Combining Concept- with Content-based Multimedia Retrieval

Menzo Windhouwer[1] and Roelof van Zwol[2]

[1] CWI, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
`Menzo.Windhouwer@cwi.nl`

[2] Utrecht University, PO Box 80.089, NL-3508 TB Utrecht, The Netherlands
`roelof@cs.uu.nl`

1 Introduction

The Internet forms todays largest source of Information, with public services like libraries and museums digitizing their collections and making (parts of) it available to the public. Likewise, the public digitizes private information, *e.g.* holiday pictures and movies, and shares it on the World Wide Web (WWW). This kind of document collections have often two aspects in common. They contain a high density of multimedia objects and its content is often semantically related. The identification of relevant media objects in such a vast collection poses a major problem that is studied in the area of *multimedia information retrieval.*

A generic multimedia information retrieval system is sketched in Fig. 1 (based on [1]). The left-hand side depicts the interactive process, where the user formulates her query, using relevance feedback. On the right-hand side of the figure a librarian is annotating the document collection. In the earlier stages of information retrieval, the annotation was done in a complete manual fashion. Today, this process is often supported by automatic annotation, which is also referred to as content-based retrieval.

The rise of XML as *the* information exchange format has also its impact on multimedia information retrieval systems. The semistructured nature of XML allows to integrate several more-or-less structured forms of multimedia annotations in the system, *i.e.* the database stores a collection of (integrated) XML documents. However, using the XML document structure directly for searching through the document collection is likely to lead to semantic misinterpretations. For example, an XML document contains the structure *company* → *director* → *name*. When searching for the name of the company, without any knowledge of the semantics of the implied structure the name of the director can, mistakenly, be found.

With the focus on document collections where the content is semantically related, it becomes feasible to use a conceptual schema that describes the

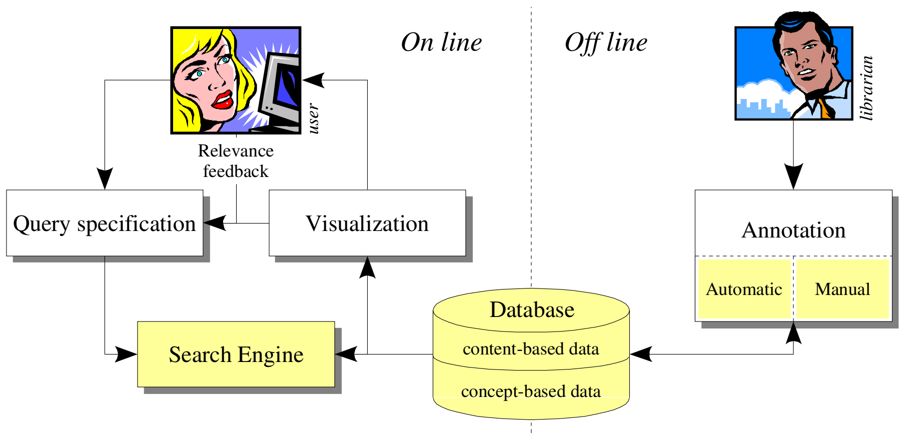


Fig. 1. Multimedia information retrieval system.

content of the document collection at a semantical level of abstraction. This approach, defined in the Webspace Method [2] allows the user to formulate complex conceptual queries that exceed the ‘boundary’ of a document. However, using a conceptual search alone is not satisfactory. The integration with a content-based technique, such as a feature grammar is essential.

A feature grammar is based on the formal model of a feature grammar system and supports the use of annotation extraction algorithms to automatically extract content-based information from multimedia objects [3].

This chapter describes the integration of the Webspace Method and feature grammars to realize a retrieval system for multimedia XML document collections. To illustrate the usability of the combined system fragments of the Australian Open case-study are used. This case study has been described in more detail in [4] and [5].

2 Automatic Annotation Extraction

The holy grail for *automatic annotation* is to extract all explicit and implicit semantic meanings of a multimedia object, *i.e.* take over a large part of the manual annotation burden. This ultimate goal may never be reached, but for limited domains knowledge can be captured well enough to automatically extract meaningful annotations. To realize this, the *semantic gap* between raw sensor data and “real world” concepts has to be bridged. The predominant approach to bridge this gap is the translation of the raw data into low-level features, which are subsequently mapped into high-level concepts.

These steps are illustrated in Fig. 2, which shows the annotation of an image of André Agassi. The image is classified by a boolean rule as a photo on basis of previously extracted feature values, *e.g.* the number and average

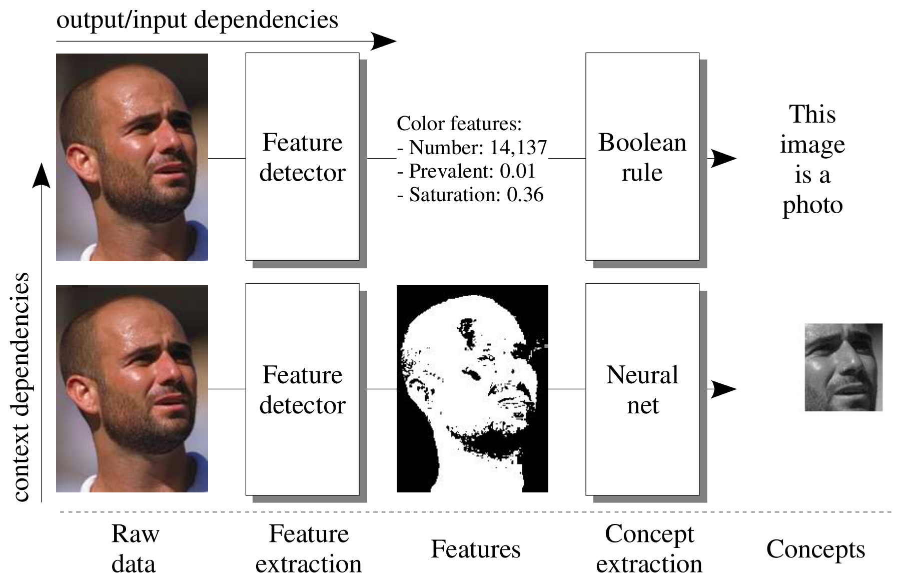


Fig. 2. Automatic information extraction steps.

saturation of the colors. Then a neural net, determines if the photo contains a human face, based on detected skin areas.

This example shows two kinds of dependencies: (1) *output/input dependencies*, *e.g.* the color features are input for the photo decision rule; and (2) *context dependencies*, *e.g.* the face classifier is only run when the image is a photo. Notice that context dependencies are inherently different from output/input dependencies. They are based on design decisions or domain restrictions and are not enforced by the extraction algorithm.

Feature grammars are based on a declarative language, which supports both output/input and context dependencies. Before introducing the language, the next section describes its solid and formal basis.

2.1 Feature Grammar Systems

As the name suggests *feature grammar systems* are related to grammars as known from natural languages, *e.g.* English or Dutch. Sentences in these languages are constructed from basic building blocks: words. Each language has a set of rules, which describe how words can be put in a valid order. These rules form the grammar of the language, and can be used to validate and build a parse tree of a specific sentence.

The fundamental idea behind feature grammar systems is that the same process of validation and derivation, and thus the underlying formal theories and practices, can be used as a driving force to produce the annotation of a

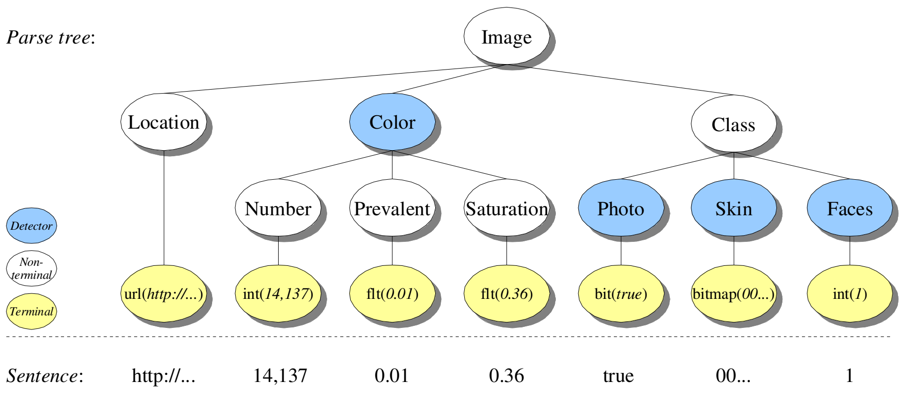


Fig. 3. Annotation sentence with parse tree.

multimedia object. In a feature grammar system individual information items in an annotation are seen as the words in an annotation sentence, on top of which a parse tree can be build (see Fig. 3).

The right-hand sides of grammar rules naturally capture contextual dependencies. However, they lack specification of output/input dependencies. The addition of feature grammar systems is the binding of extraction algorithms to specific grammar symbols, *i.e.* detectors. During derivation the output/input dependencies of these special symbols can be resolved within the parse tree. The following paragraphs will shortly introduce a formalization of this behaviour.

Grammars can cooperate in so-called *grammar systems* [6]. These systems where inspired by the will to model *multi-agent systems* according to the *blackboard architecture* [7]. In a *cooperating distributed (CD) grammar system* the common sentential form is on the blackboard and the *component grammars* are the knowledge sources, which take turn in rewriting this sentential form.

A feature grammar system is a specific instantiation of a CD grammar system, denoted as $(CD'_\infty CF, (lPC, C))$. In a feature grammar system there is a grammar component for each *detector*, or extraction algorithm. Such a grammar component describes, using a set of context-free (CF) production rules the output of the feature detector function. The output of the function is valid when it is a valid sentence in the language described by the grammar, which makes the grammar conditional (C). The input this function needs from the parse tree is specified by a regular path expression. To prevent deadlock situations only the left context of a detector is within its scope, allowing linear ordering of detectors. Hence the grammar is also a *left path-controlled (lPC) grammar*. The number of grammar components (and thus the detectors) is arbitrary, as indicated by the ∞ subscript. All components in a feature grammar system use the same alphabet (indicated by the prime (CD')).

2.2 Feature Grammar Language

The mathematical notation of a feature grammar system, although precise, is not very convenient in every day life. The *feature grammar language* allows to describe a feature grammar using just the normal ASCII set of characters. The basic version of the language allows the specification of exactly the core components of the underlying formal theory. The extended version allows the use of special shortcuts like modules, detector plug-ins and references. Due to space limitations the language is not described in detail but just an example is shown:

```
%module      Image;
%start       Image(Location);

%detector    Graphic(Number,Prevalent,Saturation);
%detector    Skin(Location);
%detector    matlab::Color(Location);

%classifier  bpnn::Faces(Skin);

%detector    Photo  [ Number     > 200
                  and Prevalent  <   0.26
                  and Saturation <   0.67 ];

%atom        www::url {^http://([^ :/]*)(:[0-9]*)?/?(.*)$};
%atom        image::bitmap {^[0|1]*$};

%atom url    Location;
%atom int    Number, Faces;
%atom flt    Prevalent, Saturation;
%atom bitmap Skin;

Image      : Location Color Class;
Color      : Number Prevalent Saturation;
Class      : Graphic | Photo Skin Faces;
```

This feature grammar describes declaratively the extraction process of Fig. 2. Notice that some detectors, *i.e.* whitebox detectors, and the detector parameters take the form of XPath expressions [8].

2.3 Feature Detector Engine

The *Feature Detector Engine (FDE)* uses a feature grammar and its associated detectors to steer an annotation extraction process. This process can now be implemented by a specific parsing algorithm. The control flow in a feature grammar system is top-down and favors leftmost derivations. The top-down algorithm used is based on an *exhaustive backtracking* algorithm. Backtracking indicates depth-first behavior: one alternative is chosen and followed until it either fails or succeeds. Upon failure the algorithm backtracks until an untried alternative is found and tries that one. The adjective exhaustive means that the algorithm also backtracks when the alternative is successful, thus handling ambiguity.

The FDE starts with validation of the start symbol, *i.e. Image*. The declaration of this start symbol specifies that at least the *Location* token should be available. In the example case the librarian provided the URL of the Australian Open picture of André Agassi. The FDE continues with building the parse tree until it encounters the first detector symbol: *Color*. The *Color* detector function needs the *Location* information, which is passed on as a parameter to the *matlab* plugin. This plugin connects to the matlab engine and requests execution of the *Color* function. The output of this function is a new sentence of three tokens: *Number*, *Prevalent* and *Saturation*. This sentence is subsequently validated using the rules for the *Color* detector. The FDE continues in this vain until the start symbol is proved valid.

The result of this proces is the following XML document:

```
<?xml version="1.0"?>
<fg:forest xmlns:fg="http://.../fg" xmlns:Image="http://.../Image">
 <fg:elementary idrefs="1@1" start="WWW:WebObject">
  <Image:Image id="5479@0">
   <Image:Location id="1@1">
    <Image:url id="2@1">http://...</Image:url>
   </Image:Location>
   <Image:Color idref="5480@0"/>
   <Image:Class id="9@1">
    <Image:Photo idref="5481@0"/>
    <Image:Skin idref="5482@0"/>
    <Image:Faces idref="5483@0"/>
   </Image:Class>
  </Image:Image>
 </fg:elementary>
 <fg:auxiliary>
  <Image:Color id="5480@0" idrefs="2@1">
   <Image:Number id="3@1">
    <fg:int id="4@1">14137</fg:int>
   </Image:Number>
   <Image:Prevalent id="5@1">
    <fg:flt id="6@1">0.01</fg:flt>
   </Image:Prevalent>
   <Image:Saturation id="7@1">
    <fg:flt id="8@1">0.36</fg:flt>
   </Image:Saturation>
  </Image:Color>
 </fg:auxiliary>
 <fg:auxiliary>
  <Image:Photo id="5481@0" idrefs="3@1 5@1 7@1"/>
 </fg:auxiliary>
```

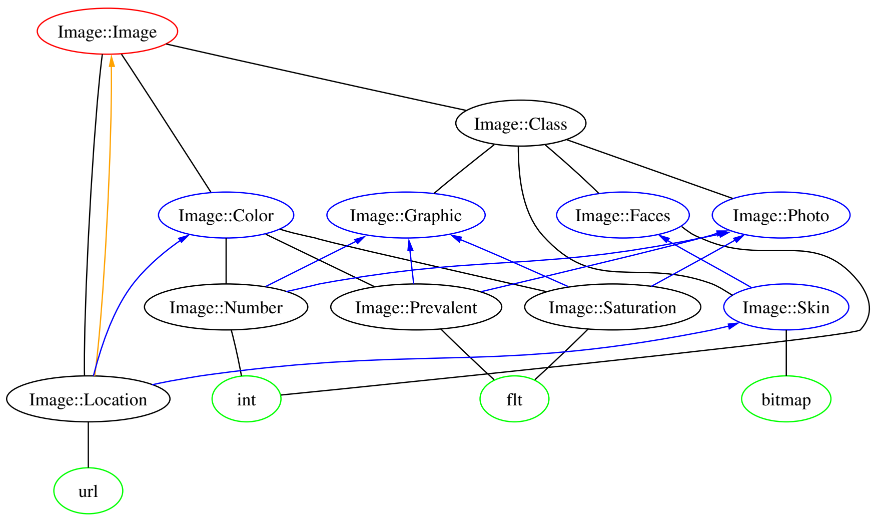


Fig. 4. Dependency graph.

```
    <fg:auxiliary>
     <Image:Skin id="5482@0" idrefs="1@1">
      <Image:bitmap id="10@1"><![CDATA[00...]]></Image:bitmap>
     </Image:Skin>
    </fg:auxiliary>
    <fg:auxiliary>
     <Image:Faces id="5483@0" idrefs="10@1">
      <fg:int id="11@1">1</fg:int>
     </Image:Faces>
    </fg:auxiliary>
   </fg:forest>
```

This document contains the constructed parse tree (see Fig. 3), and hence the desired annotation information. The database contains a collection of these XML documents, thus storing the content-based data of the multimedia collection.

2.4 Feature Detector Schedular

Over time the source data on which the stored annotation information is based may change. Also new or newer extraction algorithms may come available. The declarative nature of a feature grammar gives the opportunity to incrementally maintain the stored information. Based on the output/input and context dependencies embedded in the feature grammar a dependency graph can be build (see Fig. 4). Using this graph the *Feature Detector Schedular (FDS)* can localize changes and trigger incremental parses of the FDE. In an incremental parse the FDE only validates a single grammar component, thus only revalidating a partial parse tree. Following the links in the dependency graph the FDS can trigger other incremental parses based on the updated annotations.

The next section will describe the Webspace Method, which, combined with feature grammars, provide an advanced architecture for multimedia information retrieval.

3 Concept-based Search

When shifting the focus from the WWW to certain sub-domains of the Internet, it becomes feasible to apply some of the ideas originally developed for databases to the Web. This leads to a significant improvement in retrieval performance, when using a combination of concept-based search and content-based retrieval, as discussed in [2, 9]. This kind of search engines should particularly focus on large collections of documents, such as intranet environments, digital libraries, or large websites, where the content is semantically related.

The central idea behind conceptual search is to formulate the user's information need in a conceptual query that is based on a schema containing the important concepts used in the document collection. The *Webspace Method* for modeling and searching web-based document collections [9] exploits this idea in three different stages. Ideally, during the first stage, the document collection is constructed in four steps as explained in Sec. 3.1. The second stage is responsible for building an index for the conceptual and multimedia information that can be derived from the document collection. Finally, in the third stage the document collection can be queried, using the conceptual (database) schema that is defined in the first stage.

By using a conceptual schema to describe the content of the document collection, a semantical level of abstraction is introduced that in the end allows a user to query the document collection as a whole. This means that queries can be formulated combining information that is physically stored in several documents, whereas traditionally search engines base there answers on information found in a single document. The Webspace search engine allows the user to formulate exactly what information he is interested in as the result of a query.

To achieve this two steps are crucial. First of all during the modeling stage of the Webspace Method the information stored in the documents should be semantically related to the concepts defined in the conceptual schema. Using XML, the structure of the documents can be made explicit. Each XML-tag or attribute should then relate to a class, attribute, or association name in the conceptual schema.

Secondly, the integration of concept-based search with content-based retrieval also depends on the conceptual schema. For each of the different types of media, that is going to be included in the search engine a conceptual class

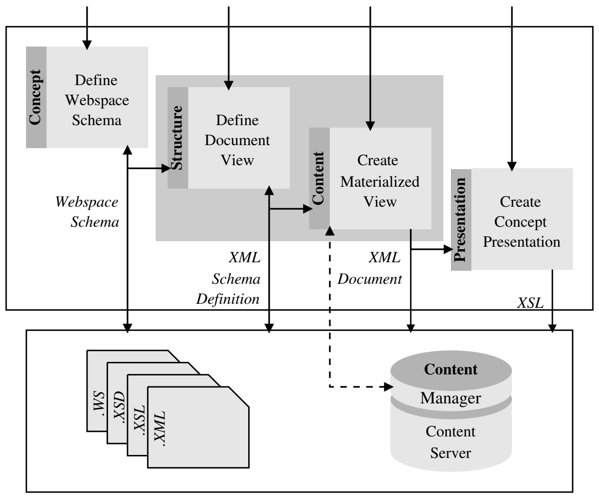


Fig. 5. Webspace Modeling Tool.

is included, and a link is made with the corresponding feature grammar. During the extraction stage all instances of multimedia classes are handed to the Feature Detector Engine for extracting the relevant meta-data.

The following subsections discuss the three stages of the Webspace Method in more detail, by the hand of an example based on the Australian Open case-study.

3.1 Modeling Stage

The modeling stage of the Webspace Method consists of four steps that are carried out to create a document collection. These four steps have been integrated in the Webspace modeling tool, which functions as an authoring tool for content management (see Fig. 5).

The first step is to identify concepts that adequately describe the content contained in the document collection. Once the *webspace schema* is (partially) defined, a view on the schema can be defined that describes the structure of the document that the author wants to create. This structure is then exported to an XML Schema Definition. Once the structure of the document is known, the content can be added. For maintenance reasons the content is ideally stored in a data management system, but the author can also choose to manually add the content to the document.

The result is an XML document that is seen as a materialized view on the webspace schema, since it contains both the data, and part of the conceptual schema. The XML document by itself is not suitable for presentation to the user. Therefore another XML-related technique is used for visualization purposes. During this last step, the author has to define a presentation for the document structure, using XSLT. Once this is done the document collection can be visualized and presented to the audience.

For standardization reasons it is desirable that the author starts with a default set of multimedia class definitions that enable the integration of concept-based search with content-based retrieval. Fig. 6 shows a fragment of the webspace schema that is used for the Australian Open webspace. It contains the class definition of a player with the attributes name, gender, country, picture, and history. The first three attribute are of type string, while the attribute picture and history are of class-type Image, and Hypertext, respectively. At this point the connection is made with the multimedia objects that are included in the document collection. Notice that Hypertext is used to refer to HTML-fragment that can be included in the XML document. This is motivated by the fact that there is always a trade-off between the concepts that are defined in the conceptual schema and conceptual information that is left out.

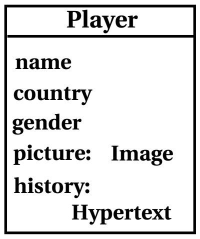


Fig. 6. Fragment from the Australian Open schema.

3.2 Extraction Stage

Before the Webspace search engine can be used, it is necessary to build an index that links the same instance of a class when it is defined in different documents. Furthermore, meta-data needs to be collected that realizes the integration of the conceptual framework used by the Webspace Method with the content-based retrieval component. In this case this is implemented by the FDE (see Sec. 2.3), which provides a highly advance interface to indexing multimedia objects that are found on the Web.

Each (XML) document of the collection is parsed for instantiations of classes and associations defined in the webspace schema. Relevant information is extracted and if a multimedia object is found the pointer to this object is stored and handed to the FDE, where the object is further analyzed.

3.3 Query Stage

Since the webspace schema allows us to identify the same object when it occurs in different documents, inter-document relations can be determined and used to enhance the retrieval performance of the conceptual search engine. Furthermore, this allows us to find information across the 'boundary' of a document. It can be concluded that a user rather queries the document collection as a whole, rather than querying the content of a single document. The figures in the next section show the process that is followed when formulating a conceptual query over a webspace using the Webspace search engine.

4 Combined Search

The combined (conceptual and content-based) search engine uses a schema-based approach for querying a document collection. The formulation of a query over a document collection, can be divided into three steps. During the first step, the *query skeleton* is constructed, using the visualization of the conceptual schema. Secondly, the constraints of the query are formulated, using the attributes of classes used in the query skeleton. In the last step, the user has to define the structure of the result of the query, which is generated as a materialized view on the conceptual schema.

Before continuing with the individual steps of the query formulation process, the queries presented below are used to illustrate the power of the webspace search engine with respect to query formulation. The queries express the typical information need of an expert user querying the 'Australian Open' document collection. It also shows, how after each query, the information need of the user can be refined, resulting in a more complex query over a document collection.

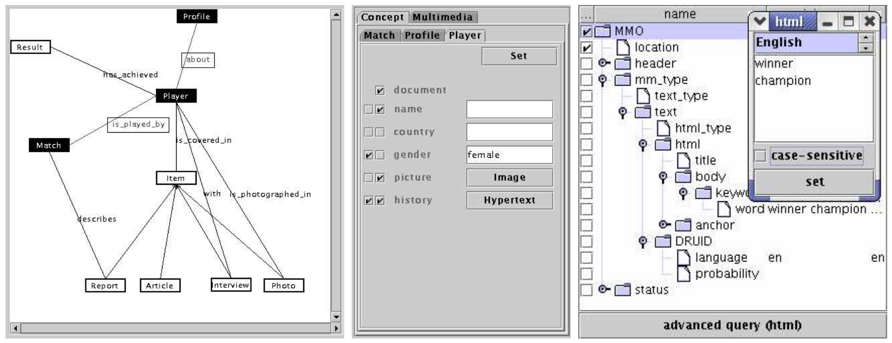
Q1.'*Search for left-handed female players, who have played a match in one of the (quarter/semi) final rounds. For each of those players, show the player's name, picture, nationality, birth-date, and the document URLs containing information about this player. Also show category, round and court of the match*'.

Q2.'*Like query 1, with the extra constraint that the player has already won a previous Australian Open. Include that history in the result of the query*'.

Q3.'*Extend query 2 with the constraint that the result should also contain video-fragments, showing net-playing events*'.

The first example query shows how conceptual search is used to obtain specific information originally stored in three different documents. The second example query extends the first query and provides an example illustrating the integration of content-based text retrieval in the conceptual framework of the Webspace Method. The third example query extends the complexity of the query even more, by integrating content-based video retrieval in the Webspace Method.

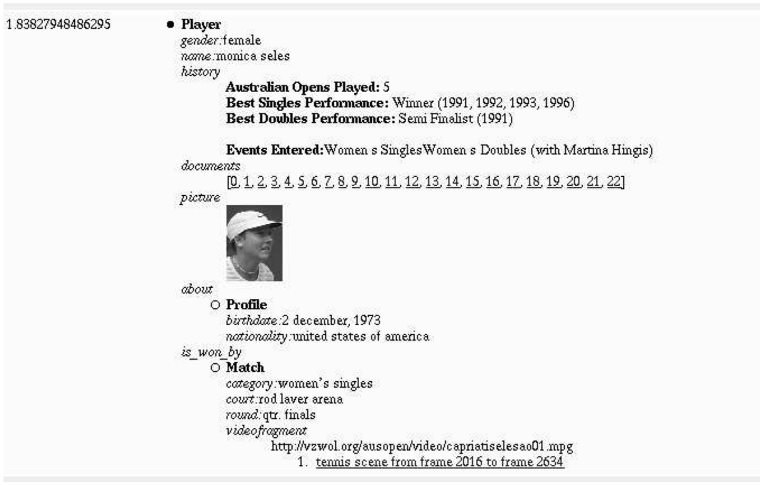
1. **Constructing the query skeleton**. The first step of the query formulation process involves the construction of the query skeleton. This skeleton is created, using a visualization of the webspace schema. This visualization consists of a simplified class diagram, and only contains the classes and associations between the classes, as defined in the webspace schema. The user simply composes the query skeleton, based on his information need, by selecting classes and related associations from the visualization. The (single) graph that is created represents the query skeleton.
   In Fig. 7.a a fragment taken from the GUI of the webspace search engine is presented, which shows the query skeleton (depicted in black-filled text boxes), that is used for the query formulation of the three example queries.



(a) (b) (c)

Fig. 7. Formulating a query.

2. **Formulating the constraints**. In the second step of the query formulation process, the constraints of the query are defined. In Fig. 7.b another fragment of the GUI of the webspace search engine is presented, showing the interface that is used for this purpose. For each class contained in the query skeleton a tab is activated, which allows a user to formulate the *conceptual constraints* of the query. As shown in the figure, a row is created for each attribute. Each row contains two check boxes, the name of the attribute, and either a text field or a button.
   The first checkbox is used to indicate whether the attribute is used as a constraint of the query. The second checkbox indicates whether the results of the query should show the corresponding attribute. If the type of the attribute is a BasicType, a textfield is available that allows the user to specify the value of the constraint, if the first checkbox is checked. If the attribute is of type WebClass, a button is available, which, if pressed, activates the interface that is used to query that particular multimedia object.
   Fig. 7.c shows the interface that is used to formulate queries over Hypertext-objects, *i.e.* define *content-based constraints*. The figure shows both a low-level and advanced interface to the underlying feature grammar system. In the low-level interface projection and selection criteria can be filled in. The advanced interface an interface similar to the interfaces offered by the well-known search engines such as Google and Alta-Vista. The main part of the query-interface allows a user to formulate one or more terms, which are used to find relevant text-objects, The interface also allows the user to perform a case-sensitive search, and to select the language of the Hypertext-object in which the user is interested.
   Fig. 7.b shows the attributes of class Player. The constraints with respect to the player, specified in the first two example queries, are transposed in the selections depicted in the figure. Two constraints are formulated. The constraint that the user is only interested in *female* players is defined



1.83827948486295

- **Player**
  gender:female
  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)
  documents
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]
  picture
  about
    - **Profile**
      birthdate:2 december, 1973
      nationality:united states of america
  is_won_by
    - **Match**
      category:women's singles
      court:rod laver arena
      round:qtr. finals
      videofragment
        http://vzwol.org/ausopen/video/capriatiselesao01.mpg
        1. tennis scene from frame 2016 to frame 2634

Fig. 8. View on 'Australian Open' containing the result of example query 3.

by selecting the constraint checkbox in front of gender, and by specifying the conceptual term '*female*'. The second constraint refers to the second example query, where an extra condition with regard to the player's history is formulated. Again, the corresponding checkbox is activated, and the interface of Fig. 7.c is started, and completed. In this case, the query-terms '*winner*' and '*champion*' are used to find the relevant Hypertext-objects that are associated with the player's history.

3. **Defining the resulting view**. The second column of checkboxes is used to specify which attributes will be shown in the resulting views defined by the query. The XML document that is generated by the webspace search engine contains a (ranked) list of views on the webspace that is being queried. Besides selecting the attributes and the classes that will be shown as the result of the query, the user also has to determine which class is used as the *root* of the resulting view. In Fig. 8 a screenshot of the result for the third query is shown. It includes a link to a tennis scene of the match played by Monica Seles in the quarter final round. The tennis scene shows a video-fragment in which Monica Seles plays near the net.

5 Conclusions and Future Work

This chapter introduced an approach for combining concept- and content-based search for multimedia objects. Both aspects are handled by seperate systems with very shallow connections, *i.e.* only a trigger. However, both approaches rely heavily on XML related technology and produce XML documents. Those documents can easily be stored in a shared XML database.

A specific search engine can now easily combine content- and concept-based queries, as has been illustrated by the Australian Open case-study.

As stated the connection between the two systems is still shallow. Interaction between the two systems can be increased, *e.g.* by allowing the feature grammar system to access conceptual data and *vice versa*. This will make the annotation process semi-automatic, and increases the opportunities for using knowledge in the automatic process and thus the quality of the extracted annotations.

References

1. Del Bimbo, A.: Visual Information Retrieval. Morgan Kaufmann Publishers, Inc. (1999)
2. van Zwol, R., Apers, P.M.: Retrieval performance experiment with the webspace method. In: proceedings of the 19th British National Conference on Databases, Sheffield, U.K. (2002)
3. Windhouwer, M.A., Schmidt, A.R., Kersten, M.L.: Acoi: A System for Indexing Multimedia Objects. In: International Workshop on Information Integration and Web-based Applications & Services, Yogyakarta, Indonesia (1999)
4. Windhouwer, M.A., Schmidt, A.R., van Zwol, R., Petkovic, M., Blok, H.E.: Flexible Digital Library Search. In Dahanayake, A., Gerhardt, W., eds.: Web-enabled Systems Integration: Challenges and Practices. Idea Group Publishing (2003) 200 – 224
5. Windhouwer, M.A., Schmidt, A.R., van Zwol, R., Petkovic, M., Blok, H.E.: Flexible and Scalable Digital Library Search. Technical Report INS-R0111, CWI, Amsterdam, The Netherlands (2001)
6. Csuhaj-Varhú, E., Dassow, J., Kelemen, J., Păun, G.: Grammar Systems: A Grammatical Approach to Distribution and Cooperation. Volume 5 of Topics in computer mathematics. Gordon and Breach Science Publishers (1994)
7. Nilsson, N.J.: Artificial Intelligence – A New Synthesis. Morgan Kaufmann (1998)
8. W3C: XML Path Language (XPath) 2.0, `http://www.w3.org/TR/xpath20`. (2001)
9. van Zwol, R.: Modelling and searching web-based document collections. PhD thesis, Centre for Telematics and Information Technology (CTIT), Enschede, the Netherlands (2002) ISBN: 90-365-1721-4; ISSN: 1381-3616 No. 02-40 (CTIT Ph.D. thesis series).

Index