A Feature Database for Multimedia Objects

M.L. Kersten
CWI
{Martin.Kersten@cwi.nl}

M.A. Windhouwer
CWI
{Menzo.Windhouwer@cwi.nl}

N.J. Nes
University of Amsterdam
{niels@wins.uva.nl}

Abstract

The Acoi project provides a large-scale experimentation platform to facilitate studies in the area of indexing multimedia objects and their subsequent retrieval. The index model is based on assembling the results of feature detection algorithms into hierarchical structures to classify the objects. This paper provides an overview of the Acoi architecture and the Feature Detector Engine (FDE) model. Central to its design is a grammatical description of the feature relations to classify the multimedia objects and to steer their detection and storage. Its role is informally introduced.

1 Introduction

Searching relevant objects in a large scale multimedia database on the basis of selective properties and realistic similarity measures has become one of the key challenges in computer science. It has gained this position in recent years, due to the abundance of material accessible through the World Wide Web, the progress in digital imaging and audio processing.

The solutions available are largely based on catalogues, which are either constructed manually, as in the case of Yahoo (http://ipix.yahoo.com), or derived from the text surrounding the objects of interest, see AltaVista (http://www.altavista.com). A significant step towards disclosing the information contained in multimedia objects can be found at the homepage of Magnifi (http://www.magnifi.com). It provides an integrated solution to index a multimedia database using a combination of text and image feature indexing techniques.

Not withstanding the success of catalogue systems to support the community at large in finding information, progress in the area of automatic indexing multimedia objects has been rather limited [JAIN97]. The challenge remaining is to determine those features that adequately capture a concept — invariant to the recording method — and that aid subsequent retrieval. The issues to be dealt with are: ”What portion of a photograph should be used as a thumbnail?”, ”What fragment of an audio stream acts as index entry?”, ”How to support visual browsing in a large collection of images?”, ”How to classify two images in a concept hierarchy?”,...

The Acoi approach

The Acoi[1] project[2] has been initiated to develop a large-scale experimentation platform to facilitate studies in this area. The prototype has been set-up to accommodate $>$1M images, $>$ 50K audio files, and $>$10K video streams. The projects main contribution and innovation is a sound model and effective system architecture to accommodate a variety of algorithms dealing with extracting properties from multimedia objects for indexing purposes. This paper reports on a part of the system design, which has been driven by the following requirements:

- It should provide access to globally stored multimedia objects.
- It should be dynamically extensible with indexing tools.
- It should support both black & white box feature detection algorithms.
- It should accommodate a broad spectrum of classification schemes.

[1] Amsterdam Catalogue of Images

[2] The Acoi project is funded through the Dutch SION project ”Amis” and Telematics Institutes project ”Digital Media Warehouses”.

- It should provide proximity and partial query answering schemes.

Their rationale can be summarized as follows.

Given the sheer size of the multimedia database, it is out of the question to retain a substantial fraction at the experimentation site. Rather, the Acoi system is the intermediary for a group of researchers to experiment with information stored at remote locations.

Dynamic extensibility is needed to support experimentation by specialists in the domains considered. The system permits a new feature detection algorithm to be linked into the system at runtime. It causes the system to quickly deploy it against easily accessible objects, i.e. those already in the cache.

Black box feature detectors are (proprietary) software tools with only a loose interaction with the Acoi system. They are generally obtained as binary executables only and limited knowledge on their internal behavior can be used to steer the indexing process or improve query responsiveness. Contrary, white-box feature detectors are described as mathematical expressions over easily derivable (or pre-calculated) object features. Often they constitute query views over the multimedia database itself. They lend themselves for optimization under control of an optimizer.

To improve cross fertilization of feature detection schemes, the system should support coexistence of both manual, semi-automatic, and automatic classification schemes. Manual classification schemes alone does not work, because it does not scale. It should be used in those cases where (semi-)automatica classification has already reduced the set to a few hundred elements.

Querying a multimedia database stresses the traditional computational model in a DBMS, because its indices are never complete and it is impossible to wait for the indexing process to finish. The prime reason being that most of the information sources are stored remotely and it is too expensive to access them repeatedly. Furthermore, the decision model behind a query expression — the predicate holds or not — should be relaxed to retain answers based on probabilistic expressions.

The Acoi project addresses these issues with a flexible architecture and sizable demonstrator.

Related research

Multimedia (database) indexing issues are studied at various places [ARYA96, GUPT97, SPIE95]. In the area of image analysis Photobook[PENT94], WebSEEk[CHAN97] and QBIC[FLIC95] illustrate that within a limited domain and relatively small databases it is possible to retrieve similar objects using easy computable image properties, such as color histograms.

A major research force has been triggered by the US Digital Library Initiative. The Initiative's focus is to dramatically advance the means to collect, store, and organize information in digital forms, and make it available for searching, retrieval, and processing via communication networks – all in user-friendly ways. For example, the Berkeley Digital Library project aims to develop technologies for intelligent access to massive, distributed collections of photographs, satellite images, maps, full text documents, and ”multivalent” documents. It involves researchers of the Computer Science Division, the School of Information Management & Systems, and the Research Program in Environmental Planning & Geographic Information Systems, as well as participation from government agencies and industrial partners. Stanford University participates in this program with a focus on inter-operation mechanisms among heterogeneous services. Carnegie Mellon University deals with content-based retrieval of video. Progress on various aspects of the Digital Libraries are published in [DL96, DL97] and http://dli.grainger.uiuc.edu/national.htm.

The remainder of this report is organized as follows. In Section 2 we introduce a model to support a broad spectrum of incremental multimedia indexing. Section 3 places this model inside a system architecture. We conclude with the current status of the Acoi project and an indication of the challenges ahead.

2 Acoi Detector Model

In this section we present a motivational example, followed by an informal definition of the Acoi data- and execution model.

2.1 Motivational example

The Acoi detector model has been developed to provide a concise description of the structure and organization of a multimedia index database. The model is based on the observation that indexing an arbitrary multimedia object is intuitively equivalent to deriving a grammatical structure that provides a name space to reason about and to access its components.

A small example will make this clear. Consider your favorite photograph referenced by a label on a photo disc and the task to index it from your multimedia album. Then the following steps are likely to occur. First, you enter the photo id# as an element in a classification tree under the header `album.graphics` - it is a graphics object (syntax)- and under the header `album.photo` - it also belongs to your photo album (semantics). Second, you attach the labels `album.photo.creation-date`, `album.photo.location`, `album.photo.caption` and classification properties - it also belongs to the class of family photos. Finally, you might pan portions of the photo to update the portrait gallery `album.people`. These actions lead to a hierarchical structure of the components associated with a single multimedia object, as illustrated in Figure 1.

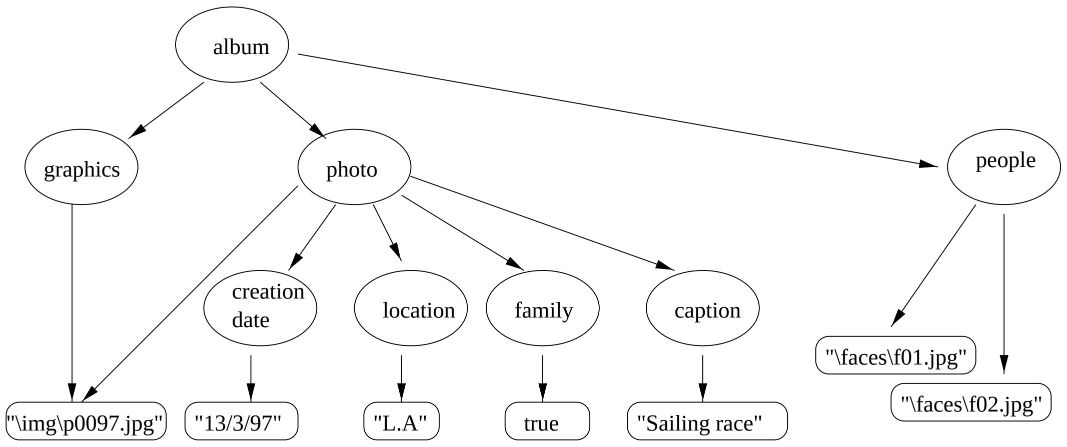

Figure 1: Complex MM object structure

You can also interpret this process as an extended parsing activity. Given the `photo`, `date`,`location`, `caption`, and `set of faces`, the structure derived forms a parse tree for an element accepted by the grammar for your album. The bottom of the parse tree contains lexical atoms belonging to a particular base domain: graphicsType, photoType, dateType, $\cdots$. Unlike traditional parsing there is no a priori fixed sequence from which the lexicals are consumed. The "parser" works its way through lexicals created dynamically, e.g. questions are asked and faces are derived in the process.

2.2 The Feature Grammar

The ultimate goal of the Acoi project is to enable (semi-)automatic indexing a large multimedia database. The starting point for this process is a collection of multimedia objects M, whose basic properties are already obtained with a Web robot and stored in a Monet[BONC95, BONC98] database using the schema in Figure 2. Further indexing this database amounts to addition of binary relations which carry values to classify the collection M in new dimensions.

For this process we borrow concepts and techniques from formal language theory. To recall, we describe a language of properties using a grammar $G = (V, T, P, S)$ where V is a collection of variables, T a set of terminals, P productions of the form $V \rightarrow (V \cup T)^*$, and S the start symbol taken from V. A sentential form α is a string of terminals and variables, such that $S \overset{*}{\rightarrow} \alpha$. The collection of parse trees is denoted by PT.

A sublanguage $L(G_w)$ is described with the sub-grammar $G_w = (V_w, T_w, P_w, w)$, taking a consistent subset of the corresponding components of G. It describes the structure of sub-sentences in the language $L(G)$.

The terminals T are ordinary typed lexicals. The built-in set of types encompasses the traditional programming types `int` $\cdots$ `str`. Furthermore, type extensibility of Monet provides for more complex types, such as `image`. The terminals are collected into token sequences or sentences $TS = [t_0(v_0), \cdots t_k(v_k)]$ where $t_i \in T$ is an atomary type name, and v_i a value in $domain(t_i)$. A token sequence ts belongs the language $L(G)$, i.e. ts is parsed against grammar G, if there exists a sequence of productions such that $S \overset{*}{\rightarrow} ts$.

Turning back to our main objective, we consider a feature database a collection of sentences with indexing values. Their parse tree denotes a hierarchical structure and provides a name space to access and manipulate components. Actually, there exists a natural mapping from sententials to complex objects. In particular, the (non-)terminals are mapped into object attributes; repetition into a list constructor; and alternatives as elements in

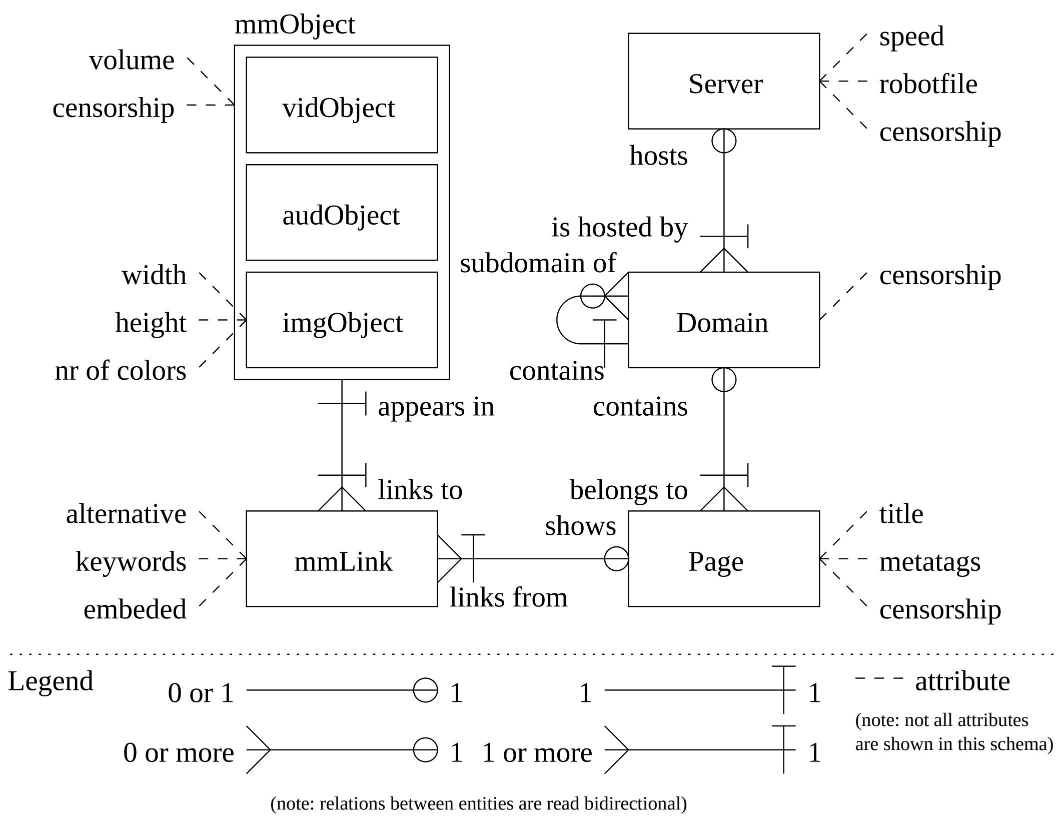

Figure 2: Acoi Database Core Schema

abstract classes. A prime advantage of our grammatical approach is its conciseness in specifying a large collection of classes. When a class description is needed for application interfacing, it can readily be derived and refined with application specific behavior.

Definition 1. For $v \in V \cup T$ the class C_v denotes the class of complex objects equivalent to the sub-language G_v.

Feature detectors fit in this framework as operations associated with non-terminals, which massage a token sequence to steer correct parsing of the corresponsing sublanguage. For this they may inspect the parse tree under construction (its sentential form).

Definition 2. A *feature detector* $d \in D \subset V$ is a function that maps a token sequence $w \in TS$ into $w' \in TS$ using its parse tree d_t, such that the head of w' is a sentence in the sublanguage G_d.

A feature detector may involve user interaction to identify the element in F or even extend F in the classification process. For example, the detector could ask the user explicitly for the classification information using a dialogue initialized with a set of choices *ask(”car”,”house”,...)* or to let the user draw geometric structures on the screen to identify the portions of interest, e.g. *faces*.

Ordinary functions differ from the feature detectors in that the information derived is not kept permanently in the database for recall. As such, they are also total functions instead of partial functions (over the database extent).

Since detectors may be introduced long after the database has been created, the Acoi indexing process necessarily is incremental, because the source may not be available at all times. This leads to two sub-classes for any class C as follows:

Definition 3. The object class indexed by feature detector d is denoted by $\vec{C}_d$. Those not yet indexed are denoted by $\overset{\rightharpoonup}{C}_d$. At any time class $C_d = \vec{C}_d \cup \overset{\rightharpoonup}{C}_d$.

2.3 An example

To illustrate, consider the feature grammar defined in Figure 3. The top part defines atoms (typed terminals) and feature detectors. Detector `avatar` is a white-box detector; its behavior is defined by an expression understood by the Acoi system. The other detectors are black box detectors, known by their name only. It is up to the user to supply an implementation. The body may inspect parse tree - it provides access to contextual information- and

```
# Atoms
%ATOM           image;
%ATOM           str protocol server, directory;
%ATOM           str basename, extension
%ATOM           int width, height;

# Detectors
%DETECTOR       url;
%DETECTOR       picture;
%DETECTOR       icon(image);
%DETECTOR       avatar(thumbnail) ? thumbnail.picture.width=40
                && thumbnail.picture.height=60;

# Production rules
mmo:            url category;
url:            protocol server directory* basename extension;
category:       thumbnail | avatar;
thumbnail:      picture icon;
picture:        image width height;
icon:           picture;
```

Figure 3: A Feature Grammar Example

change the token sequence to assure proper continued parsing.

The bottom part contains a grammar for a hierarchical structured feature space. An object *o* that is known to obey this grammar has an implied syntax tree where the edges are labeled with the names of the corresponding production rules. Components of this parse tree can be accessed with regular (path) expressions.

Unlike traditional grammars, alternation between `thumbnail` and `avatar` is not exclusive. Both productions describe alternate views on the same underlying object. The category rule succeeds when for all alternatives that succeed produce the same token sequence for continuation. An alternative that fails is further ignored.

Observe that semi-structured databases follow the same pattern, a document is a hierarchical composition whose structure is conveniently described by a grammar (e.g. SGML, HTML, XML, Hytime). However, in Acoi we expect an a priori geven grammar and do not derive the schema on the fly from the documents in the database.

2.4 Execution model

An informal description of how the feature grammar is used to obtain the index runs as follows (using the example feature grammar in Figure 3). At some point in time, a string (e.g. ”http://www.cwi.nl/∼ monet/lady.gif”) is inserted in the token pool from which the grammatical structure is parsed. The start node `mmo` creates a parsing context that ultimately leads to acceptance or rejection of the object as a `mmo` object. This proof is attempted by proving the right hand side of the `mmo` rule, which starts with calling the `url` detector. It searches the pool for a string and breaks it into components as follows: `[protocol(http), server(www.cwi.nl), directory(∼monet), basename(lady), extension(gif)]` and the detector returns SUCCEED. The modified token pool can be consumed by the parser looking for a valid url. The `mmo` rule can then proceed with the `category` proof with two alternatives, `thumbnail` and `avatar`, both are valid continuations.

The `thumbnail` rule triggers the detector `picture`. Its body has access to the complete parse tree built so far. It uses this information to access the file being referenced and determine its type from the extension component. Upon success (it is a `gif` file) it opens the corresponding file and generates atoms `[image(cache/lady.gif), width(85), height(250)]` pushed in front of the token queue.

Subsequently the `icon` detector is called with the most recent `image` object as parameter. It derives a small icon, leaving it behind in the token stream for consumption as `[image(cache/lady.icon.gif), width(75), height(75)]`. When `thumbnail` proof has ended successfully, the category proof proceeds with the next alternative, `avatar`.

The `avatar` is an example of a predicate-based detector. The `thumbnail` argument sets the context. But there are two pictures available in the parse tree (thumbnail and icon). Therefore, the path should explicate the context to locate the correct `width` and `height`.

The `category` rule succeeds if at least `thumbnail` or `avatar` reports success. When the complete `mmo` rule has been proven the original string object has been parsed into a hierarchical structure containing classification and feature information.

This execution model gives a systematic parsing method to classify a new object. The feature detector engine uses this method to steer feature detector behavior. Basically classification is based upon the success or failure of parsing the token sequence. The detectors merely assure that the proper classification information is available just in time.

3 Architecture Overview

An overview of the Acoi architecture is shown in Figure 4. This section offers a short description of the role and approach taken in each component.

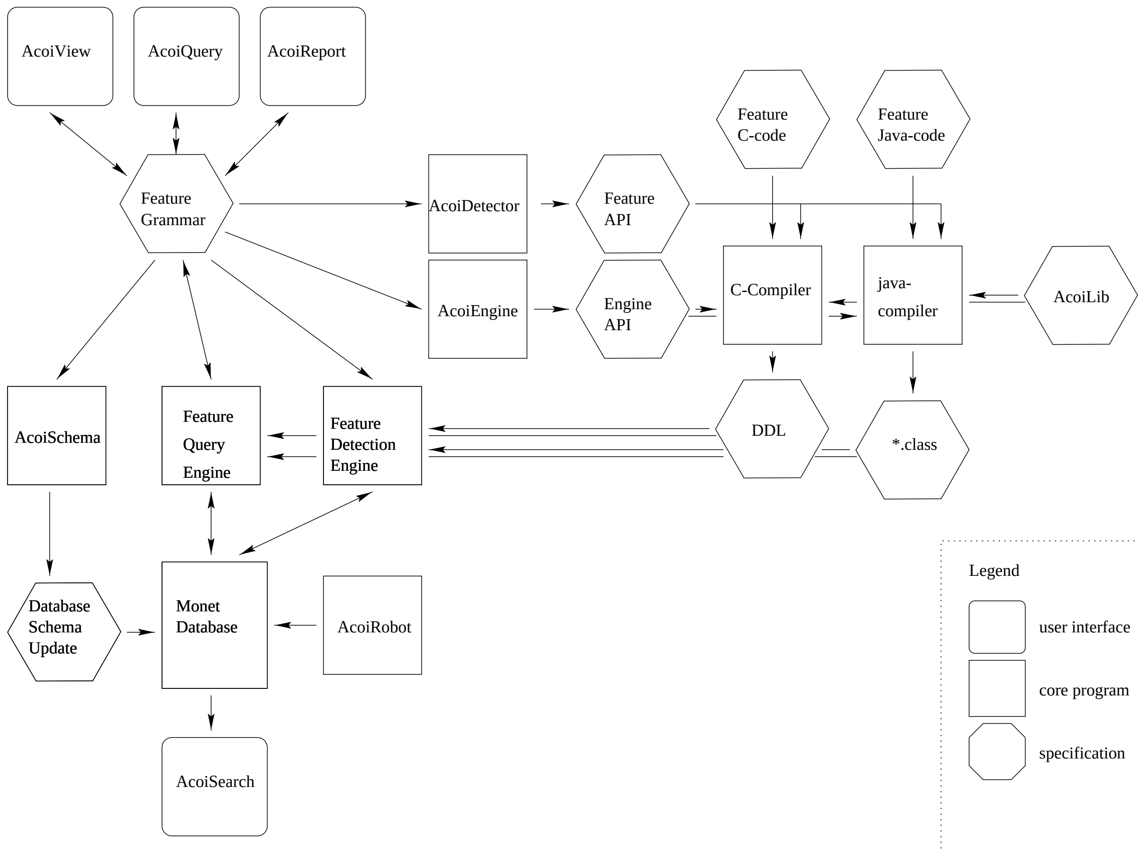

Figure 4: Acoi Architectural Overview

3.1 Web Robot

The first component designed and implemented is the AcoiRobot. The AcoiRobot traverses the World Wide Web in search of multimedia objects. Some basic object features are computed and surrounded HTML text is used to derive index terms. These basic features and index terms are used to provide an initial parse tree for further feature detection. The database scheme for this information is shown in Figure 2.

Related to the AcoiRobot is the AcoiSearch interface which can be used to retrieve multimedia objects from the database using only this basic information. AcoiSearch provides an AltaVista like service, e.g. retrieval based on keywords.

3.2 User Interfaces

The primary interfaces to the Acoi system consists of Java applets. AcoiView is an applet to inspect and modify a feature grammar. It contains administrative functions to manage the underlying database and provide simple query primitives to inspect resource consumption.

AcoiQuery is a Java applet to formulate a feature grammar query. It consists of two parts: the target- and restriction-grammar. The former is a subset of the feature grammar augmented with expressions to derive the information before being displayed. The restriction-grammar is a subset of the database schema, augmented with predicates to derive the collection of object considered for retrieval.

AcoiReport is a Java applet to browse the results of a query. It uses a straightforward mapping from target grammar to HTML.

3.3 Database Management

The core compilers and utilities are ordinary Unix programs. AcoiSchema takes a feature grammar and derives the underlying database structure. For Monet this involves generation of a collection of binary relations to manage the database. This mapping retains the names from the grammar to ease access to the database using non-Acoi tool sets, such as direct querying with the Monet MIL processor, or a SQL interpreter.

AcoiSchema also takes care of adapting an existing database. In that case the program takes the old and new grammar to produce updates statements for the database. This include invalidating the classifications and feature values inconsistent with the new grammar.

3.4 Detector Libraries

The application interfaces for both C and Java detector implementations are derived from the feature grammar using AcoiDetector, AcoiEngine, AcoiLib.

AcoiDetector generates a skeleton file for all detectors in a feature grammar. It should be extended by the user with bodies to derive the feature values of interest and to make them known to the token pool for the parser. The detectors should be designed as memoryless devices; there is no causal order between successive calls. Moreover, the code should be thread-safe, otherwise it will be run in a separate process and communication is set up using a plain ASCII stream.

AcoiEngine generates a parser from the feature grammar. Linked with the AcoiLib library and detector bodies, it produces a self-contained feature detector engine. It can be called from the command line. Its output is a sequence of updates on the Monet database.

AcoiLib constitutes a collection of binaries to ease development of detectors. It provides functions to access the parse trees, to modify the token pool, and general utilities for debugging.

3.5 The Feature Detector Engine

The Feature Detector Engine (FDE) is a multi-threaded process derived from the feature grammar and in charge of updating the feature database within the resource limitations given. It provides a harnass for multiple AcoiEngine parsers to update the database in parallel.

The FDE responds to insertions and modifications of the data by AcoiRobot, because such updates may lead to starting a (partial) recomputation of feature values and reclassification of multimedia objects. Furthermore, FDE provides management functions to (de-)register detectors and to keep track of performance.

FDE maintains a small cache of multi-media objects, such that experimentation with new detectors can be accommodated without excessive load on the network. Once the algorithms have proved stability, the scope of applicability of a feature can be extended to cover larger domains.

3.6 The Feature Query Engine

The Feature Query Engine (FQE) provides a harnass for multiple query streams, described by target- and restriction-grammars. A complicating factor is that the complete answer set can’t be determined in finite time for several reasons. First, the database is constantly updated by AcoiRobot with new information about multi-media objects on the World Wide Web. Likewise, the FDE constantly updates the database with indexing information obtained from the feature detectors. Finally - more challenging - the raw information is not likely to be available to aim indexing and retrieval. Downloading it for feature detection should be scheduled such that maximal information is extracted each time it is accessed.

These factors require FQE to be able to provide partial query answers and to provide quality figures for answers returned. An assessment (and prediction) of the cost to access another set of objects is also highly relevant for the user, before he calls upon downloading all images over the net for a simple detector experiment.

4 Implementation status

The AcoiRobot has been implemented. It obeys the robot exclusion protocol and has collected so far a candidate list of 200K image urls. As soon as the hardware platform has been installed, the candidate list is explored. This base database will be made available for public use as soon as possible using the AcoiSearch demonstrator, a kind of Alta Vista applications.

Prototype AcoiSchema, AcoiEngine, and AcoiDetector programs have been constructed. Demonstrator programs are currently being developed along with the necessary documentation. The interaction with the database is loose. Updates are cast into ASCII files of Monet commands. Moreover, no access is provided to partial parse tree already stored in the database. The envisioned harnass FDE with tight database coupling will be developed after our first experimentation with the envisioned architecture has produced more insights in its requirements.

The AcoiView applet has been developed in Java 1.1 using the Swing library. The applet helps the user to define a correct and complete feature grammar.

5 Summary and conclusions

We have introduced a novel method to index a large multi-media database with user-defined features. It is based on sound parsing techniques with a keen eye towards support for incremental parsing. Actually, the database is considered a large collection of parse trees, readily available for querying as a database of complex-structured objects.

A large experimentation platform is currently under construction to validate the approach taken. It is scheduled for access in the summer of 1998.

The prime areas of research are the formulation of the query language and the two harnass programs, FDE and FQE, to handle the intrigate issues raised by accessing a highly volatile and distributed store.

References

[ARYA96] M. Arya, W. Cody, C. Faloutsos, J. Richardson and A. Toga. The QBISM Medical Image Database. In *Multimedia Database Systems* V.S. Subrahmanian, S. Jajodia (eds), 1996, pp. 79-97.

[BONC95] P.A. Boncz and M.L. Kersten. Monet: An Impressionist Sketch of an Advanced Database System. In *Proc. IEEE BIWIT workshop, San Sebastian (Spain)*, July 1995

[BONC98] P.A. Boncz, A.N. Wilschut and M.L. Kersten. Flatting an Object Algebra to Provide Performance. In *Proc. 14th International Conference on Data Engineering, Orlando, Florida, USA*, February 1998

[CHAN97] S. Chang, J.R. Smith, M. Beigi and A. Benitez. *Visual Information Retrieval from Large Distributed Online Repositories*. Communications of the ACM, December 1997, pp. 63-71.

[DL96] Proceedings of the 1st ACM International Conference on Digital Libraries. Bethesda, Maryland, March 1996.

[DL97] Proceedings of the 2nd ACM International Conference on Digital Libraries. Philadelphia, PA, USA, July 1997.

[FLIC95] M. Flickner et al. *Query by Image and Video Content: the QBIC system*. IEEE Computer, 28(9), 1995, pp. 23-32.

[GUPT97] A. Gupta, S. Santini and R. Jain. *In Search of Information in Visual Media*. Communications of the ACM, December 1997, pp. 35-42.

[JAIN97] R. Jain. *Visual Information Management*. Communications of the ACM, December 1997, pp. 31-32.

[PENT94] A. Pentland, R.W. Picard and S. Sclaroff. *Photobook: Tools for Content-based Manipulation of Image Databases*. In Proceedings of Storage and Retrieval for Image and Video Databases II, 2, 185, SPIE, Bellingham, Wash. pp. 34-47, 1994

[SPIE95] Proc. Storage and Retrieval for Image and Video Databases I, II, and III, Vol. 1,908; 2.185; and 2,420; W. Niblack and R. Jain, (eds.), SPIE, Bellingham, 1993, 1994 and 1995.