feature values and concepts will be subject to change, as researchers will discover alternative or new methods to index a multimedia object. So an important aspect of maintaining a semantic index is to have a means to gradually extend it when new features or concepts are to be integrated into the index tree. The fact that we work with context-free grammars as opposed to context-sensitive grammars enables us to provide simple extensibility. We simply add new symbols to the rules of a grammar to enrich its semantics.

To summarize, we emphasize the fundamental role detectors play in our theory. On the one hand, a variety of black-box detectors can be used to link semantic representations of the source objects into the hierarchical structured index. On the other hand white-box detectors query the same index and insert query results, meta-concepts, into the index.

In the terminology of databases this means that detectors mediate between objects outside the database and the index database. Again, this implies that they allow us to describe and store views of data items in a database. However, this power comes at a cost. As [CaM98] point out, there is always a trade-off between simplicity and expressive power of translation/mediation schemata. We believe that the efforts of writing specialized detectors and their benefits keep a good balance. Although feature grammars are a general framework, good implementations call for design guidelines; like easily parsable grammars (*e.g.* [GH67]) or safe and restricted grammars [SWK99] to avoid backtracking in the parser.

3 System architecture

The previous section introduced feature grammars as a means flexible enough for describing such diverse tasks as specifying the behavior of an indexing program and aiding a user in querying a database. In this section, we discuss the impact of this flexibility on the design of the actual implementation.

Feature grammars form the core technology of the Acoi platform[1] whose main goal is to build and maintain an extensible index to multimedia objects on the world wide web. The architectural layout is sketched in Figure 2. Its design aims at supporting the complete life cycle of an indexing program starting from the compilation of a feature grammar to endorsing and compiling a user's interactive and traditional queries on the multimedia index database. In the rest of this section we describe the core components in more detail.

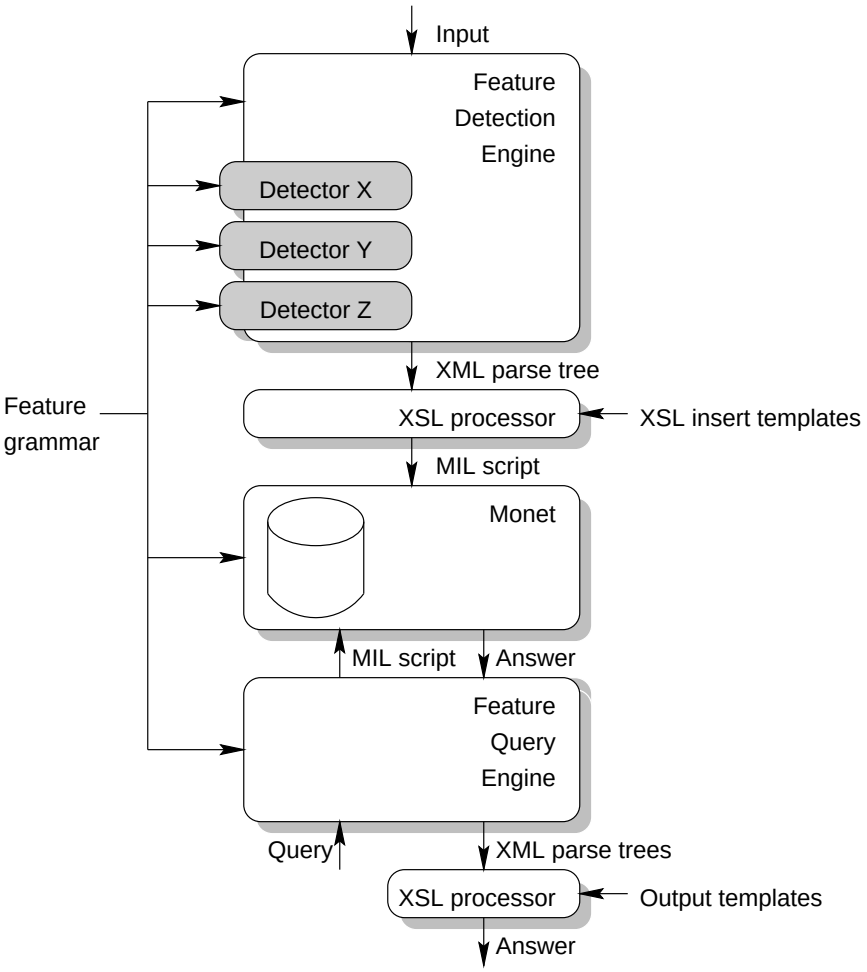

Figure 2: System architecture

As feature grammars are used throughout the system, the first step is to design a grammar which suits our needs. This design step closely resembles the one for lex and yacc applications [LMB92], but it is more intricate as detectors take over two roles: the 'lex-like' part specifies the atoms (*i.e.*, lexical units), whereas the 'yacc-like' part corresponds to the rest of the rules. As the lay-out of both parts impacts on the behavior of the feature engines, care is needed to make sure that parsing can go ahead efficiently. Some design guidelines are given in [SWK99].

The compiler-compiler combines the feature grammar and the detector code into a parser, the so-called Feature Detector Engine (FDE). When the grammar contains black-box detectors, their external implementation is linked into the code. As described in Section 2 these detectors modify the token stream (the input) of the parser, so their output should resolve against the grammar rules. If the compiler encounters a white-box or atom detector, the corresponding

[1]The Acoi (Amsterdam Catalogue Of Images) project [CWI99a] is funded through the Dutch Telematics Institutes project "Digital Media Warehouses" [CWI99c].

code is generated and linked into the parser.

The FDE can now be used to index data sources. A small walk-through of the FDE, generated for the example in Figure 1, illustrates the start of such a process. The token stream is initialized with a string containing the *url* of the web object we want to index, *e.g.* `http://www.cwi.nl/~acoi/`.

In the feature grammar the *web_object* rule is tagged as being the start of the parsing process. If the parser wants to resolve this rule it has to apply the non-terminals and terminals at its right-hand side, so it starts resolving the *url* rule. Because *url* is an atom detector the regular expression is used to validate the first token in the stream, our initial string. Because this token is valid it is removed from the stream, labeled as an *url*, and put in the parse tree the parser is constructing. The parser then proceeds to the *web_header* rule and, as it is also a detector, calls the program provided by the developer. This program sends a HTTP `HEAD` request to the server identified by the *url* and filters out the information needed, the content type of the object. The detector puts this information, labeled as *content_type*, as a token into the stream. The parser then validates the contents of the stream against the right-hand side of the *web_header* rule and, on success, proceeds. This process continues until the start rule is considered valid, because its right-hand side is valid or, in case of alternatives, one of its right-hand sides is valid. At this moment the parser will have completed the construction of the parse tree. This tree will have the start symbol as root, as displayed in Figure 3, and its nodes and leafs contain the features and concepts computed in the parsing process.

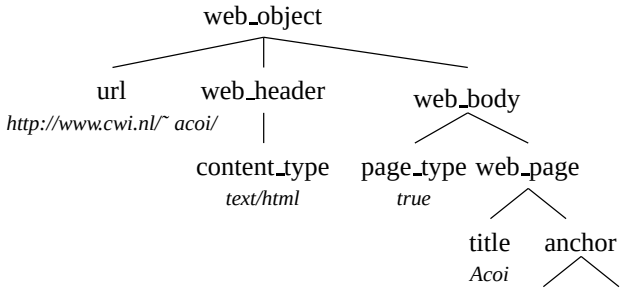

Figure 3: Parse Tree

Because the data the parse tree contains can be regarded as semi-structured, it is only natural to use the eXtensible Markup Language (XML) documents [W3C98] represent, export and exchange data between the different components of the system. For example, the FDE outputs a XML document which contains all index values; by using eXtensible Stylesheet Language Transformations (XSLT) templates [W3C99] this document can be readily translated into a Monet Interpreter Language (MIL) script. This script contains algebraic commands to insert the parse tree into our state-of-the-art database management system, Monet [BK99].

On the database side, we need to convert the parse trees so that they fit a certain relational database schema. To map the parse tree into database tables there are two fundamental choices to make: we may either store it in a node-based or in an edge-based manner. If we opt for the node-based schema, we could use just a single relation to store the structure of the tree or distribute it over multiple relations. In the first case, the database schema is very simple: we define a relation `parent_child` to record all parent-child OID pairs that occur in the parse tree, and use the OIDs as foreign keys to store all other attributes in dedicated relations. However, the price for the simple schema is to be paid at query time. Either we need to do expensive pointer chasing on a single large table or, as an alternative, as many self-joins as our search depth. In terms of efficiency, both alternatives are not very promising.

If we distribute the tree over multiple relations, we have the opportunity to encode additional information in the relation names. As feature grammars induce types (namely the left-hand side

```
%atom       int width, height, depth, color, id;

%detector   image_type
             select true
             from web_object
             where content_type = "image/gif";
%detector   web_image(url);
%detector   icon(url);
%detector   banner
             select true
             from web_image
             where (width / height) < 0.33
             and (width / height) > 3;

web_body  →  image_type web_image;
web_image →  width height depth histogram;
histogram →  color*;
web_image →  icon | banner;
icon      →  id;
```

Figure 4: Image extension of the feature grammar

of the corresponding rules) on the parse tree nodes, we can store all nodes of a certain type in one relation without having to record their type explicitly. This keeps relations small and, importantly, allows us to make use of the set-oriented operators of the relational algebra at query time and, therefore, avoids the need to chase individual pointers. However, to keep track of which relations belong to our tree, we need to record their names in a meta-table we call a *dataguide* [GW97].

But we can still do better. If we store edges instead of nodes, we can expect to keep relations even smaller, more compact and semantically unique. We benefit from the fact that edge types are uniquely determined by the string concatenation of types of the two nodes they connect. This edge-based approach fits nicely into the binary model adhered to in Monet. Relationships between non-terminal symbols are stored in $\{parent\ oid, child\ oid\}$ pairs, while relationships between non-terminal and terminal symbols will lead to the storage of $\{parent\ oid, child\ value\}$ pairs.

When the meta-index database has been constructed and filled with parse trees, the user can interactively query it by using the Feature Query Engine (FQE). The query language has a SQL flavour, which reduces the learning curve and opens the system for integration with other applications. The FQE uses the feature grammar to expand wildcards in path expressions, which are used to reason about the hierarchy, and to translate the query into a query execution plan, which consists of binary relational algebra commands in MIL.

4 WWW Multimedia Indexing

In the previous sections of this paper indexing multimedia objects on the World Wide Web is used as a running example. In this section we will describe our approach in some more detail.

Currently we traverse the Internet in a robot-like fashion to gather index values for (multimedia) objects of interest. An extended version of the feature grammar shown in Figure 1 is used to describe the network that is formed by the World Wide Web and is traversed by the web robot. This feature grammar describes on one hand the technical information needed by the robot and the feature values used to index HTML pages. And on the other hand the execution sequence of the robot, which main part is basically the parser or FDE. The HTML pages are parsed by

one of the detectors, *i.e. web_page*, which also evaluates the hyperlink-structure inside the page. The robot uses this information to find more candidate objects to continue his traversal of the World Wide Web.

The parse tree database, constructed on basis of this feature grammar and its FDE, enables the user to query the index on keywords to retrieve web pages. This database functions in the same way as the indexes used by the classical search engines. Additional detectors can already enhance this index with more feature values, *e.g.* the language of the page determined by a language classifier.

However, to enable content- and concept-based retrieval for other multimedia types, the grammar has to be further extended. This extension consists of detectors, and their corresponding rules, which interpret and compute useful features and concepts of the source multimedia objects.

In Figure 4 the extension of the grammar for images is shown. The *web_image* detector returns some simple image features like *width* and *height* and a color *histogram*. Color histograms are used in many content-based retrieval techniques to determine the similarity of an image to a given image. The second *web_image* describes alternative views on the same image. The *icon* detector creates a thumbnail used to represent the image in a query answer (the *id* identifies the thumbnail in a directory structure). The *banner* rule shows the use of a white-box detector to determine a meta-concept: it annotates an image as a banner on the basis of its aspect ratio. This second *web_image* rule shows the use of conjunctive alternatives: for an image an icon representation can be created, but at the same time it is also a banner.

Queries can now combine the index information of two or more multimedia objects linked on the World Wide Web. Figure 5 shows a simple query which combines keywords from a HTML page and the banner concept for an image. The query searches for non-banner images embedded in a web page containing the keyword “soccer”.

```
select   i.url, i.*.icon
from     anchor a, web_object i
and      a.surround.*.word = “soccer”
and      a.web_object = i
and      i.web_body.web_image
and      not i.web_body.web_image.banner;
```

Figure 5: Example of a query

The same kind of grammar extensions are developed for other multimedia types, like audio and video. Detectors are constructed by partners in other subprojects of the Dutch national projects: “Digital Media Warehouses” [CWI99c] and “Advanced Multimedia Indexing and Searching” [CWI99b].

5 Related Research

The problem we address is closely related to information integration (*e.g.*, see [RS97, PGMU96, GKD97, CaMH+94]). However, there is a crucial difference. Normally mediators provide a fixed set of predicates. In our case, the mediator provides just a collection of semi-structured parse trees whose schema is given by a feature grammar.

Thus, on the one hand we can consider feature grammars a variant of a structuring schema of the parse trees very much like *a priori approximate data guides* [GW99]. This characteristic makes them useful for outlining/visualizing the information stored in a feature database. On the other hand we can draw benefit from their grammatical structure and use a recognizer to provide an execution mechanism for indexing actual external objects.

So our concept unites two previously distinct worlds: the schema exported by a mediator and the structural capability description of the source.

6 Conclusions and Future Work

We have described an architecture for indexing multimedia objects on the World Wide Web. The feature grammars which play a fundamental role in this architecture provide a uniform framework for the ontology as well as the execution of the multimedia index agent. Furthermore, they allow seamless integration of third-party plug-in modules into the indexing program, called detectors. This, together with their semi-structured capabilities, gives feature grammars the required flexibility for constructing and maintaining an useful meta-index for multimedia objects on the Internet.

The ideas presented are implemented in the Acoi project. The current status of the project is that a compiler-compiler, a query processor, as well as some other tools, have been implemented. A feature grammar for indexing multimedia data together with its detectors is designed and implemented. This system is operational and has been indexing large amounts of web pages, images, and MIDI files.

Future research will focus on how to 'roll forward' the parsing process to incorporate new detectors and new rules and thus also new index values into an existing index. This functionality calls for techniques to invalidate or revalidate older index values.

Another focus is probabilistic querying. Many multimedia detectors not only return the feature value, but also a probability value. This probability value can be used to assess the quality of the query answer and as a basis for ranking expressions.

References

[AGJ97] S. Santini A. Gupta and R. Jain. In search of information in visual media. *Communications of the ACM*, 40(12):35–42, December 1997.

[AR97] Robert B. Allen and Edie Rasmussen, editors. *Proceedings of the 2nd ACM International Conference on Digital Libraries*, Philadelphia, PA, USA, 1997.

[BK99] P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World. *The VLDB Journal*, 1999. To appear.

[CaM98] Chen-Chuan Chang and Héctor García Molina. Conjunctive constraint mapping. In *ACM Digital Libraries 1998*, pages 49–58, 1998.

[CaMH+94] Sudarshan S. Chawathe, Héctor García Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The tsimmis project: Integration of heterogeneous information sources. In *IPSJ*, pages 7–18, 1994.

[CWI99a] CWI: Centre for Mathematics and Computer Science. *Acoi*. `http://www.cwi.nl/~acoi/`, 1999.

[CWI99b] CWI: Centre for Mathematics and Computer Science. *AMIS*. `http://www.cwi.nl/~acoi/AMIS/`, 1999.

[CWI99c] CWI: Centre for Mathematics and Computer Science. *DMW*. `http://www.cwi.nl/~acoi/DMW/`, 1999.

[FLM98] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world wide web: A survey. *ACM SIGMOD Record*, 27(3):59–74, 1998.

[GH67] S. Ginsburg and M. A. Harrison. Bracketed context-free grammars. *Journal of Computer and System Sciences*, 1(1):1–23, 1967.

[GKD97] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In *Proceedings of 1997 ACM SIGMOD Conference*, 1997.

[GW97] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, 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*, 1999.

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

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

[LMB92] John R. Levinne, Tony Mason, and Doug Brown. *lex & yacc*. O'Reilly & Associates, Inc., 1992.

[OS95] V.E. Ogle and M. Stonebraker. Chabot: Retrieval from a relational database of images. *IEEE Computer*, pages 40–48, September 1995.

[PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 132–141. IEEE Computer Society, 1996.

[RS97] M. Roth and P. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data. In *VLDB*, pages 266–275, 1997.

[SWK99] A. Schmidt, M. Windhouwer, and M. L. Kersten. Feature grammars. In *ISAS'99 The 5th. Int'l Conference on Information Systems Analysis and Synthesis*, Orlando, Florida, 1999.

[W3C98] W3C: World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*. `http://www.w3.org/TR/REC-xml`, February 1998.

[W3C99] W3C: World Wide Web Consortium. *XSL Transformations (XSLT) Specification 1.0, Working Draft*. `http://www.w3.org/TR/WD-xslt`, October 1999.

[WAS98] Ian Witten, Rob Akrcyn, and Frank M. Shipma, editors. *Proceedings of the 3rd ACM International Conference on Digital Libraries*, Pittsburgh, PA, USA, 1998.

[xFr99] xFront.org. *xFront*. `http://www.xfront.org/`, 1999.