Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 15482, Pages 1–11
DOI 10.1155/ASP/2006/15482

BeTrIS—An Index System for MPEG-7 Streams

Andrea Kofler-Vogt,[1] Harald Kosch,[1] Joerg Heuer,[2] and André Kaup[3]

[1] Department of Information Technology,University Klagenfurt, 9020 Klagenfurt, Austria
[2] Siemens AG, 80333 Munich, Germany
[3] Chair of Multimedia Communications and Signal Processing, University of Erlangen-Nuremberg, 91058 Erlangen, Germany

Received 8 September 2004; Revised 9 February 2005; Accepted 4 March 2005

The ISO/IEC Motion Picture Group (MPEG) issued in 2002 a standard, called MPEG-7, which enables the content description of multimedia data in XML. The standard supports applications to exchange, identify, and filter multimedia contents based on MPEG-7 descriptions. However, especially mobile applications that deal with MPEG-7 suffer from limited bandwidth, low computational power, and limited battery life. In this document, we describe an index system adopted from database systems that allows filter mechanisms and random access to encoded MPEG-7 streams and which overcome the limitation of the network and the consuming terminal. Encoding is applied in order to reduce the data rate of the XML documents to be transmitted. The indexed parts of the encoded streams can be accessed without the need to deserialize the complete stream. Furthermore, the system is evaluated and results of the experimental evaluation are discussed.

1. INTRODUCTION

MPEG-7 [1–3] is a multimedia description standard which has been standardized by the ISO/IEC Motion Picture Group (MPEG) to enable content providers and consumers to identify and search multimedia by content. MPEG-7 descriptions cover a wide spectrum of multimedia features like creation information, semantic information about places, persons, and/or events, low-level features (color, textures, sound), spatial-temporal decompositions, and so forth. Since its introduction in 2002, more and more applications use MPEG-7 to exchange their multimedia data between different parties. Examples are the M^3-Box [4] that is based on MPEG-7, the TV-Anytime forum (see [5]) which integrated the binary coding of MPEG-7 in its standard, or other application scenarios that are shown in [2]. Also, the constant progress of digital TV, the multiplication of channels, the competition and convergence with the Web, and the widespread deployment of a variety of set-top boxes call for new services that may rely on MPEG-7. For instance, broadcasters provide electronic program guide (EPG) services to their users or try to integrate TV with web technologies. The problem is that TV bandwidth is extremely expensive. Thus, a compression mechanism for XML-based MPEG-7 data is required. Furthermore, most set-top boxes are cheap, and the low-end ones have roughly half the power of a low-end mobile device. Due to this, the deserialization of compressed XML data must not be too complicated. If the data were sent as XML plain text, it would have to be fragmented so that set-top boxes would not have to wait for the end of the entire document to have been carouselled in order to start exploiting the data. That means the compression mechanism has not only to be efficient in terms of the reduction of the size and processing but should also work in a streaming environment and should allow the encoding and filtering of independent parts of the XML document.

In this context, the system part of MPEG-7 introduced a codec called binary format for metadata (BiM) [6] to enable efficient transmission of the XML-based MPEG-7 data. With this codec, the metadata is sent in a number of access units (AUs), each containing several fragment update units (FUUs). To speed up the random access on dedicated information within the stream, an index system for BiM-encoded metadata is required. This system should provide fast random access to particular fragment update units within an encoded stream. Consider as example the following application scenario. A user heard in the office that a blockbuster movie will be played this week on TV. But the user does not know on which day, at which time, and which channel will play this movie. The schedule of various TV-channels can be requested from a provider in form of an MPEG-7 description. The usage of an index system allows the user to process a quick stream lookup which returns the position of the relevant information in the BiM-stream. Thus, decoding exactly only the relevant FUU(s) which contains the broadcast information of the queried film is required instead of decoding, in the worst case, the complete stream containing the schedule of several TV stations over 7 days and then searching within the decoded data.

In this paper, we will present the conception and implementation of such an index system. It processes efficiently typical database operations (i.e., element queries, multiple-field queries) on MPEG-7 data streams. Performance evaluation will validate our approach.

This paper is structured as follows. First, we briefly discuss mechanisms and systems to index and compress XML data already available. In Section 3, our index system called BeTrIS (for B-tree index system) is described. There we show how a B-tree can be used to index XML data of MPEG-7 descriptions. Furthermore, we describe a mapping mechanism that allows the streaming of index trees. Section 4 describes improvements of the system in order to gain a more compact coding of the index information and to reach a better search performance. We compared the performance of our system against an alternative index system called XISS. Results of these evaluations are shown in Section 5. Finally, in Section 6, we summarize our work and give a brief overview of open issues.

2. RELATED WORK

An XML document such as an MPEG-7 description contains structured data, that is, elements that describe a hierarchy, and element contents and attribute values containing application-specific data. Transmitting such a description in plain text produces a high overhead concerning the storage consumption and thus is not applicable in many cases. Therefore, compression mechanisms for XML documents are required.

2.1. XML compression tools

A couple of compression tools were developed that take advantage of the structure of an XML document. Examples of such tools are XMill [7], Millau [8], XGrind [9], and XPRESS [10]. In the case of MPEG-7, a compression tool was required which achieves a high compression rate, which can be used in a streaming environment, and which allows the compression of independent subtrees. In order to provide these functionalities, MPEG-7 standardized its own binary format called BiM [6].

BiM assigns binary codes to elements and attributes based on the type declarations in the associated XML schema and allows the independent compression and decompression of single subtrees of an XML document.

However, random access mechanisms are not provided by the first version of BiM. Instead, the complete stream has to be parsed before a query can be processed. For fast random access to desired FUUs, an index system is required.

2.2. XML indexing strategies

A number of indexing solutions for XML have been proposed; among them are the Index Fabric [11], SphinX [12], a multidimensional indexing strategy [13], XISS [14], APEX [15], and ToXin [16, 17].

The *XML indexing and storage system (XISS [13]),* for example, makes extensive usage of the B-tree implementation called generalized search trees (GiSTs [18]). The B-trees are used to efficiently find all elements or attributes with the same name string. The basic index structure of this system is a numbering scheme [19] for elements and attributes of XML documents. This numbering scheme quickly determines the ancestor-descendant relationship between elements in the hierarchy of XML data. Furthermore, XISS includes several algorithms for processing regular path expressions called EE-Join, EA-Join, and KC-Join.

The indexing scheme *ToXin* (*Toronto XML indexing engine* [15, 16]) consists of two different types of structures: a *path index* and a *value index*. The path index can be used for forward and backward navigation in the document tree. It has two components: the *index tree* which is a *minimal dataguide* (see [20]) and a set of *instance functions*, one for each edge in the index tree. These functions store the parent-child relationship of the XML elements and are used to navigate in the index tree. The *value index* stores the XML nodes and values corresponding to an index edge.

Most proposed systems are optimized according to extended query functionality and provide for this a complex set of index structures. Mapping these kinds of indexes into one stream is difficult, first due to the complex reference mechanisms when index trees are aligned, second, these index structures are often not optimized for the requirement of a small storage size, which is crucial in the streaming context described in this paper. Thus, we will introduce an index system that is based on a B-tree. We will refer to this system as BeTrIS (for B-tree index system) in the rest of this work. BeTrIS provides fast access to certain FUU(s) in the streaming mode.

3. BeTrIS

3.1. Usage of a B-tree to index XML data

The B-tree [21] is a field-proofed data structure for fast index access in databases and file repositories.

The use of the B-tree in this context is motivated by the requirements of the streaming environment, such as

(i) block-oriented partition of the memory in mobile devices, that is, it is quite expensive to load a new block in mobile devices and the block size is quite small; the B-tree has been proven to be the most efficient data-structure [22] in such memory layout, minimizing the number of accessed data blocks;
(ii) efficient search behavior; since a B-tree is balanced, the logarithmic search behavior can be guaranteed;
(iii) network bandwidth restrictions; here, we propose an efficient coding schema of the B-tree (streamlined B-tree) to optimize the space consumption.

Further advantages are that the B-tree may be applied to any kind of data and that updates caused by insertions or deletions affect only a limited number of nodes.

The usage of the B-tree in the context of XML documents is the following. Paths of a description tree are taken as index keys. Contexts, such as attribute and element values, are stored along the respective paths. In this work, only root-to-leaf paths are considered. Due to this, only leaf elements or attributes of a document tree are indexed.

In order to index the data, first the relevant paths of a document to be indexed are generated. These paths, without the context and the values of attributes, serve as keys. They are inserted into the index tree.

Figure 1 shows parts of the generated B-tree, after all root-to-leaf paths of a small example document were inserted. The example B-tree is of order 2, thus any node must have at least one key entry ($n - 1$) and can have at most three entries ($2n - 1$). The same path may occur in a document several times with different instance values. However, a key is added only once into the tree. For this reason, for each key the number of occurrences of the path in the document tree, the values of attributes, or the context of elements, and where in the stream the FUU(s) containing the indexed nodes can be found, is transmitted too.

For better readability, only the beginning letters of each path element are shown in Figure 1. For this reason, the string MDMVUFACN stands for the path "Mpeg7/Description/MultimediaContent/Video/UsageInformation/FinancialResults/AccountItem/CostType/Name." This path appears in the example document twice, once the Name has the value "Total for Production" and once "Broadcast."

3.2. *The mapping of the tree to a linear stream*

Once the index structure is generated, the information is stored in the index stream. In a typical scenario, the index stream is sent together with or before the description stream. In any case, the hierarchical B-tree is mapped to a linear stream.

For the stream mapping, the tree is traversed in a depth-first manner, and the nodes are packed into the stream in the order they are visited. This order is signaled in Figure 1 with the node numbers of the tree.

For each node in the stream, the start offset of all children beside the first child is coded too. This enables a skipping of undesired node information. If a node does not have any children, the offset is set to zero. For the first child node, the offset is not transmitted since it is coded immediately after its parent node. Figure 2 shows parts of the stream for the B-tree represented above.

In this example, the node numbers are shown only for better understanding. They are not transmitted in the index stream. For one node in the B-tree, the stream stored the following information. First, the numbers of key entries are written in the stream, for example, in Figure 2, node number one contains exactly one key. Afterwards, for each key entry first, the key is written in the stream. After the key, a number signals how much instances for this key are available in the document. As shown in Figure 2, the key MDMVUAFCN appears twice in the indexed document. For each instance, the value of the key (value of the attribute or the context of an element) is written in the stream. After the value, a reference to the occurrence in the indexed stream is coded. In our implementation described next, such a reference provides the index of the access unit and fragment updated unit in order to identify the FUU containing the indexed leaf nodes. At the end of each node, the offsets of the children beside the first one are written in the tree stream.

3.3. *The index search based on the tree stream*

We consider an XPath [23] like query string as input. An index search in the B-tree starts at the root node. Through comparison of the search pattern with the keys, it is decided in which child node the searched entry can be found. Thus, if the desired key is not in the parent node, only one child has to be visited.

The usage of offsets in the index stream enables one to directly process the proper node without having to parse the information of other nodes stored in between both nodes in the stream. Thus, during the lookup, particular node information is read until the node content matches the query string, or no more children are available. In the latter case, the desired information is not present in the indexed document.

As an example, the following query: "Mpeg7/Description/MultimediaContent/Video/UsageInformation/Availability/Dissemination/Disseminator/Agent/Name" where the name of the agent is "Discovery" short (MDMVUADDAN = "Discovery") should be processed. Comparing the query string with the entry in the first node leads to a further proceeding in the second node. The search string is smaller than its key. Thus, node number 3 is considered next. Since the query string is greater than the first key in this node, but smaller with respect to the lexicographical ordering than its second key, the search proceeds at the node with the number 5. This node can be directly accessed through the usage of the coded child offset. Here, the string matches with the first key and the FUU containing the desired element can be identified. The search path through the original B-tree for this example is presented in Figure 3.

In this example, only the information of the nodes 1, 2, 3, and 5 is parsed, the other data in the stream is skipped or ignored.

4. IMPROVEMENTS OF THE PROTOTYPE

4.1. *Compacter coding of the index information*

In order to carry the index information in a compact manner, strings are convoyed in a string repository. This repository carries each string exactly once. In the index tree, their start offsets are used instead of the binary representation of strings.

First experiments have shown that the size of the index information may become almost that big as the size of the BiM encoded data. Thus, we considered improvements in

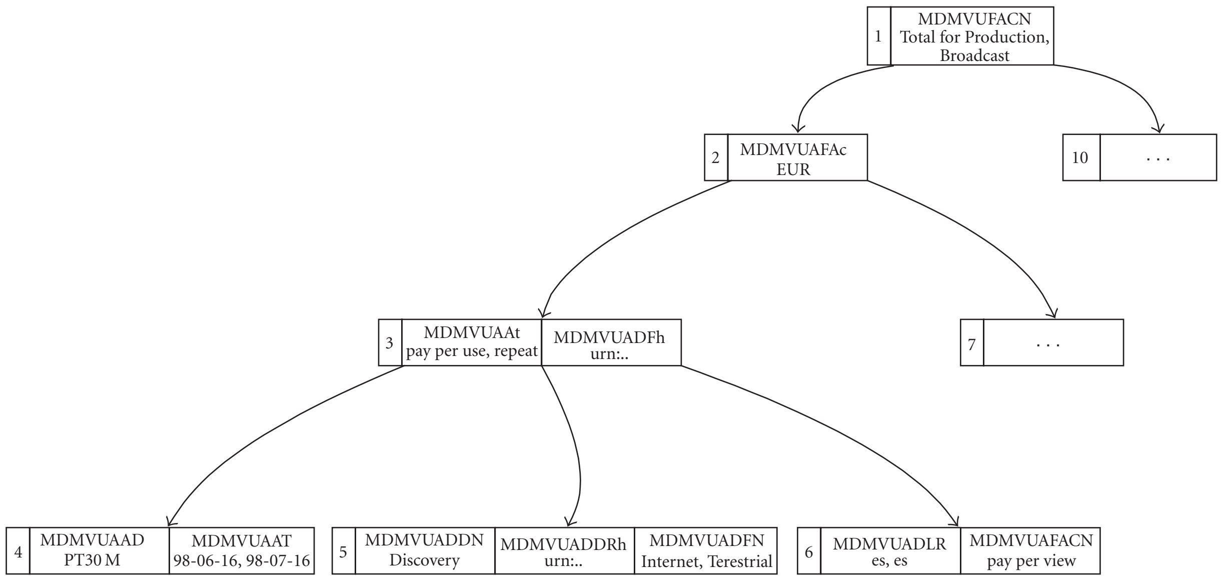

Figure 1: Parts of a B-tree.

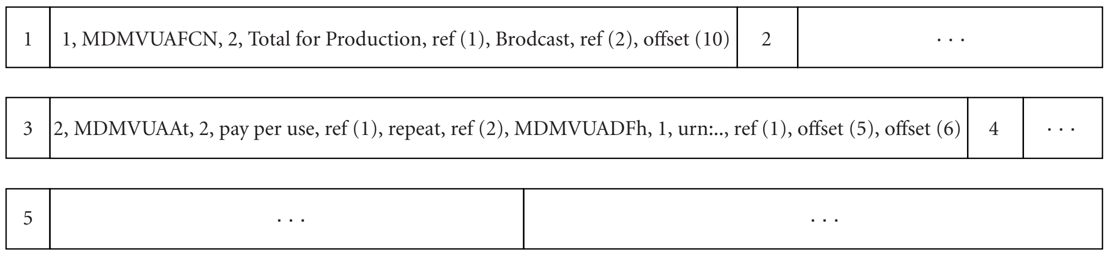
1
1, MDMVUAFCN, 2, Total for Production, ref (1), Brodcast, ref (2), offset (10)
2
. . .
3
2, MDMVUAAt, 2, pay per use, ref (1), repeat, ref (2), MDMVUADFh, 1, urn:.., ref (1), offset (5), offset (6)
4
. . .
5
. . .
. . .

Figure 2: Example of an index stream.

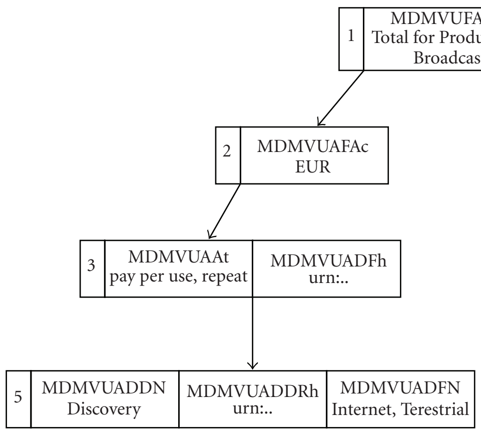

Figure 3: An example of a search path.

order to reduce the index size. As a first step, we decided to use the binary coding of the path in the indexed tree as standardized by MPEG-7 BiM (see [24]). In experimental comparisons, we could achieve a reduction of the storage size of about 38%. The concrete reduction depends on the test data, that is, for documents with fewer instances per indexed path, this rate is higher than for those with many instances per indexed path.

4.2. *Extended query functionality*

The system described so far supports only simple path queries. Such a query consists of a path containing several elements and possibly one attribute and optionally asks for a certain value. For example, with the following query, all images are identified that were taken by a person (i.e., creator) whose last name is "Vogt": Mpeg7/Description/MultimediaContent/Image/CreationInformation/Creation/Creator/Agent/Name/FamilyName = Vogt. For a first evaluation, this query functionality was sufficient. However, in many applications, more complex query functionality is required.

Due to this, we extended our prototype, in order to support multiple field queries which contain more than one condition. For example, we may extend the query from above, that is, not only the creators last name is of interest but the picture needs also to show a person whose last name is

"Kofler." This query can be formulated as XPath expression as follows: Mpeg7/Description/MultimediaContent/Image/CreationInformation/Creation/Creator/Agent/Name [FamilyName = Vogt][../../../../../Semantic/SemanticBase/Agent/Name/FamilyName = Kofler]. Here, "../" denotes one stepup in the hierarchy.

The system we explained above transmits the keys as binary encoded paths. A binary encoded path consists of a sequence of tree branch codes (TBCs) which are tokens for the elements appearing in this path followed by a sequence of position codes for those elements that may occur more than once (see [24]). Because BeTrIS was optimized concerning the storage size, the position codes were omitted in our first approach. But while they do not influence the navigation within the B-tree, we propose to use them to answer more complex queries. In the modified version of BeTrIS still only the tree branch codes serve as key entries in the B-Tree. But in order to provide more complex query functionality, the position codes are transmitted too. This leads to a modified syntax of the index stream which is shown in Figure 4.

As mentioned above, a multiple-field query contains at least two conditions. We propose to perform a separate search for each condition and merge the resulting list by the usage of position codes to a final result set. Due to this, a multiple-field query is divided into a common prefix and at least two conditions. For instance, the example query from above can be separated in the following parts:

(i) prefix = Mpeg7/Description/MultimediaContent/Image;
(ii) condition$_1$ = prefix/CreationInformation/Creation/Creator/Agent/Name/FamilyName = Vogt;
(iii) condition$_2$ = prefix/Semantic/SemanticBase/Agent/Name/FamilyName = Kofler.

After the separation of the query a search in the B-tree is processed for each condition. Each search returns a list with matching instances. Thus, in our example, two lists will be created.

For better understanding, this will be explained more detailed on an example. Figure 5 shows parts of a document tree containing an MPEG-7 description of three pictures.

Processing the query for the first condition (all creators with the last name "Vogt") leads to two matching instances. The context nodes of these instances are highlighted with dotted lines in Figure 5. For the instances the following position code sequences will be returned: 1,2,1,1,2,1,2,2,1,2 and 1,3,1,1,2,1,2,2,1,2.

Similar, a search for the second condition (any picture showing a person with the last name "Kofler") again leads to a list of 2 entries: 1,1,1,1,4,2,2,1,2 and 1,2,1,1,4,2,2,1,2.

The proper context nodes of the two instances are highlighted in gray in Figure 5.

Each entry of the first list is then compared with each entry of the second list with respect to their position codes. Relevant for the comparisons are those position numbers which are required to encode the positions of the common prefix. In our example, the prefix Mpeg7/Description/MultimediaContent/Image consists of four elements. Thus, the first four numbers of each identified position code sequence are compared.

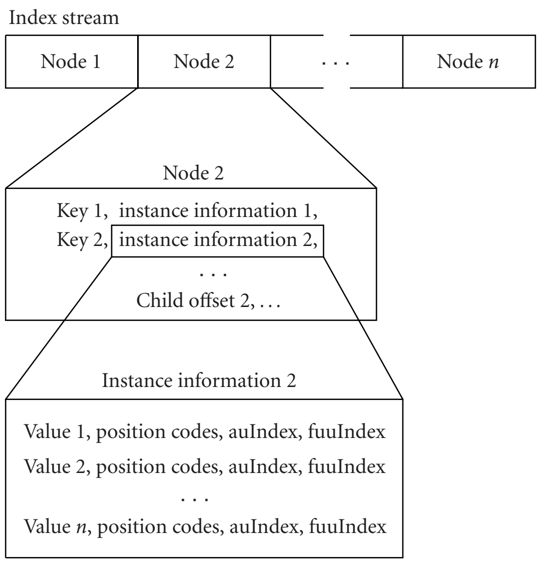
Index stream
Node 1
Node 2
. . .
Node n
Node 2
Key 1, instance information 1,
Key 2, instance information 2,
. . .
Child offset 2, . . .
Instance information 2
Value 1, position codes, auIndex, fuuIndex
Value 2, position codes, auIndex, fuuIndex
. . .
Value n, position codes, auIndex, fuuIndex

FIGURE 4: Modified syntax of the index stream.

If for one entry in the first list an entry with matching position codes for the prefix is found in the second list, this means that the GivenName element of the Agent element and the GivenName element of the Creator element both have the same Image element as a predecessor. And thus, this path will belong to the final result set. In our small example, entry one of the first list starts with the same four position codes as entry two of the second list. Thus, only one Image node exists that has a proper child node for each condition. This node is identified through the position codes 1,2,1,1 and it is highlighted in gray having dotted lines in Figure 5.

Note that MPEG-7 position codes are only encoded for those elements that may appear several times according to the type definitions in the MPEG-7 schema. Thus, the sequences of position codes are smaller than those shown in Figure 5.

4.3. Final system with improvement: data organization

While we evaluated the improvements discussed above, we encountered three major drawbacks.

(i) If a matching entry is found in the index tree, all values of the instances have to be processed in a sequential manner in order to identify the matching values and generate the result set. Consider, for example, an MPEG-7 document that describes 50 pictures. If a query is performed that searches for a certain creator after identifying the matching key entry in the index tree, a set of 50 values (called instance information) has to be processed sequentially in order to identify those pictures that were taken by the desired creator.

(ii) Although in the case that the current key entry does not match the query string, the complete instance

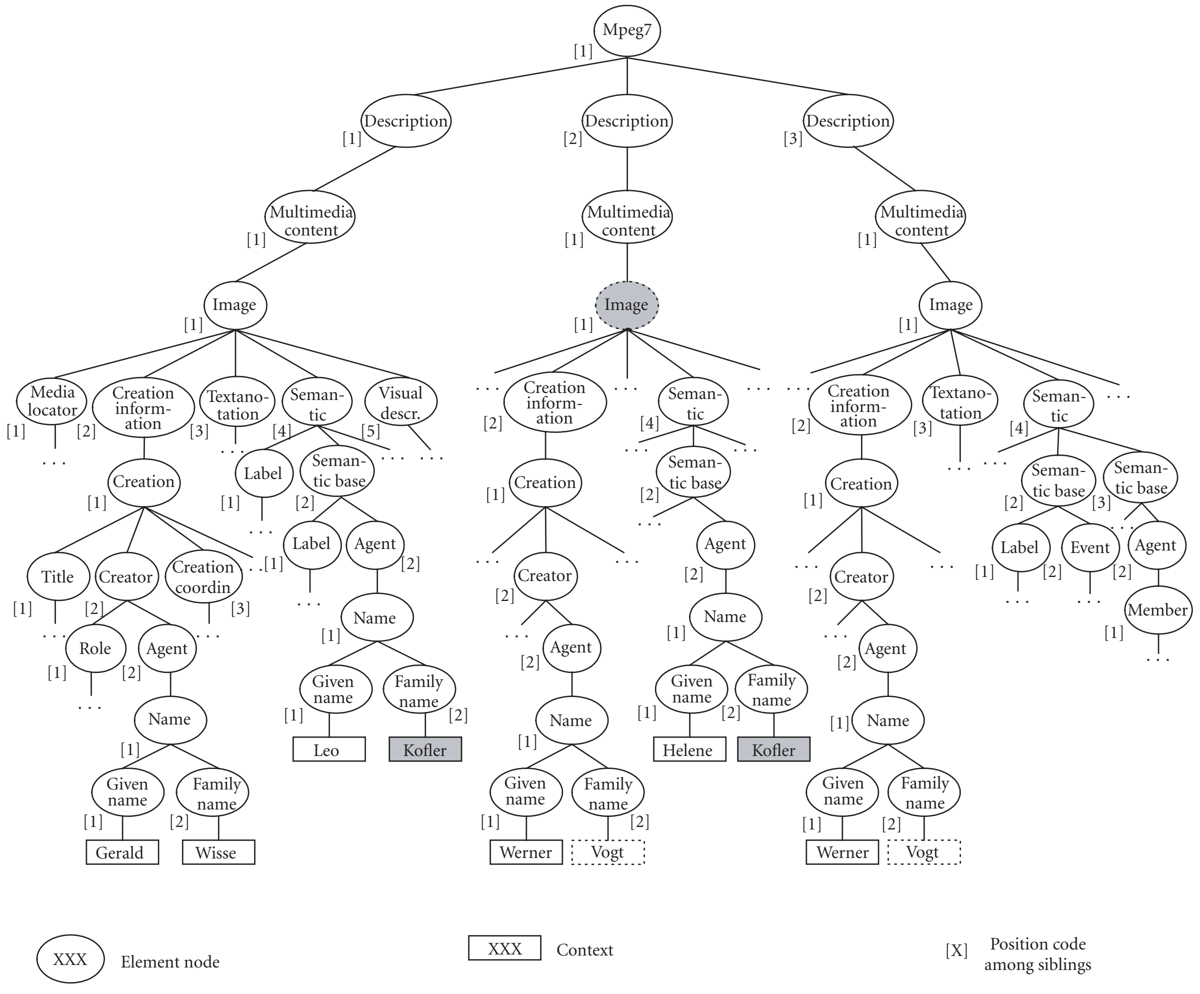

Figure 5: Parts of a document tree.

information of this entry (i.e., values, position codes, etc.) has to be read out of the index stream in order to reach the next key of the encoded node. This is because the instance information is coded immediately after its key (see Figure 4) and no skipping information is available in the current approach.

(iii) If the decision has been taken that the searched key is not in the current node and the child to be next visited is already identified, although the complete information of the current node (i.e., remaining keys and their instance information) has to be read. The reason for this inconvenience is that the child offsets are coded at the end of the node after all keys and their instance information and again, no skipping information is available.

To overcome the first problem, we decided to implement a data organization that allows a more efficient search within the indexed values. We built a data structure called value tree for each key entry (see Figure 6). By storing these value trees outside the index tree and simply referencing them, we already found a solution for the later two problems. Now the instance information is not present any longer within the index tree and thus is only accessed if necessary. In this section, we will provide a detailed description on this new idea concerning the data organization.

In order to avoid a sequential search within the instance information of an indexed path, we implemented an extension of BeTrIS, where the instance information is organized in a simple binary tree. The tree is sorted by the values of the different instances belonging to the same path.

Consider, for example, the following list of values that appear in an indexed document as instances for a certain path: *Daniel, Christina, Alex, Andrea, Rainer, Hans, Stefan, Hiltrud, Agnes, Ulrike, Daniel, Hans, Andrea, Stefan, Rainer, Rainer, Andrea, Stefan, Hans, Hiltrud,* and *Agnes*. These names are inserted in a simple binary tree which is shown in Figure 7.

Such a tree is created for each indexed path. Thus, each entry in the B-tree points to a binary tree (see Figure 6).

Each different value appears only once in the value tree although it may occur several times for the indexed path. Due to this, it is necessary to store additional information in each node, like, for example, the position codes for each

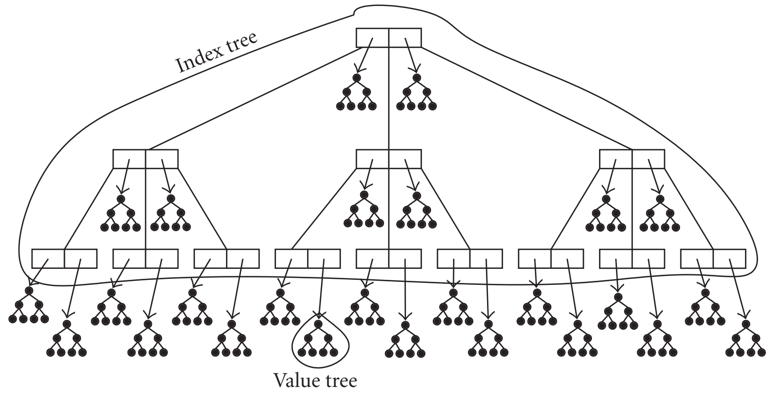

Figure 6: Index tree referencing several value trees.

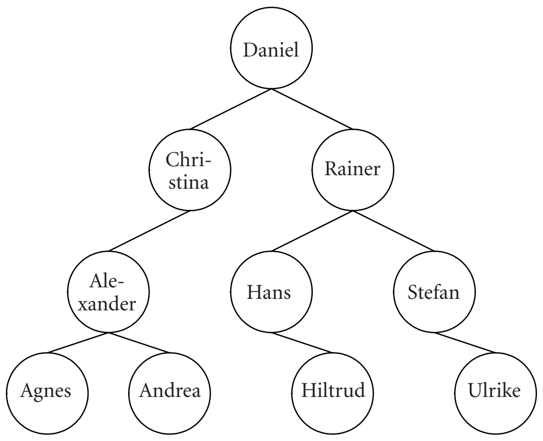

Figure 7: Example of a value tree.

appearance, and the reference to the actual occurrence of the indexed path in the encoded description stream.

When creating the index, a B-tree is set up where each indexed path references a binary tree (see Figure 6). Before transmitting this index, all the information has to be mapped in a flat stream. Similar to the mapping of the B-tree containing the indexed paths, each value tree has to be mapped to a linear stream too. This mapping is performed for each value tree and their encoded information is packed at the end of the index stream. Thus, the index stream consists of three different parts (see Figure 8):

(i) the stream of the linear B-tree,
(ii) the string repository,
(iii) the stream containing all value trees.

Experimental performance evaluations have shown that the usage of data organization leads to a reduction of the index size if there are several instances per indexed paths. This is because in the modified BeTrIS, each value for a certain path is stored only once and then the data for each occurrence is presented. In the previous system, the offset to the same value was possibly stored several times for the same path. On the other hand, if there are only a few instances per path, the modified system leads to a further overhead which is founded by the more complex structure of the value tree in comparison to a sequential list. The concrete results of these measurements are shown in Table 1. We used the test data set described in Section 5.1.

Concerning the query efficiency which was measured in number of comparisons, the experiments have shown that even for documents with a small number of instances per indexed paths a better performance is visible. This is founded in the logarithmic search behavior reached by the usage of binary trees. In the former solution, the complete set of indexed values for a matching path have to be compiled. For those documents that have a greater number of instances, the reduction of comparisons was between 92% and 99% in our experiments.

5. PERFORMANCE CONSIDERATIONS

We performed a series of experiments to validate our indexing solution. We were interested in two factors: the storage size of the index system and the number of comparisons to be performed during the index search.

In order to get an overview of the performance, we evaluated the final BeTrIS system (as described in Section 4.3) against a simulation of XISS described in Section 2.2 and [14]. We choose this system, because it provides a rich set of functionalities and because of the extensive usage of the concept of B-trees, it was easily comparable to our system.

5.1. The test data set

We indexed a number of subtrees of three large example documents.

(i) Pictures.xml. This document contains an MPEG-7 description of 50 different pictures describing persons, places, objects, and so forth appearing on these pictures. A total number of 2.483 paths are indexed for this document. Among them, there are 82 different ones that serve as key entries in the B-tree. This leads to an average number of 30 instances per path.

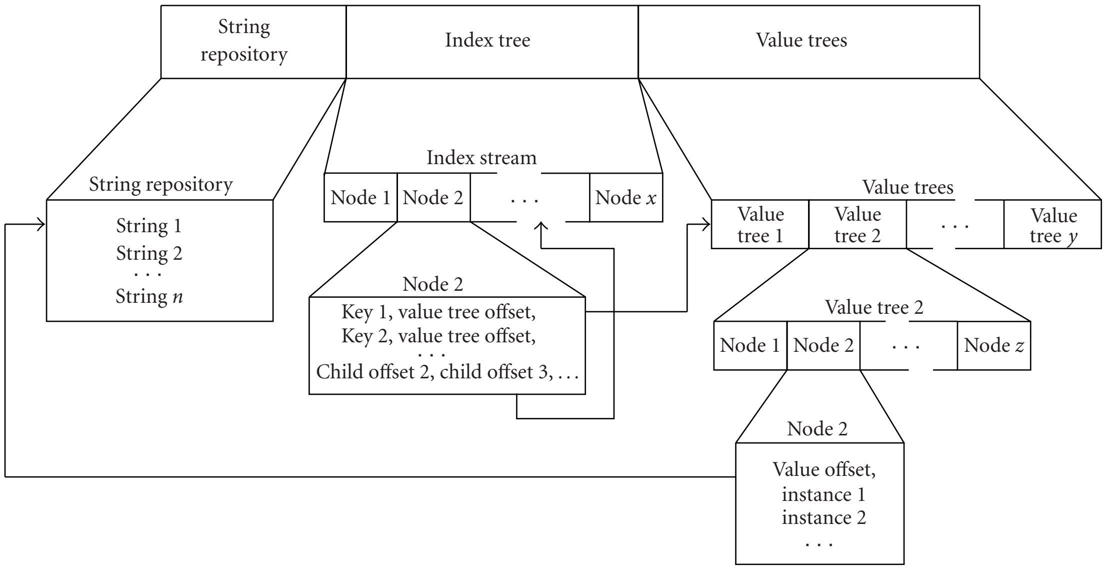

Figure 8: Modified index stream.

Table 1: Size in bytes of the transmitted index information.

Document	Original BeTrIS	Extended BeTrIS	ANP[1]	Difference
Pictures	31.997	30.711	30	−4,02%
Advertisement	41.140	36.601	106	−11,03%
DescriptionEx	14.978	17.806	2	+18,88%
Average	22.029	21.280	46	+1,28%

[1]ANP stands for average number of path instances, that is, how often the same path occurs in the indexed document.

This description allows to ask, for instance, for all pictures that show certain persons or places or that were created by a certain person.

(ii) Advertisement.xml. This document contains an MPEG-7 description of 100 advertisement spots. For this document, a total number of 2.960 paths are indexed. Among them there are 28 different paths which lead to an average of 106 instances per path.

The description can be used to find advertisement spots which last a certain time period or spots that advertise for a certain product or all advertisements of products of a certain company, and so forth.

(iii) DescriptionExample000.xml. This document contains an MPEG-7 description that demonstrates the usage of different descriptors and description schemes defined in the MPEG-7 standard. There are 714 paths indexed, among them there are 361 different ones. This leads to an average number of 2 instances per path.

The example queries, for instance, ask for color histograms with certain properties, segments of a certain duration, and so on.

The queries we formulated for each example document return a different number of indexed fragment update units as result.

5.2. The index size

For each test document, we measured the size in bytes required to transmit the complete index information of the two systems. In both cases, the information of different components has to be encoded.

As already mentioned above, BeTrIS consists of the following parts.

(i) The string repository which contains all values of attributes and the content of elements. All these strings are transmitted in an uncompressed manner.

(ii) The information of the inline B-tree is transmitted in the index stream.

(iii) Finally, all value trees are mapped to a linear stream and packed in the value stream.

On the other hand side, for XISS the following parts have to be transmitted:

(i) the value table which maps a values to value IDs (vids);
(ii) the name index which maps names of elements of attributes to name IDs (NIDs);
(iii) the attribute index which maps NIDs of attributes to the occurrence information of the proper attribute nodes;

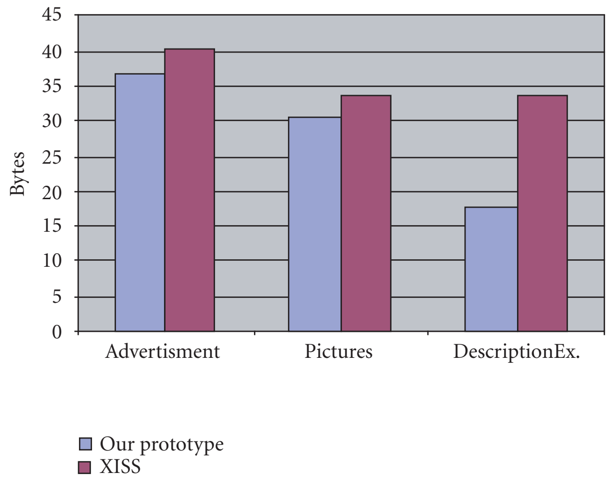

Figure 9: The index size in bytes.

(iv) Finally, the element index maps the NIDs of elements to the instance information of the indexed elements.

All these components are realized as B-trees. We implemented them using our approach of flat trees.

As shown in Figure 9, the index information to be transmitted for BeTrIS is in all three cases smaller than that of XISS.

Tables 2 and 3 show the average storage consumption evaluated for both systems more detailed.

There are several reasons, why our prototype is better concerning the storage size than the alternative system. First, the systems use different modes to index the same data. In BeTrIS only leaf nodes are indexed by storing the XPaths leading at these nodes. Contrary, in XISS each single node is indexed separately. Second, BeTrIS compresses the data by using the binary coding of paths while in XISS the complete information is transmitted in an uncompressed manner. Binary coding of paths is not applicable to XISS because there a numbering scheme [19] is used to store parent-child relationships while in our case those are resolved through path expressions. Finally, the redundant data XISS stores in its different components also leads to a bigger index size.

5.3. Query efficiency

As mentioned above, for each example document, several queries were formulated. These queries were performed in both systems and afterwards the average number of required comparisons for each document was calculated.

BeTrIS requires one search in the index tree and a further search in the identified value tree to answer a simple path query. Contrary, in the case of XISS, two search operations are necessary for each path step of the query. One to identify the NID of the element or attribute name and a second one to find all occurrences stored for this NID. Similar, two search operations are required for the searched value. Afterwards, several Join-Algorithms are performed as proposed in [14] in order to identify parent-child relationships. This causes further comparisons required to answer a query. Due to this, XISS produces a significant higher number of comparisons as shown in Table 4.

Table 2: Detailed storage consumption of BeTrIS.

Component	Size in %
String repository	3%
Index stream	12%
Value stream	85%

Table 3: Detailed storage consumption of XISS.

Component	Size in %
Value table	28%
Name index	6%
Element index	54%
Attribute index	12%

Table 4: Average number of comparisons.

Example document	BeTrIS	XISS
Advertisement.xml	15	41.120
Pictures.xml	12	80.621
DescriptionExample.xml	12	2.061

5.4. Summarization of the performance consideration

The evaluation of the two systems has shown that the index size of BeTrIS is smaller than that of XISS. On average, there is a difference of about 10.5%. Detailed analyses of these results have shown that on average 35% of the complete index size of our system is occupied by the position codes. Thus, at the moment we are examining mechanisms for differential coding of these codes which is estimated to lead to a significant reduction of the storage size. Furthermore, we plan to compress the values in the string repository which are transmitted in an uncompressed manner at the moment.

Concerning the query efficiency which was measured in number of comparisons, BeTrIS could present significant better results (factor of approximately 1:3000). This is founded in the design of XISS which is in favour of efficiently answering queries of several types in applications with sufficient memory space and CPU power. Furthermore, XISS supports more complex query functionality. For example, it is able to answer wildcard queries (when the exact nesting of nodes is not known) while BeTrIS in its current implementation does not support those queries. However, the index presented in this paper focuses on environments with limited bandwidth, memory, and/or processing power.

6. CONCLUSION AND OUTLOOK

In this work, we proposed an index system that provides fast random access to binary encoded MPEG-7 descriptions.

The approach was motivated by the increased usage of MPEG-7 in mobile multimedia application relying on metadata for identification, filter, and search, for instance, electronic program guides (EPG), multimedia mail services, and so forth.

With our approach, we have shown how a B-tree can be used to index XML-based MPEG-7 data. We proposed a coding scheme that allows the streaming of an index tree. Additionally, we improved our prototype by applying the binary coding of XML-based path structures in order to reduce the index size. The functionality was extended to support multiple-field queries. For this purpose, we utilize the position codes of the MPEG-7 BiM. Finally, we achieved a decrease in search time by the usage of value trees for the element or attribute values which belong to the same path.

Detailed performance evaluations against an alternative system called XISS have shown that our index system requires less storage size for the complete test data set. While about 10% seems to be a minor difference, we discussed some ideas that allow a further reduction of the storage size. At the moment, we are working on these compression mechanisms which is estimated to increase the difference significantly. Comparing the number of required comparisons to answer a query has shown that the presented significantly improves processing efficiency. With our query set, we not only demonstrated the efficiency of our index system concerning searching and filter mechanisms but we also have shown different application scenarios for the usage of MPEG-7-based description.

As already mentioned above, further work addresses the reduction of the index size. Especially type-specific compression algorithms for the values currently stored in string repositories have to be considered. Beside that compact, for example, differential encoding of position codes have to be investigated. Based on the observation that on average 35% of the present index stream consists of position information, improvement in the position coding can have significant effect on the index size. Finally, we will evaluate the gain of the reordering of entries in the index tree, when the frequency of different queries is considered too. At the moment, we assume a uniform distribution of the key entries in the tree.

ACKNOWLEDGMENT

This project was supported by the Austrian Science Fund (FWF) under Project no. P14789.

REFERENCES

[1] J. M. Martínez, "Overview of the MPEG-7 standard," in *ISO/IEC JTC1/SC29/WG11 N4980*, Klagenfurt, Austria, July 2002, available from http://www.mp7c.org/.

[2] H. Kosch, *Distributed Multimedia Database Technologies Supported by MPEG-7 and MPEG-21*, CRC Press, Boca Raton, Fla, USA, 2003, 248 pages.

[3] B. S. Manjunath, P. Salembier, and T. Sikora, *Introduction to MPEG-7*, John Wiley & Sons, New York, NY, USA, 2002.

[4] J. Heuer, J. L. Casas, and A. Kaup, "Adaptive multimedia messaging based on MPEG-7—the M3-box," in *Proceedings of 2nd International Symposium on Mobile Multimedia Systems & Applications (MMSA '00)*, pp. 6–13, Delft, the Netherlands, November 2000.

[5] Official homepage of the TV-Anytime forum (see http://www.tv-anytime.org/).

[6] U. Niedermeier, J. Heuer, A. Hutter, W. Stechele, and A. Kaup, "An MPEG-7 tool for compression and streaming of XML data," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME '02)*, vol. 1, pp. 521–524, Lausanne, Switzerland, August 2002.

[7] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 153–164, Dallas, Tex, USA, May 2000.

[8] M. Girardot and N. Sundaresan, "Millau: an encoding format for efficient representation and exchange of XML over the Web," in *Proceedings of 9th International World Wide Web Conference*, Amsterdam, the Netherlands, May 2000, http://www9.org/.

[9] P. Tolani and J. R. Haritsa, "XGRIND: a query-friendly XML compressor," in *Proceedings of IEEE International Conference on Data Engineering (ICDE '02)*, pp. 225–235, San Jose, Calif, USA, February–March 2002.

[10] J.-K. Min, M.-J. Park, and C.-W. Chung, "XPRESS: a queriable compression for XML data," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 122–133, San Diego, Calif, USA, June 2003.

[11] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon, "A fast index for semistructured data," in *Proceedings of 27th International Conference on Very Large Data Bases (VLDB '01)*, pp. 341–350, Rome, Italy, September 2001.

[12] L. K. Poola and J. R. Haritsa, "SphinX: schema-conscious XML indexing," Tech. Rep. RE200104, Indian Institute of Science, Bangalore, India, 2001.

[13] H. V. Jagadish, N. Koudas, and D. Srivastava, "On effective multi-dimensional indexing for strings," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 403–414, Dallas, Tex, USA, May 2000.

[14] Q. Li and B. Moon, "Indexing and querying XML data for regular path expressions," in *Proceedings of 27th International Conference on Very Large Data Bases (VLDB '01)*, pp. 361–370, Rome, Italy, September 2001.

[15] C.-W. Chung, J.-K. Min, and K. Shim, "APEX: an adaptive path index for XML data," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 121–132, Madison, Wis, USA, June 2002.

[16] F. Rizzolo and A. O. Mendelzon, "Indexing XML data with ToXin," in *Proceedings of 4th International Workshop on the Web and Databases (WebDB '01)*, pp. 49–54, Santa Barbara, Calif, USA, May 2001.

[17] A. O. Mendelzon, "ToX: the Toronto XML server," in *Proceedings of International Database Engineering and Applications Symposium (IDEAS '02)*, IEEE CS Press, Edmonton, Canada, July 2002.

[18] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, "Generalized search trees for database systems," in *Proceedings of 21st International Conference on Very Large Data Bases (VLDB '95)*, pp. 562–573, Zurich, Switzerland, September 1995.

[19] P. F. Dietz, "Maintaining order in a linked list," in *Proceedings of 14th Annual ACM Symposium on Theory of Computing (STOC '82)*, pp. 122–127, San Francisco, Calif, USA, May 1982.

[20] R. Goldman and J. Widom, “DataGuides: enabling query formulation and optimization in semistructured databases,” in *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB ’97)*, pp. 436–445, Athens, Greece, August 1997.
[21] R. Bayer and E. M. McCreight, “Organization and maintenance of large ordered indexes,” *Acta Informatica*, vol. 1, no. 3, pp. 173–189, 1972.
[22] B. C. Ooi and K.-L. Tan, “B-trees: bearing fruits of all kinds,” in *Proceedings of 13th Australasian Database Conference (ADC ’02)*, IEEE CS Press, Melbourne, Victoria, Australia, January–February 2002.
[23] J. Clark and S. DeRose, “XML path language (XPath), version 1.0,” Tech. Rep. REC-xpath-19991116, World Wide Web Consortium, New Delhi, India, W3C recommendation, November 1999.
[24] ISO/IEC 15938-1 Multimedia Content Description Interface Part 1: Systems, Geneva 2002.

Andrea Kofler-Vogt received the Dipl.-Ing. degree in computer science from the University of Klagenfurt, Austria, 2001, with a specialization on storage of multimedia content in database systems. At the moment she is a Ph.D. student supported by the University of Klagenfurt and the Corporate Technology of Siemens in Germany. She is researching in the field of metadata indexing.

Harald Kosch is an Associate Professor at the University of Klagenfurt. His domains of interest are distributed multimedia systems, multimedia databases, middleware, and Internet applications. He started research at the École Normale Supèrieure in 1993 during postgraduate study and entered the Ph.D. program in 1994, obtaining his Ph.D. degree in June 1997. He actively participates in the Moving Picture Experts Group MPEG-7 and MPEG-21 standardization and is involved in several international research projects in the domain of distributed multimedia systems.

Joerg Heuer received the Dipl.-Ing. degree in electrical engineering at the Friedrich-Alexander University of Erlangen, Germany, 1997, with a specialization on digital signal processing and high-frequency engineering. Supported by the Corporate Technology of Siemens, he received a Ph.D. degree from the Friedrich-Alexander University of Erlangen in 2003 for his work on multimedia content description. He joined the Corporate Technology of Siemens AG in 2002, where he is working as a Senior Scientist in the fields of multimedia content description and metadata coding in the domain of communication applications. Further research interests include multimedia adaptation in heterogeneous environments and indexing of metadata. He has been actively contributing to the MPEG standardization activity since 1999, in particular, on MPEG-7 and MPEG-21. He is a coeditor of Part 5, “Multimedia Description Schemes,” Part 8, “Extraction and Use of MPEG-7 Descriptions,” and Amendment 1 of Part 1, “Systems” of MPEG-7. Currently he is also contributing to the DVB CBMS standardization. He has also been active in several European research projects, including IST-SAMBITS, SAVANT, and ISIS.

André Kaup received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering from Aachen University of Technology (RWTH), Germany, in 1989 and 1995, respectively. From 1989 to 1995, he was with the Institute for Communication Engineering at Aachen University of Technology, where he was responsible for industrial as well as academic research projects in the areas of high-resolution printed image compression, object-based image analysis and coding, and models for human perception. In 1995 he joined Siemens Corporate Technology in Munich, where he chaired European research projects in the area of very low bit rate video coding, image quality enhancement, and mobile multimedia communications. In 1999 he was appointed Head of mobile applications and services, with research focussing on multimedia adaptation for heterogeneous communication networks. Since 2001 he is a Full Professor and Head of the Chair of Multimedia Communications and Signal Processing at University of Erlangen-Nuremberg. He is a Member of the German Informationstechnische Gesellschaft and a Senior Member of the IEEE. He was elected Siemens Inventor of the year 1998 and is the recipient of the 1999 ITG Award. From 1997 to 2001, he was also the Head of the German MPEG delegation and an Adjunct Professor at Technical University of Munich.