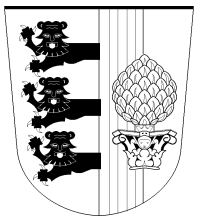
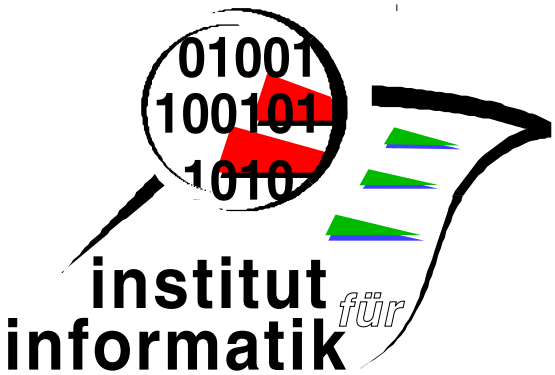
Universität Augsburg



Client-Server Optimization for Multimedia Document Exchange

Gerhard Köstler Wolfgang Kowarschick
Werner Kießling

Report 1996-3 September 1996



Institut für Informatik
D-86135 Augsburg

Client-Server Optimization for Multimedia Document Exchange

Gerhard Köstler[1] Wolfgang Kowarschick[2]
Werner Kießling[1]

[1] Institut für Informatik
Universität Augsburg
86135 Augsburg, Germany
{koestler, kiessling}@
informatik.uni-augsburg.de

[2] Institut für Informatik
Technische Universität München
80290 München, Germany
kowa@informatik.tu-muenchen.de

Abstract

The success of the World Wide Web is boosting the development of multimedia database systems and their integration into the internet. For the documents stored and exchanged in the Web there is a variety of multimedia data formats differing in aspects such as resolution, sampling rate, and compression. Furthermore there is a large heterogeneity of Web browsers, the data formats they support, and their network access. Thus storage servers have to store and proxy servers have to transfer many different formats.

However, the data formats are not independent from each other but interrelated by conversion tools. There is a large number of alternatives for storage and proxy servers to store some formats or to transfer them via the network and to compute the remaining ones by applying conversion tools. To determine an optimal choice is a nontrivial optimization problem and is subject to changes of the parameters such as query profile, available disk storage, and network bandwidth. We examine the outlined optimization problem in the context of object-oriented databases and illustrate our approach by a practical application.

1 Introduction

The integration of multimedia data such as images, audio, video, and full text is a significant trend of today's information system. This development was promoted by the phenomenal success of the internet and its most important application, the World Wide Web, bringing multimedia documents to one's fingertips. The

Web can be characterized as a worldwide collection of document servers delivering (multimedia) documents to the clients.

As the data volume accessible from the participating servers increases by an enormous rate, it becomes obvious to take advantage of the well-known virtues of database systems considering the efficient and secure storage and administration of huge amounts of data . Consequently, multimedia database systems and the integration into the Web are hot topics in database theory and system development [KB96, SSe96].

The major drawback of the current World Wide Web is its often complained lack of bandwidth (the "World Wide Wait"). The caching of documents by proxy servers and establishing hierarchies of such servers can only mitigate this nuisance by reducing wide area network accesses. On the other hand, the information exchanged and stored within the Web is often redundant. There exists a variety of multimedia data formats which are interrelated by conversion tools and may differ in aspects such as compression (none, lossless, or lossy), color depth and resolution for images, or the sampling rate for audio. The set of multimedia formats supported by the particular browsers varies as well (the Web transport protocol draft HTTP/1.1 [FGM+96] even optionally offers content negotiation: clients can specify their preferences). The consequence is a redundant storage of the multimedia documents at the storage server and a non-optimal exchange of them.

This paper proposes to integrate the conversion tools into storage and proxy servers aiming at a dynamic optimization of multimedia document exchange. Formally this can be done by representing these tools as functional constraints on the attributes of an object-oriented schema design. This gives the option either to physically represent an attribute (by storing it or accessing it via the network) or to compute it from other ones. This approach both reduces the data volume and relieves the networks. Given a set of conversion tools for a *particular* server there is, in general, more than one way of partitioning the formats into physically represented and computed ones. By applying a cost function each server locally determines its optimal choice considering aspects such as storage consumption, network bandwidth, and complexity of the conversions. We will demonstrate that this optimization is a non-trivial task that cannot be done manually and must be performed periodically since the parameters of the cost function may change depending on the access profile (e.g. the server can use statistical data for this purpose).

The rest of this paper is organized as follows: Section 2 illustrates our approach by a practical application, a retrieval and delivery system for journal articles, and presents an object-oriented database design for it. In Section 3 we abstract from the application and formally describe the optimization problem for object-oriented databases with functional constraints. Section 4 discusses search space reductions for the optimization problem and Section 5 examines a cost function

by means of several scenarios. Section 6 summarizes the contributions of this paper and sketches possible extensions.

2 A Sample Application

We will illustrate our approach by the document exchange subsystem of the digital library project ELEKTRA. The ELEKTRA project aims at the retrieval and delivery of digitized journal articles. The retrieval of the papers is based on the bibliographic data as well as on the full text of the tables of contents and the first pages of the articles. Query results are raster image representations of the first pages of the selected papers and their bibliographic data. Authorized users can order the raster images of complete articles which are delivered both in a browsable and in a printable format.

Technically the ELEKTRA system is based on the distributed multimedia database system TransBase/Myriad [Tra96]. For copyright reasons each participating library operates two strictly separated databases: one for the retrieval containing the bibliographic data and the first pages of all articles and the other one for order and delivery. Papers are included into the delivery DB on demand only, i.e., not yet available articles are scanned by an operator. Standard Web browsers are employed as user interface to the system and specialized proxy servers which perform caching and format conversions are used (the proxy servers are separate also due to copyright restrictions).

In the ELEKTRA project browsable and printable representations of a document are distinguished for the following reasons:

- Resolutions higher than 150 dpi are not appropriate for today's standard displays. For zooming into a page ELEKTRA additionally offers 300 dpi as resolution.
- Printing, however, should be done with at least 300 dpi ([TUL] strongly advocates this claim).
- The built-in printing facilities of today's Web browsers are inappropriate. They do not optimize the conversion of raster images to a print format what can result in print jobs of 100 MB and more for a twenty page paper. Scaling of the images to the correct printer page size is another critical issue not handled very well.

In general, journal articles contain greyscale or even color illustrations. For such pages we decided to offer two raster images: a black-and-white (B/W) image and a color image. The B/W image is for reading the text; a lossless graphic format is used for this. The color image is for viewing the illustrations; a lossy graphic format is used for that. To satisfy the needs of all the different clients

	text pages	illustration pages
browsing	GIF	GIF
(150 dpi)	PNG	PNG
		JPG
zooming	GIF	GIF
(300 dpi)	PNG	PNG
		JPG
printing	PS	PS
(300 dpi)	PS_LZW	PS_DCT
	PDF	PDF
	PDF_LZW	PDF_DCT
(faxing)	G4	

Table 1: Formats supported in ELEKTRA

the raster image and printing formats summarized in Table 1 are supported by the ELEKTRA system. PNG (Portable Network Graphics) [PNG] was developed as a successor of the popular GIF format. G4 (International Telecommunications Union (ITU; formerly known as CCITT) Fax Group IV encoding scheme) is used for faxing B/W pages as a special case of printing. The raster image components of PostScript (Level 2 and higher [Pos90]) or PDF files can be compressed. The internal interpreter of the printer will then perform the decompression. For B/W images the lossless Lempel-Zif-Welch (LZW) algorithm and for greyscale and color images the lossy discrete cosine transform (DCT) technique, based on the JPEG standard, are employed.

A simplified object-oriented database design [BM93, Cat94] of the ELEKTRA system is depicted in Figure 1 (using the ODMG-93 notation [Cat94]). We neglect the methods except those to access the image formats of pages. Journals, journal numbers, articles, and pages are modeled by the classes `journal`, `journal number`, `article`, and `page`. Journals consist of journal numbers and numbers contain articles. This is formalized by a 1:n relationships between `journal` and `journal number` and between `journal` and `article`. Articles are composed of pages and the numbers themselves contain own pages such as the table of contents. Again this is expressed by 1:n relationships. The image representations of pages are modeled as objects of the abstract class `image` with the disjoint subclasses `text image` and `illustration image`. As discussed above, a page may have one or two image representations. This is represented by a 1:n relationship between the classes `page` and `image`. One image representation is for reading and the other one is for viewing the illustrations. They are formalized as objects of classes `text image` and `illustration image` (modeling multimedia data as objects is also done by [Mas91]). In the sequel we will concentrate on the taxonomy of the class `image` and its subclasses `text image` and `illustration image`.

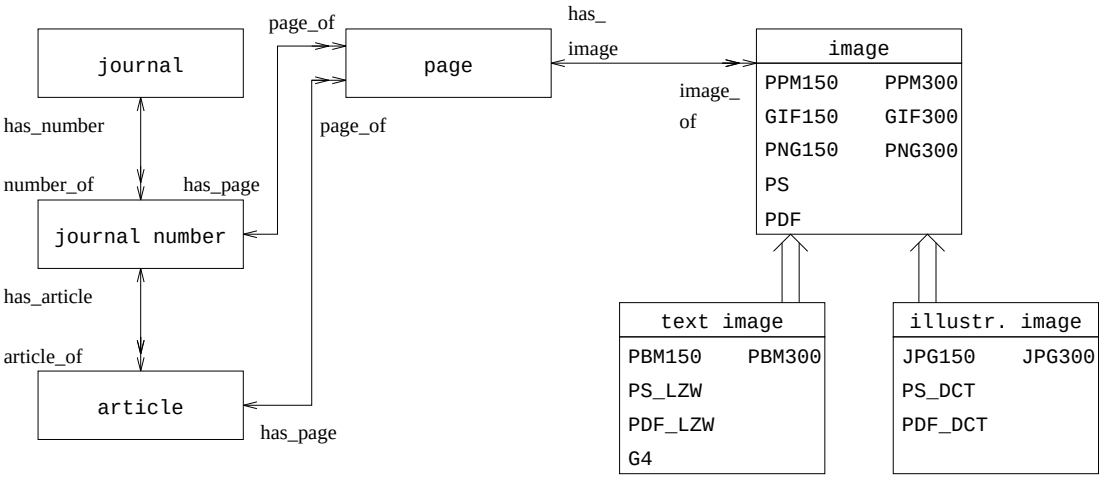


Figure 1: The schema design of ELEKTRA

The graphic formats supported in the ELEKTRA system are modeled as attributes of those classes such are `PPM300` and `PS`. Their types are (typed) BLOBS. Following the ODMG standard [Cat94, Sec. 2.4.1], we assume that attributes are accessed via methods such as `get_value` and `set_value` only (nevertheless an attribute may be referenced by its name; this can be realized by a simple preprocessor). This is also in the spirit of [Mey88] who does not distinguish attribute from methods at all denoting both of them as features). The exclusive access of attributes through methods enhances the physical data independence of the model. The decision whether to physically represent an attribute or to compute it is transparent to the application programs; this choice can be subject to changes. Additional to the formats of Table 1 we introduce attributes `PPM300`, `PPM150`, `PBM300`, and `PBM150` for the Portable Pixmap and Bitmap formats of Jef Poskanzer at the resolutions 150 dpi and 300 dpi. These attributes are used for format conversions.

The supported graphic formats are strongly interrelated by tools such as the NetPPM suite by Jef Poskanzer and other ones. For example a 300 dpi `GIF` image can be converted to 150 dpi. Furthermore, there are tools for transforming `GIF`, `PNG`, and `JPG` into each other, and to translate each of them into PostScript, `PDF`, or `G4`. We model those dependencies amongst the attributes as *functional constraints*. Like attributes, functional constraints are inherited by subclasses (see [KKK96]). Table 2 lists a set of constraints that are typically available. However, as the storage servers and the proxy servers operate in a heterogeneous environment *the set of available conversion tools (and consequently the set of functional constraints) is individual for each particular server.*

The optimization problem that arises from this database design of the ELEKTRA system can be stated as follows. Given a set of functional constraints on the attributes of a particular ELEKTRA server (storage or proxy), *what is an optimal*

image

browsing and zooming:
GIF150 = ppm2gif(PPM150)
PNG150 = ppm2png(PPM150)
PPM150 = gif2ppm(GIF150)
PPM150 = png2ppm(PNG150)
GIF300 = ppm2gif(PPM300)
PNG300 = ppm2png(PPM300)
PPM300 = gif2ppm(GIF300)
PPM300 = png2ppm(PNG300)
GIF150 = gifreduce(GIF300)
PNG150 = pngreduce(PNG300)
printing:
PS = pdf2ps(PDF)
PDF = ps2pdf(PS)
PS = ppm2ps(PPM300)
PDF = ppm2pdf(PPM300)
PS = png2ps(PNG300)
PDF = png2pdf(PNG300)

text image

browsing and zooming:
PPM300=pbm2ppm(PBM300)
PPM150=pbm2ppm(PBM150)
printing:
PS_LZW = LZWEncode(PS)
PDF_LZW = LZWEncode(PDF)
PS = LZWDecode(PS_LZW)
PDF = LZWDecode(PDF_LZW)
faxing:
G4 = ppm2g4(PBM300)
G4 = png2g4(PNG300)

illustration image

browsing and zooming:
JPG300 = png2jpg(PNG300)
PNG300 = jpg2png(JPG300)
JPG150 = png2jpg(PNG150)
PNG150 = jpg2png(JPG150)
printing:
PS_DCT = DCTEncode(PS)
PDF_DCT = DCTEncode(PDF)
PS = DCTDecode(PS_DCT)
PDF = DCTDecode(PDF_DCT)

Table 2: Functional constraints induced by conversion tools

partition of attributes into those that are physically represented and the attributes computed from the physically represented ones? Let us illustrate the optimization problem by the following case study. More elaborate scenarios are discussed in Section 5.

Example 1

Assume that a storage server receives many requests for text images in the `G4` fax format (i.e., many users apply their fax machines as output device). We observed that the conversion from graphic formats such as `PBM` to `G4` is very expensive (We measured 235 seconds on a Intel 486/DX4 with 48 MB main storage under Linux 1.99.4 for an average image). Depending on the cost function, this computation costs can outweigh the storage costs (130 kByte for an average image) of the `G4` format and the optimization process on the server decides to additionally store this format. However, as the number of users requesting the fax format decreases the additional storage costs for `G4` outweigh the computation costs and a new optimization decides to compute `G4` instead of storing it.

As the optimization will depend on statistical data collected by the servers and is repeated periodically, this kind of optimization should be done automatically.

3 The Optimization Problem

The ELEKTRA system is only one instance of the following more general setting: Let the attributes $\mathcal{A}$ of an object-oriented schema be interrelated by functional constraints. The optimization problem treated in this paper is how to determine an optimal partition of those attributes into physically represented $\mathcal{A}_{phys}$ and computed ones $\mathcal{A}_{comp}$.

In [KKK96] we argue that functional constraints between attributes are not an exceptional case. On the contrary, they evolve very naturally when ordering the classes of an application taxonomically, i.e., according to their natural IS-A relationship (note that in Figure 1 the class `image` and its subclasses form such a taxonomy).

In the following we will formalize the notions and the optimization problem. Attributes model the properties of the instances of a class and functional constraints are dependencies between these properties.

Definition 2 (Functional Constraints)

Let A be an attribute, X a set of attributes, and f a function. Then $fc \equiv (A = f(X))$ is called *functional constraint*. A and X are denoted as *head* (*head*(fc)) and *body attributes* (*body*(fc)).

Let $\mathcal{F}$ be a set of functional constraints. Then an attribute A *depends directly* on an attribute B iff there is a $fc \equiv (A = f(X)) \in \mathcal{F}$ such that $B \in X$. The transitive closure, $\leftarrow$, of this relation is denoted as *dependency relation.*

$\mathcal{F}$ is named *acyclic* if no attribute depends on itself.

Let $\mathcal{F}$ be acyclic. Then the *compositions* $\overline{\mathcal{F}}$ of $\mathcal{F}$ are defined as follows:

1. $\mathcal{F} \subseteq \overline{\mathcal{F}}$.
2. If $(A = f(A_1, \ldots, A_n)) \in \mathcal{F}$ and $(A_i = g(B_1, \ldots, B_m)) \in \overline{\mathcal{F}}$, then $(A = f(A_1, \ldots, A_{i-1}, g(B_1, \ldots, B_m), A_{i+1}, \ldots, A_n)) \in \overline{\mathcal{F}}$.

$\overline{\mathcal{F}}$ contains all the compositions of functional constraints from $\mathcal{F}$. The definitions of head or body attributes for such compositions is obvious. The composition of $(A = f(A_1, \ldots, A_n))$ and $(A_i = g(B_1, \ldots, B_m))$ is itself a functional constraint $(A = h(A_1, \ldots, A_{i-1}, B_1, \ldots, B_m, A_{i+1}, \ldots, A_n))$.
In the sequel let $\mathcal{A}$ be a set of attributes of one class of the database model and $\mathcal{F}$ be the set of constraints on $\mathcal{A}$. A functional base of $\mathcal{F}$ is a subset partitioning the attributes such that some are physically represented and the rest can uniquely be computed by compositions of constraints.

Definition 3 (Functional Base)
Let $\mathcal{F}_{comp}$ be a subset of $\mathcal{F}$. $\mathcal{F}_{comp}$ is called a *functional base* iff:

1. $\mathcal{F}_{comp}$ is acyclic.
2. For every $A \in \mathcal{A}$ there is at most one $fc \in \mathcal{F}_{comp}$ such that A is the head of fc.

$\mathcal{A}_{comp}$ is the set of head attributes of $\mathcal{F}_{comp}$, and $\mathcal{A}_{phys} := \mathcal{A} \setminus \mathcal{A}_{comp}$ are the remaining attributes.

The attributes $\mathcal{A}_{phys}$ must be physically represented (either be stored or accessed via the network) and the attributes $\mathcal{A}_{comp}$ are computed from attributes in $\mathcal{A}_{phys}$.

Lemma 4
Let $\mathcal{F}_{comp}$ be acyclic. Then $\mathcal{F}_{comp}$ is a functional base iff for every $A \in \mathcal{A}_{comp}$ there is a unique composition $fc_A \in \overline{\mathcal{F}}_{comp}$ such that $A = head(fc_A)$ and $body(fc) \subseteq \mathcal{A}_{phys}$.

Proof.
$\Longrightarrow$ $\mathcal{F}_{comp}$ is acyclic. Therefore $\leftarrow$ is a noetherian partial ordering on $\mathcal{A}$. Let $A \in \mathcal{A}_{comp}$. Since $\mathcal{F}_{comp}$ is a functional base there is a unique functional constraint $fc \equiv A = f(X) \in \mathcal{F}_{comp}$. Proof by noetherian ordering over $\leftarrow$.

Let $\{A_1, \ldots, A_n\} := X \cap \mathcal{A}_{comp}$ and $Y := X \cap \mathcal{A}_{phys}$. By inductive hypothesis there are unique $A_i = f_i(X_i) \in \overline{\mathcal{F}}_{comp}$ such that $X_i \subseteq \mathcal{A}_{phys}$. Then $A = f(f_1(X_1), \ldots, f_n(X_n), Y) \in \overline{\mathcal{F}}_{comp}$ is unique as well.

$\Longleftarrow$ Let $\mathcal{F}_{comp}$ be acyclic and $A \in \mathcal{A}_{comp}$. Then there is a unique $fc \in \overline{\mathcal{F}}_{comp}$ such that $body(fc) \subseteq \mathcal{A}_{phys}$. If $fc \in \mathcal{F}_{comp}$ there is nothing to show. Otherwise let $A = head(fc)$. Then by definition of $\overline{\mathcal{F}}_{comp}$ we get $fc \equiv A = f(A_1, \ldots, f_i(X_i), \ldots, A_n)$ such that $A_j \in \mathcal{A}_{phys}$, for $j \neq i$, $A_i \in \mathcal{A}_{comp}$, and $X_i \subseteq \mathcal{A}_{phys}$ and $A = f(A_1, \ldots, A_n) \in \mathcal{F}_{comp}$. $\square$

According to Lemma 4 only functional bases uniquely determine which attributes are physically represented and how the other ones are computed.

Example 5

One possible base $\mathcal{F}_{comp}$ of the `illustration image` is

PPM300 = png2ppm(PNG300)
GIF300 = ppm2gif(PPM300)
GIF150 = gifreduce(GIF300)
PNG150 = pngreduce(PNG300)
PPM150 = gif2ppm(GIF150)
JPG300 = png2jpg(PNG300)
JPG150 = png2jpg(PNG150)
PS = ppm2ps(PPM300)
PDF = ps2pdf(PS)
PS_DCT = DCTEncode(PS)
PDF_DCT = DCTEncode(PDF)

The set $\mathcal{A}_{phys}$ of physically represented attributes contains `PNG300` only, the set $\mathcal{A}_{comp}$ of computed attributes contains all attributes occurring in the heads of $\mathcal{F}_{comp}$.
`GIF150` = `gifreduce(ppm2gif(png2ppm(PNG300)))` is the unique composition computing `GIF150`. On the system configuration described above we measured about 30 seconds for this computation.

The optimization problem described above can formally be stated as follows: *Determine a functional base optimal w.r.t. a cost function.*

Definition 6 (Optimal functional base)

Let *cost* be a cost function. A functional base $\mathcal{F}_{opt}$ is called *optimal* iff

$cost(\mathcal{F}_{opt}) =$
$\min\{cost(\mathcal{F}_{comp}) \mid \mathcal{F}_{comp}$ is a functional base$\}$.

Note that at least one optimal functional base exists since the empty set is always a functional base and the number of functional bases is finite. In Section 5 we discuss reasonable cost functions.

According to Lemma 4 only sets $\mathcal{F}_{comp}$ that contain at most one functional constraint $A = f(X)$ for every attribute $A \in \mathcal{A}$ must be tested for cycles. If $\mathcal{F}_{comp}$ is acyclic, then it is a functional base and the cost function *cost* is applied.

Algorithm 7 (Basic Optimization)
Let $\mathcal{F}_A \in \mathcal{F}$ be the set of constraints with head attribute A. Then the algorithm depicted in Figure 2 determines an optimal functional base $\mathcal{F}_{opt}$ of $\mathcal{F}$.

```
OFB(𝒜, ℱ, cost):  (ℱ_opt, min)
 begin
   min := cost(∅);
   ℱ_opt := ∅;
   for ℱ_comp ⊆ ℱ where
     ∀A ∈ 𝒜: |ℱ_comp ∩ ℱ_A| ≤ 1
   do
     if ℱ_comp is acyclic and
        cost(ℱ_comp) < min
     then
       min := cost(ℱ_comp);
       ℱ_opt := ℱ_comp;
     fi
   done
 end
```

Figure 2: Optimization Algorithm

The complexity of this basic optimization algorithm depends on the cardinalities of the functional base and the set of attributes.

Lemma 8 (Complexity of Basic Optimization)
Let $m := |\mathcal{F}|$, $n := |\mathcal{A}|$, and $m_A := |\mathcal{F}_A|$ for $A \in \mathcal{A}$.

1. *There are $\Pi_{A \in \mathcal{A}}(m_A + 1)$ sets $\mathcal{F}_{comp}$ that are tested for cycles and $(\frac{m}{n} + 1)^n$ is an upper bound to this number.*

2. *Testing for cycles has a worst case complexity of $O(m \cdot n + n^2)$.*

3. *So the worst case complexity of the basic optimization algorithm is $O((m \cdot n + n^2) \cdot (\frac{m}{n} + 1)^n)$*
   (note that the parameters m and n are in general independent).

4. *If $m = c \cdot n$ (there is a maximal number c of constraints per attribute) then the worst case complexity is $O(n^2 \cdot (c+1)^n)$.*

5. *If $m = n^2$ (every attribute depends on all other ones) and there is a maximum number d of body attributes then the worst case complexity is $O(n^2 \cdot (n+1)^n)$.*

Proof.

1. The product $\Pi_{A \in \mathcal{A}}(m_A + 1)$ under condition $m = \sum_{A \in \mathcal{A}} m_A$ is an $n-1$-dimensional, two times continuously differentiable function. The gradient equals zero, iff $m_A = m_B$, for all $A, B \in \mathcal{A}$. By induction it can be proved that the negative Hessian is positive definite. Thus, $(\frac{m}{n} + 1)^n$ is a global maximum of $\Pi_{A \in \mathcal{A}}(m_A + 1)$ under condition $m = \sum_{A \in \mathcal{A}} m_A$.

2. Testing for cycles can be done by depth-first search (with adjacency matrix). Constructing this matrix costs $O(m \cdot n)$, for m functional constraints each of which defines at most n edges between the nodes $\mathcal{A}$. The depth-first search, itself, with adjacency matrix costs $O(n^2)$ [Sed88].

3. An immediate consequence of 1 and 2.

4. An immediate consequence of 3.

5. An immediate consequence of 3.

□

We summarize that in reasonable cases (such as 4 and 5), the complexity of the basic optimization algorithm is exponential or even hyper-exponential. In Section 4 we will discuss strategies to reduce this search space.

Example 9

For our sample application we have to test $k := \Pi_{A \in \mathcal{A}}(m_A + 1)$ sets for cycles. We observe that these numbers are remarkably better than the worst case $(\frac{m}{n} + 1)^n$.

class	m	n	k	$(\frac{m}{n} + 1)^n$
`image`	16	8	5184	6561
`text image`	24	13	172800	804190
`illust. image`	24	12	259200	531441

4 Search Space Reductions

The exponential or even hyper-exponential complexity of the basic optimization algorithm seems prohibitive for large database designs. In this section we therefore examine possible search space reductions.

One of the most promising possibilities of reducing the search space is to partition the set of functional constraints into simply connected components. W.l.o.g. we assume that $\mathcal{A}$ does not contain attributes which do not occur in any constraint. Those attributes must be physically represented and do not take part in the optimization.

Definition and Lemma 10 (Components)
Let $fc_1, fc_2 \in \mathcal{F}$. Then transitive closure of the binary relation

$$fc_1 \sim fc_2 : \Longleftrightarrow \ (head(fc_1) \cup body(fc_1)) \cap (head(fc_2) \cup body(fc_2)) \neq \emptyset$$

is an equivalence relation on $\mathcal{F}$. This relation partitions $\mathcal{F}$ into the *(simply connected) components* $\mathcal{F} = \mathcal{F}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{F}_k$. Let $\mathcal{A}_i := head(\mathcal{F}_i) \cup body(\mathcal{F}_i)$. Then $\mathcal{A} = \mathcal{A}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{A}_k$ is a partition of $\mathcal{A}$.

Proof. Immediately from the definition. □

Example 11
The sets of functional constraints depicted in Table 2 consist of one component only, viz. raster image formats. However, as more and more electronic journals are published and as these journals may also contain video and audio sequences, it will be necessary to integrate this kind of data into the ELEKTRA system. The audio formats and the video formats are strongly interrelated as well but operate on disjoint attribute sets. Thus the functional constraints of the image, audio, and video formats form three components.

Partitioning of functional constraints can be used to compose functional bases of the entire set from functional bases of the components.

Lemma 12
Let $\mathcal{F} := \mathcal{F}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{F}_k$ *be the components of* $\mathcal{F}$ *and* $\mathcal{A} := \mathcal{A}_1 \dot{\cup} \ldots \dot{\cup} \mathcal{A}_k$ *be the corresponding partition of the attributes. Moreover let* $\mathcal{F}_{comp} \subseteq \mathcal{F}$.
Then $\mathcal{F}_{comp}$ *is a functional base of* $\mathcal{F}$ *iff* $\mathcal{F}_{comp_i} := \mathcal{F}_{comp} \cap \mathcal{F}_i$ *is a functional base of* $\mathcal{F}_i$, $1 \leq i \leq k$.

Proof.
$\Longrightarrow$ Let $\mathcal{F}_{comp}$ be a functional base of $\mathcal{F}$. Then $\mathcal{F}_{comp}$ is acyclic on $\mathcal{A}$, i.e., no attribute $A \in \mathcal{A}$ (and therefore $A \in \mathcal{A}_i$) depends on itself. Removing constraints does not add cycles. Thus $\mathcal{F}_{comp_i}$ is acyclic on $\mathcal{A}_i$. $\mathcal{F}_{comp}$ contains at most one constraint $A = f(X)$ for each $A \in \mathcal{A}$. Thus $\mathcal{F}_{comp_i}$ contains at most one constraint $A = f(X)$ for each $A \in \mathcal{A}_i$ and so $\mathcal{F}_{comp_i}$ is a functional base as well.
$\Longleftarrow$ Let $\mathcal{F}_{comp_i}$ be a functional base of $\mathcal{F}_i$, $1 \leq i \leq k$. Then $\mathcal{F}_{comp_i}$ is acyclic on $\mathcal{A}_i$, i.e., no attribute $A \in \mathcal{A}_i$ depends on itself. Let $fc \in \mathcal{F}_{comp_j}$, $j \neq i$, be a constraint. Since the attributes of fc do not occur in $\mathcal{A}_i$, $\mathcal{F}_i \cup \{fc\}$ is acyclic on

$\mathcal{A}_i \cup head(fc) \cup body(fc)$. Thus $\mathcal{F}_{comp}$ is acyclic on $\mathcal{A}$. $\mathcal{F}_{comp_i}$ contains at most one constraint $A = f(X)$ for every $A \in \mathcal{A}_i$. Since the head attributes of constraints $fc \in \mathcal{F}_{comp_j}$, $j \neq i$, do not occur in $\mathcal{A}_i$, $\mathcal{F}_{comp}$ contains at most one constraint $A = f(X)$ for each $A \in \mathcal{A}$ as well. □

In the sequel we concentrate on a reasonable class of cost functions.

Definition 13 (Additive cost functions)
A cost function *cost* is called *additive* iff $cost(\mathcal{F}_{comp}) = \sum_{1 \leq i \leq k} cost(\mathcal{F}_{comp_i})$.

Optimal bases for additive cost functions can be determined from optimal bases of the components.

Lemma 14
Let $\mathcal{F}_{opt_i}$ be optimal functional bases of $\mathcal{F}_i$ and let cost be additive. Then $\mathcal{F}_{opt} := \bigcup_{1 \leq i \leq k} \mathcal{F}_{opt_i}$ is an optimal functional base of $\mathcal{F}$.

Proof. An immediate consequence of Lemma 12 and Definition 13. □

Partitioning $\mathcal{F}$ into components in many cases reduces the worst case complexity of the optimization tremendously. Partitioning itself can be done in almost linear time (using the union-find algorithm, see e.g. [Sed88]); Algorithm 7 is then applied to the smaller components.

Example 15
Imagine that the class `text image` contains 6 attributes for audio data interrelated by 10 constraints. Then in the worst case we have to test 291,691,050 sets for cycles. After partitioning the constraints into two components, the audio formats and the rest, we have to test 804,190 + 340 = 804,530 sets.

Further reductions such as the treatment of constant constraints $A = \text{const}$ and equalities $A = B$ are examined in [Ste96]. This kind of constraints arise, e.g., in CAD applications (see [KKK96]).
All these search space reduction guarantee the optimality of the selected functional base. In many cases, however, it may be sufficient to find a good suboptimal solution (cmp. query optimization in database systems). This could be accomplished by performing a heuristic search.

5 Cost Functions

The quality of the optimization result highly depends on an appropriate cost function. In this section we discuss a rather simple sample cost function that is modeling crucial aspects of the ELEKTRA system as illustrated by several

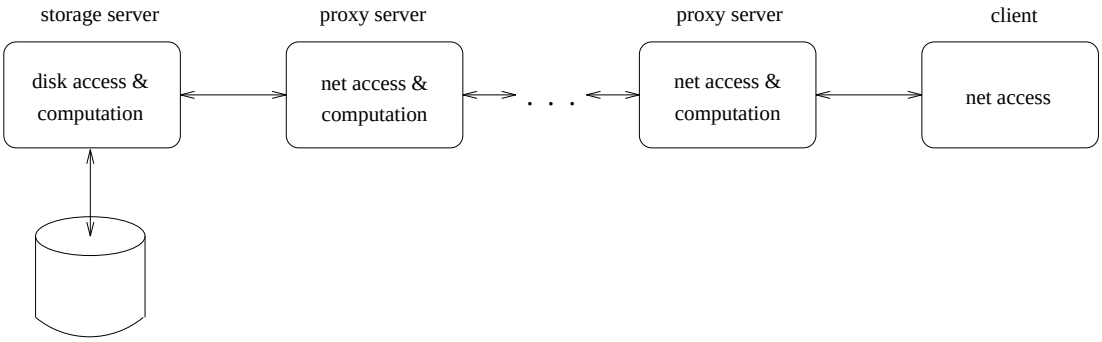


Figure 3: Hierarchy of multimedia document exchange

scenarios. We think that the function is appropriate for many other applications as well.

As depicted in Figure 3 in the ELEKTRA project two types of servers are used: storage and proxy servers. The storage servers physically store documents and deliver attributes of them on demand to proxy servers. Proxy servers receive document attribute requests from clients, accesses attributes from storage servers via the network, and deliver them to the clients. The network access is accelerated by the cache storage of the proxy servers. As a result of the optimization process described in the last section both storage and proxy servers can compute document attributes instead of storing or accessing them via the network.

Thus an appropriate cost function has to consider the following:

Proxy servers

1. Access costs *acc_cost* (i.e., attribute access or attribute computation)

Storage servers

1. Access costs *acc_cost* (as above)
2. Storage costs *store_cost*

Note that for proxy servers no costs arise, if no documents are accessed. Storage servers, however, have to store at least some attributes of the documents causing storage costs. The described costs are modeled by the following cost function.

Definition 16 (Cost function)
Let $\mathcal{F}_{comp}$ be a functional base and $\mathcal{A}_{comp}$ and $\mathcal{A}_{phys}$ be the sets of computed and physically represented attributes, respectively. Then the cost function for proxy servers is defined as

$$cost(\mathcal{F}_{comp}) := acc\_cost(\mathcal{F}_{comp}) \ .$$

The cost function for storage servers is defined as

$$cost(\mathcal{F}_{comp}) := \begin{array}{rcl} (1-z) & \cdot & store\_cost(\mathcal{F}_{comp}) + \\ z & \cdot & acc\_cost(\mathcal{F}_{comp}) \ . \end{array}$$

By adjusting the parameter $z \in [0,1]$ the system administrator influences the overall behavior of a storage server. For low values of z the optimization process will try to minimize the storage costs even if this increases the attribute access costs. At the other extreme, having abundant disk storage the administrator may decide to optimize attribute access instead, choosing a z value of almost one. In the ELEKTRA project a rather low value of z seems appropriate as we estimate that the data volume will increase by at least 50 CD-ROMs[1] a year just for the institution the second author is affiliated with. With the new DVD standard for CD-ROMs available, z probably will be increased.

The storage costs *store_cost* are modeled as being proportional (with factor c_S) to the sum of average sizes $avg\_size(A)$ of the attributes $A \in \mathcal{A}_{phys}$.

Definition 17 (Storage costs)
The storage costs are defined as
$store\_cost(\mathcal{F}_{comp}) := c_S \cdot \sum_{A \in \mathcal{A}_{phys}} avg\_size(A)$.

The access costs *acc_cost* of $\mathcal{F}_{comp}$ are modeled by the sum of the access costs for the particular attributes weighed by the average access rates *avg_acc* to these attributes.

Definition 18 (Access costs)
The access costs are defined as
$acc\_cost(\mathcal{F}_{comp}) := \sum_{A \in \mathcal{A}} avg\_acc(A) \cdot acc\_cost(A)$.

The access costs to attributes differ for physically represented and computed attributes. In the physical case we assume the costs to be proportional to the average size. For a computed attribute we have to consider the access costs of attributes it is computed from and the computation costs themselves.

Definition 19 (Attribute access costs)
The attribute access costs are defined as

$$acc\_cost(A) := \begin{cases} c_A \cdot avg\_size(A) & \text{if } A \in \mathcal{A}_{phys} \\ comp\_cost(f, \mathcal{A}_{comp}) + & \\ \sum_{B \in X} acc\_cost(B) & \text{if } A \in \mathcal{A}_{comp} \end{cases}$$

where $A = f(X) \in \overline{\mathcal{F}}_{comp}$ is the unique composition computing $A \in \mathcal{A}_{comp}$ from $X \subseteq \mathcal{A}_{phys}$.

[1] Assuming 30,000 new articles a year and 1 MByte per article

Choosing the parameter c_A depends on many factors. For example for storage servers it depends on the used mass storage devices, whereas for proxy servers the network bandwidth (modem or T-1), the cache hit rate, hence the cache size are relevant. The computation costs for a combination of conversion tools f are influenced by the workload on the host. Computing more attributes obviously increases this workload.

Definition 20 (Computation costs)
The computation costs are defined as a function $comp\_cost(f, \mathcal{A}_{comp})$ with the following properties:

1. $comp\_cost(f, \mathcal{A}_{comp})$ has a lower bound (the costs of applying f to an average set of attributes on an idle host).

2. If $\sum_{A \in \mathcal{A}_{comp}'} avg\_acc(A) \leq \sum_{A \in \mathcal{A}_{comp}''} avg\_acc(A)$, then $comp\_cost(f, \mathcal{A}_{comp}') \leq comp\_cost(f, \mathcal{A}_{comp}'')$.

   That means that $comp\_cost$ is monotonic in the number of computations.

3. Let $\mathcal{F}_i$ be the component of f and $\mathcal{A}_i$ be the corresponding attribute set. Then $comp\_cost(f, \mathcal{A}_{comp}) = comp\_cost(f, \mathcal{A}_{comp} \cap \mathcal{A}_i)$.

The last property may appear artificial. It says that only the computation costs of attributes within the same component are taken into account. However, this restriction guarantees with Definition 19 the additivity of the cost function and we consider it as a justified approximation.
The exact shape of $comp\_cost$ depends on the particular host and must be determined by experiments. The parameters $avg\_size(A)$, $avg\_acc(A)$ can be determined by statistical data collected by the servers and evolve over the time.
In the sequel we discuss the proposed cost function by several scenarios.

Example 21
We discuss the optimization of a proxy server on three stages.
Stage 1: The server has many clients still operating Level-1 PostScript printers without built-in decompression and some clients operating Level-2 PostScript printers. Thus it receives about seven time as many requests for the `PS` format than for `PS_LZW`. It is connected to a storage server via a 64 kBit network connection yielding a 4 kByte average data transfer rate.
Stage 2: The proxy server gets connected to a new 10 MBit local network yielding a 100 kByte average data transfer rate. According to the improved network access the number of requests increases.
Stage 3: As time goes by, the old Level-1 PostScript printers are replaced by new Level-2 ones. The LAN average data transfer rate decreases to 50 kByte as the network is more occupied.

In the sequel we concentrate on the PS and PS_LZW attributes. There are three alternatives for the proxy server to answer requests for this data formats:

- Accessing both via the network. Formally
  $\mathcal{A}_{comp_1} = \mathcal{F}_{comp_1} = \emptyset$.
- Accessing PS_LZW via the network and computing PS from it by decompression. Formally
  $\mathcal{A}_{comp_2} = \{\texttt{PS}\}$,
  $\mathcal{F}_{comp_2} = \{\texttt{PS} = \texttt{LZWDecode}(\texttt{PS\_LZW})\} =: \{\texttt{PS} = \texttt{f}_2(\texttt{PS\_LZW})\}$.
- Accessing PS via the network and computing PS_LZW from it by compression. Formally
  $\mathcal{A}_{comp_3} = \{\texttt{PS\_LZW}\}$,
  $\mathcal{F}_{comp_3} = \{\texttt{PS\_LZW} = \texttt{LZWEncode}(\texttt{PS})\} =: \{\texttt{PS\_LZW} = \texttt{f}_3(\texttt{PS})\}$.

Let the average sizes be $avg\_size(\texttt{PS}) = 1300kByte$ and $avg\_size(\texttt{PS\_LZW}) = 380kByte$. Then the costs for the three alternatives are

$$
\begin{array}{l}
cost(\mathcal{F}_{comp_1}) = \\
\quad avg\_acc(\texttt{PS}) \cdot c_A \cdot 1300kByte + \\
\quad avg\_acc(\texttt{PS\_LZW}) \cdot c_A \cdot 380kByte
\end{array}
$$

$$
\begin{array}{l}
cost(\mathcal{F}_{comp_2}) = \\
\quad avg\_acc(\texttt{PS}) \cdot (comp\_cost(f_2, \mathcal{A}_{comp_2}) + c_A \cdot 380kByte) + \\
\quad avg\_acc(\texttt{PS\_LZW}) \cdot c_A \cdot 380kByte
\end{array}
$$

$$
\begin{array}{l}
cost(\mathcal{F}_{comp_3}) = \\
\quad avg\_acc(\texttt{PS}) \cdot c_A \cdot 1300kByte + \\
\quad avg\_acc(\texttt{PS\_LZW}) \cdot (comp\_cost(f_3, \mathcal{A}_{comp_3}) + c_A \cdot 1300kByte)
\end{array}
$$

Assume that on Stage 2 the proxy server receives 30 times as many requests than on Stage 1 and that on Stage 3 the ratio of PS and PS_LZW requests is reverted. Let access parameter c_A be the reverse of the average data access rate. Then the parameters $avg\_acc(\texttt{PS})$, $avg\_acc(\texttt{PS\_LZW})$ and c_A are

	PS	PS_LZW	c_A
Stage 1	7	1	0.25/kByte
Stage 2	210	30	0.01/kByte
Stage 3	30	210	0.02/kByte

The computation costs grow with the workload. For the functional base $\mathcal{F}_{comp_2}$ the workload in the average increases from 7 to 210 executions of LZWDecode (per time unit) from Stage 1 to Stage 2 and decreases to 30 from Stage 2 to Stage 3. For the functional base $\mathcal{F}_{comp_3}$ the workload in the average increases from 1 to 30 executions of LZWEncode (per time unit) from Stage 1 to Stage 2 and increases

to 210 from Stage 2 to Stage 3. This is reflected by the following computation costs *comp_cost*.

	$f_2, \mathcal{A}_{comp_2}$	$f_3, \mathcal{A}_{comp_3}$
Stage 1	13	13
Stage 2	17	14
Stage 3	14	17

The cost parameters above may be determined from log data the server collects. Now the costs of the functional bases $\mathcal{F}_{comp_i}$ are

	$\mathcal{F}_{comp_1}$	$\mathcal{F}_{comp_2}$	$\mathcal{F}_{comp_3}$
Stage 1	2370	851	2613
Stage 2	2844	4482	3540
Stage 3	2376	2244	9810

Therefore, on Stage 1 — because of the poor network access — the optimization decides to access the compressed `PS_LZW` format via the network and to compute the uncompressed `PS` from it. On Stage 2 — due to the improvement of network access — the proxy server revises this choice and accesses both formats via the network, even the big `PS` format. On Stage 3, as the network access rate decreases and the proxy receives only a few `PS` requests, it is slightly cheaper to compute `PS` from `PS_LZW` again.

In this example we considered two attributes and two conversion tools only. It illustrates that this kind of optimization — even if the cost parameters would be static — could hardly be done manually. Note that local optimization on the proxy server can influence the storage server. In Example 21, as the number of requests for `PS_LZW` increases, the storage server optimization may decide to additionally store the `PS_LZW` format or even to store `PS_LZW` instead of `PS`.

Example 22

A new data format is released, e.g., the `JBIG` format [JBI93] for `text image` showing a superior compression rate (110 kByte compared to 160 kByte for an average page). Assume that in the moment according to $\mathcal{F}_{comp}$ the format `PNG300` is stored for this class, $c_S := \frac{1}{kByte}$, and that the value of z is very low, e.g., $z = 0.05$. Thus, simplifying the considerations we ignore the access costs. Then $store\_cost(\mathcal{F}_{comp}) = 160$ and $store\_cost(\mathcal{F}_{comp} \cup \{\texttt{JBIG2PNG}\}) = \texttt{110}$, hence the `JBIG` will be stored in the future.

Note that introducing a new format at the storage server does not influence the proxy servers directly. However, in the process of updating the software on the proxy servers, in our example the optimization on these servers may decide to access the smaller `JBIG` and compute the formats requested by the clients itself. Moreover, it is not necessary to convert all objects of class `text image`

to the new physical schema immediately. Instead we can introduce a subclass of `text image` for new B/W raster images with the additional `JBIG` attribute and additional constraints of the form `PNG300 = JBIG2PNG(JBIG300)` and migrate existing objects of `text image` to this new class little by little.

Example 23
A storage server runs out of disk storage and tertiary storage in form of a CD-ROM jukebox has to be added. We do not change the value of z. One might assume that the optimization tends to store more attributes because there is plenty of storage volume available now. But instead, as the value of c_A increases (storage access gets a little bit more expensive), the optimization will try to reduce the storage need of the objects and thus to compute even more attributes.

6 Summary and Outlook

In this paper we discussed an optimization problem that is ubiquitous in client-server multimedia exchange systems. A variety of different multimedia data formats required by the clients faces conversion tools that interrelate those formats. Thus the server can compute some formats as well as physically represent them. We formalized the conversion tools as functional constraints among the attributes of an object-oriented schema design and presented an algorithm to determine an optimal division of physically represented and computed attributes. To cope with the high complexity of this algorithm we presented techniques to reduce the search space. Finally we illustrated the benefits of our approach by several scenarios arising from a practical application, the journal article retrieval and delivery system ELEKTRA.

In this paper we assumed more or less dumb clients sending requests to intelligent proxy servers performing the optimization described above. With the advent of the Java technology [AG96] for Web browsers the borderline between clients and proxy servers gets futile.

Our approach implies that an attribute of an object-oriented schema design is not necessarily physically represented but may be computed. This enhances physical data independence of query methods, but it complicates update methods. Updates in multimedia systems are exceptional. Nevertheless, in general, they occur. In [KKK96] we examined possibilities to deal with this problem.

Based on the results in [KKK96] and [Ste96] we are currently implementing a prototype for a storage and a proxy server. Afterwards we will experiment on appropriate parameters for cost functions. Furthermore the question of good default values for those parameters and how often to re-optimize the servers have to be examined thoroughly.

As a summary we think that our approach can contribute to relieve the global data networks from their current bandwidth crisis.

Acknowledgments

We would like to thank Christopher Forkin, Alexander Clausnitzer, and Hans Kempe for fruitful discussions.

References

[AG96] K. Arnold and J. Gosling. *The Java Programming Language*. Addison Wesley, 1996.

[BM93] E. Bertino and L. Martino. *Object-Oriented Database Systems*. Addison Wesley, 1993.

[Cat94] R. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1994.

[FGM+96] R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol — HTTP/1.1, July 1996. Draft — work in progress.

[JBI93] Information technology — coded representation of picture and audio information — progressive bi-level image compression. International Telecommunication Union, Recommendation T.82, March 1993.

[KB96] S. Khoshafian and B.A. Baker. *MultiMedia and Imaging Databases*. Morgan Kaufmann Publishers, 1996.

[KKK96] W. Kowarschick, G. Köstler, and W. Kießling. Subtyping by constraints in object-oriented databases. In *Proc. Int'l. Symposium on Object Technologies for Advanced Software (ISOTAS'96)*, volume 1049 of *LNCS*, pages 287–307, 1996.

[Mas91] Y. Masunaga. Design issues of OMEGA: An object-oriented multimedia database management system. *Journal of Information Processing*, 14(1):60–74, 1991.

[Mey88] B. Meyer. *Object Oriented Software Construction*. Prentice-Hall, 1988.

[PNG] Portable network graphics. http://quest.jpl.nasa.gov/PNG.

[Pos90] Adobe Systems Inc,. *PostScript Language Reference Manual*, 2^{nd} edition, 1990.

[Sed88] R. Sedgewick, editor. *Algorithms*. Addison-Wesley, 2nd edition, 1988.

[SSe96] A. Silberschatz, M. Stonebraker, and J. Ullman (editors). Database research: Achievements and opportunities into the 21st century. *SIGMOD Record*, 25(1), March 1996.

[Ste96] W. Stein. Storage optimization of object-oriented database schemas with functional integrity constraints. Master's thesis, Institut für Informatik, Universität Augsburg, 1996. In German.

[Tra96] TransAction Software GmbH. *TransBase Relational Database System, Version 4.2.2*, 1996.

[TUL] The TULIP final report.
http://www.elsevier.nl/locate/tulip.