A Heuristics-Based Approach to Query Optimization in Structured Document Databases*

Dunren Che and Karl Aberer
GMD–IPSI, Dolivostr. 15, 64293 Darmstadt, Germany
{che, aberer}@darmstadt.gmd.de

Abstract

The number of documents published via WWW in form of SGML/HTML has been rapidly growing for years. Efficient, declarative access mechanisms for this type of documents – structured documents in general – are becoming of great importance. This paper reports our most recent advance in pursuit of effective processing and optimization of structured-document queries, which are important for large repositories of structured documents. Our methodology emphasizes applying exclusively deterministic transformations on query expressions to achieve the best possible optimization efficiency. A new approach is thus proposed that facilitates the exploitation of the DTD-knowledge, structural properties, and structure indices of structured documents for the purpose of fast query optimization.

Keywords. *Query optimization, document database, document management, structured document, SGML-document.*
General topics. *Query processing and optimization, text databases.*

1. Introduction

Structured documents form the basis of electronic publishing, both in the traditional, paper, form, and in the electronic form as available on WWW. A huge amount of structured documents, especially on the Web and in the digital library (DL) area, has been accumulating day and night. Efficient, declarative access to the document collections is becoming increasingly important.

Query optimization has been typically used in database management systems (DBMS) as an effective way of accelerating the evaluation of posed queries. Specifically, query optimization is to transform an input query (expression) to another form that is typically a much improved (if not the "optimal") alternative to the original one with regard to the efficiency of the evaluation plan generated eventually for that query. The very functionality – query optimization – is necessary for the management of large document repositories (let's call it document databases) because of two reasons. First, as the repositories of structured documents tend to be large, document query evaluation is becoming unacceptably slow. Second, while a simple interface is always desired by the users and consequently simple form-based query interfaces to information systems on the Web are now popular, more complicated query functions may be formulated through proper combination of individual simple ones. Thus, the achieved queries may not be fairly appropriate from the viewpoint of the users nor adequately efficient in evaluation. Query optimization can be exploited to deal with this issue equally well as in the traditional database environment.

We therefore initiated a project – HyperStorM (Hypermedia document Storage and Modeling) [3, 4] to investigate the appropriate techniques for the management of structured documents, in particular the processing/optimization of declarative queries in structured-document databases.

We argue for that structured documents should benefit from the same database management functionality offered to traditional data, although past efforts made to model structured text in a relational DBMS turned out to keep encountering difficulties [22]. But object DBMSs seem to be well suited for structured document management. So in this research we choose our own OODBMS, VODAK [19], as an experimenting platform.

SGML has long been used in the publishing industry for electronic creation of documents. XML (eXtensible Markup Language)[21] is a new emerging standard – a light-weight SGML – that will replace HTML as the basis of future WWW documents. Therefore SGML-compliant documents have been chosen as targets for management and querying in our work.

Toward a declarative query language, we adopted the PAT algebra [18] as it is designed as an algebra for searching structured documents and bears the salient features of being data model independent and highly declarative. In

*This paper is published in the proc. of the IDEAS99 international symposium.

contrast, most related work is based on plain extensions to relational or object-oriented query languages, like OQL-doc [2], or devising new sophisticated ones [17, 15, 12]. An interesting query language is recently proposed: XML-QL [9] takes a similar form as SQL but incorporates XML document pieces as search patterns into the queries.

In the context of this work, structured document data are stored within an object-oriented database, and queries are formulated using the PAT algebraic language that is eventually mapped to the representation of the underlying host system for evaluation.

A first result of our work regarding document query processing has been reported in [4]. Since last year we have carefully examined the adopted approach and drawn valuable lessons from this experience. We redesigned our testbed according to a new approach. Our new approach is better elucidated from three aspects: methodology, deterministic transformation (using rules), and implementation. Due to space limitation, in this paper we focus on the general methodology part.

The main optimization issue which we address in this paper is fast, algebraic optimization of SGML/XML document queries that enables structure index substitution into a query according to DTD knowledge and heuristics, which "traditional techniques", e.g., object optimization, may not achieve. Our approach directly facilitates the exploitation of structural properties of documents for query optimization, and performs exclusively *deterministic* transformations to achieve high optimization efficiency through a relatively simple implementation.

The remainder of this paper is arranged as follows: Section 2 provides a brief introduction to the preliminary knowledge needed for this paper; Section 3 addresses our new approach and methodology. Section 4 describes our rule system organization, and presents representative rules and a running optimization example. Section 5 reviews previous work, including our own forgoing approach. Section 6 provides concluding remarks and indicates future work.

2. Preliminary

In this section, we introduce the basic concepts of SGML and the PAT algebra, which we use to specify declarative access to document databases.

2.1. SGML overview

One significant concept about SGML and XML is the document-type definition – DTD, which is mandatory for SGML documents and optional for XML documents. HTML itself is a DTD specified in SGML. One important objective of this work is to take the full advantage of DTD knowledge, obtaining efficient optimization of structured-document queries. To offer a first impression on DTD and SGML documents, we give the logical structure of a sample DTD in Figure 1, which at the type level reflects the structure of the documents complying with the DTD, and is formally referred to as a *DTD-graph* as in [4, 6].

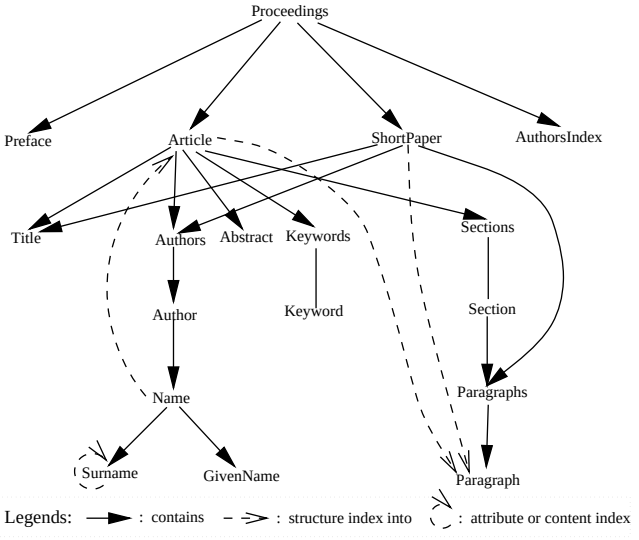


Figure 1. Graph of a Sample DTD

In Figure 1, each vertex corresponds to an element type defined in the DTD. A DTD-graph is based on the *content model* of each element type involved in the DTD.

The *containment* relationship between element types is essential for the structure definition of SGML-compliant documents. It is also essential for identifying useful DTD knowledge for efficient optimization of document queries. The notion comes out in two senses: in the plain case, it refers to the *directly-contains/directly-contained-in* relationship; in the second case, it is rendered as the result of a limited number of applications of this transitive relationship. Therefore, it suffices to give only the plain case definition of this concept.

Definition 1 (Directly-contains/contained-in) *Element type ET_i directly-contains element type ET_j if there is an occurrence of ET_j in the content model of ET_i (this corresponds to an edge in the DTD-graph). Conversely, ET_j is directly-contained-in ET_i.*

The *DTD-graph* helps visualizing the important *containment* relationships among document components induced by the DTD.

Also, we accept the trivial case of this notion, i.e., both "ET_i contains ET_i" and "ET_i is contained-in ET_i" hold.

In addition, three other notions about the structural properties of a DTD are very valuable to the semantic query optimization we concerned. But due to space limitation, here we provide only informal interpretations of them, instead of repeating the definitions given in [6].

Obligation. With regard to the containment relationship "$E1$ contains $E2$", $E1$ *obligatorily* contains $E2$ if the occurrence of $E2$ in the content model of $E1$ is not an *optional* one in any sense.

Exclusivity. With regard to the containment relationship "$E1$ is-contained-in $E2$", $E1$ is *exclusively* contained in $E2$ if $E1$ occurs only in the content model of $E2$.

Entrance location. With regard to the containment relationship "$E1$ contains $E2$" (or "$E1$ is-contained-in $E2$"), element type $E3$ is called an *entrance location for* $E1$ *and* $E2$ if there does not exist such a traversal path between $E1$ and $E2$ in the DTD-graph that does not go through $E3$ ($E1$ and $E2$ are trivial entrance locations for themselves).

2.2. PAT algebra

The PAT algebra, originally described in [18], is designed as an algebra for searching structured documents. Toward declarative access to structured document databases, we chose PAT as an algebraic query language due to the distinguished features [6, 3, 4] of it. The algebra we adopted in this research is an extended one for further suiting the particular features of SGML documents.

The PAT algebra is *set* oriented in the sense that each PAT algebra operator and each PAT expression evaluate to a *set* of document elements. Consequently, the three basic operations of set theory, *union*, *intersection* and *difference*, are all incorporated into the PAT algebra. A complete version of the extended PAT algebra is elaborated in a previous paper [3]. In the following, we present just a short version of it, which is enough for the purpose of this paper.

All PAT query algebra expressions are generated according to the following grammar:

$$E ::= etn \mid E1 \cup E2 \mid E1 \cap E2 \mid E1 - E2 \mid \sigma_r(E) \mid \sigma_{A,r}(E) \mid E1 \subset E2 \mid E1 \supset E2 \mid (E)$$

E (as well as $E1$ and $E2$) generally stands for a PAT expression, and etn for a document element type name, while r, as a regular expression, indicates a condition to be met on the textual content, and A designates a specific attribute of document elements on which a regular expression is to be matched.

$\cup$, $\cap$ and $-$ are the standard *set* operations: *union*, *intersection* and *difference*. A PAT algebra expression is valid if the two arguments of each involved set operation are type-compatible, i.e. both return elements of the same type. We do not assume that PAT algebra queries can only be posed with regard to typed documents. Nevertheless, it is required for any expression to have a well defined type, and as a consequence, type constraints on the composition of PAT expressions are imposed with regard to type compatibility of these binary set operations.

$\sigma_r(E)$ takes a set of elements and returns those whose content match the regular expression r, while $\sigma_{A,r}(E)$ takes a set of elements and returns those whose value of attribute A match the regular expression r. Operator $\subset$ returns all elements of the first argument that are contained in an element of the second argument, while $\supset$ returns all elements of the first argument that contain an element of the second argument.

The $\supset$ and $\subset$ operators reflect the "contain/ contained-in" relationship in our set based algebra. They allow to refer within document queries to the document structure and will be our focus on optimization for structured document queries.

In addition, two more notions need to be remarked:

Type of PAT expressions. Each expression, say E, with defined types for the involved operands, evaluates to a set of document elements of a single type, namely $\tau(E)$.

Extension of PAT expressions. For a given document base (database state), each expression, say E, with defined types, evaluates to a set of document elements, which is referred to as the *extension* of the expression, and denoted as $ext(E)$. The extension of an etn expression (i.e., an element type name) consists of all the elements of that type. If the evaluation of an expression E returns all the elements of type $\tau(E)$, the extension is said "free of restriction", and is notated as $free(E)$.

Finally in this subsection, we give a concrete query formulated using the introduced PAT operators. As a running example, the query will later be used to illustrate the transformation process of our rule system.

Example Find all paragraph containing both "OLAP" and "multidimension" from either *Article* or *ShortPaper*.

$$((\sigma_{r=`OLAP'}(Paragraph) \subset Article) \cup (\sigma_{r=`OLAP'}(Paragraph) \subset ShortPaper)) \cap ((\sigma_{r=`multidimension'}(Paragraph) \subset Article) \cup (\sigma_{r=`multidimension'}(Paragraph) \subset ShortPaper))$$

3. Methodology

In our testbed [3, 4], document databases are built on an OODBMS as a platform. Query processing in our framework [6] consists of two separate stages: the PAT stage, as a preprocessor of PAT expressions for the host OODB system, and the OQ (object query) stage which invokes standard object query optimization and evaluation for optimized query expressions.

Input queries are first checked for syntactic correctness and semantic validness by the PAT paser. Passed queries then get to the PAT expression optimizer (or PAT optimizer for short) which in turn performs normalization, semantic optimization, and expression simplification. After that, the query expressions are expected to be considerably improved (or optimized). The optimized expressions now are mapped to equivalent OQ forms which immediately initiate a "standard" object query processing.

The kernel part of structured-document query processing in our work is the PAT optimizer, which is implemented as three transformation phases (i.e., *normalization*, *semantic optimization*, and *expression simplification*) with the main goal of exploiting DTD-knowledge and related heuristics to achieve fast query optimization.

Our approach thus can be characterized as *2-stage* processing and *3-phase* optimization, applying the *all deterministic transformation* strategy [1] for fast query optimization. The rationale of our approach are discussed in following subsections.

3.1. 2-stage processing

In principle, structured-documents are a specific class of composite objects for which OODBMS is conventionally designed; consequently, structured-document queries can be treated as a class of object-oriented queries. In the meanwhile, structured-document queries pose additional processing requirements beyond the "standard" capabilities of most OODBMSs. For example, query processing calls for the exploitation of the DTD knowledge of structured documents and the specific semantics of the documents characterized into PAT expressions in our case.

Therefore, query optimization in structured document database environments can be reasonably characterized from two complementary perspectives: i.e., the one concerning structured-document specific features (which we simply call as *PAT-features* since the PAT algebra is designed to characterize these features), and the one relating to standard *object-oriented* features. The two aspects are independent of each other. So the "divide-and-conquer" rule applies pretty well to the optimization of structured-document queries, and we separate the entire processing of structured-document queries into two stages, PAT-stage and OQ-stage. The first stage concentrates on the exploitation of structured-document specific features for query optimization. The task is greatly facilitated by the adoption of PAT algebra as a query language at which optimal transformations are performed. The second stage focuses on object-oriented features for query optimization, thus the underlying OODBMS is directly used for this task.

[1]The "all deterministic transformation" strategy is to be addressed later in this section and in Section 4 in detail.

This approach protrudes the valuable features of structured-document queries by separating PAT-feature based optimization from the common assignment of object query optimization. The exploitation of the rich PAT semantics, structural properties and DTD-knowledge of structured-documents is most favorably achieved at the PAT expression level, i.e., semantic transformations are to be performed on the PAT expressions of the queries, rather than on the expressions formulated according to the underlying object algebra designed for object query optimization as we did previously [4, 6].

3.2. 3-phase optimization

Rule-based approach bears a number of advantages [1], one of which is the convenience in formulating various transformation criteria. Using a rule-based approach, structured-document query optimization consists of algebraic and semantic transformations on query expressions. In our case for PAT expression optimization, the transformations are organized as three phases because of the following reasons:

(1) While semantic transformation is recognized as an effective (short-cut) way of optimizing a query, it may not be attained to in an effective way in practice. A carefully selected *normal form* is typically used to serve as a starting point for effectively initiating semantic optimization. For example, expression "$E \cup E$", without being first simplified (through normalization) as a single E, subsequent semantic transformations pursued on the second occurrence of the same E (sub)expression would be a complete waste of the precious optimization time. Therefore, a specific preparation phase, commonly called *normalization*, is favorably designed for this purpose.

(2) Semantic transformations typically introduce new PAT (sub)expressions encompassing new element types and new PAT operators, which are very likely to be redundant with each other and with the existent part of the original input expression. Therefore, after all semantic transformations are performed, a final cleaning-up phase – *simplification*, is required.

Therefore, PAT expression optimization in our framework is deployed as a three-phase organization – *normalization*, *semantic optimization*, and *simplification*, among which the semantic optimization phase is put between the two others.

We will come back to the three transformation phases with more details in the next section.

3.3. All deterministic transformation

It has been stressed for many times that "fast achieving a much improved alternative" for an input query expression is

far more important than pursuing the much expensive "optimal" one in practice. This is especially true in the context of optimizing structured-document queries as in our HyperStorM scenario. Due to the same consideration, we come up to the adoption of the so-called *all deterministic transformation* strategy that performs exclusively deterministic transformations on query expressions. Deterministic transformation in the context of our research refers to the application of a transformation rule that determinately transforms an input expression into a much improved alternative. The implementation of this strategy requires a broad variety of heuristics which exploit diverse resources such as DTD-knowledge, structural properties, and structure indices of documents for the purpose of query optimization.

Specifically, two aspects motivated us to the adoption of this strategy:

(1) Many heuristics with regard to query optimization are available in the context of structured document databases. Appropriate use of these heuristics helps make the right choice from a number of alternatives during optimization without the need of evaluating their costs. Among others, structure indices are a class of the main sources of these heuristics; they are very valuable for bringing about significant improvements on query expressions but may not be straightforwardly attainable. These knowledge can be even more favorably exploited for query optimization if used in cooperation with the DTD-knowledge and other structural properties of the structured documents.

(2) Object-oriented features stand for another essential facet of structured-document queries. The host OODBMS serving as a platform can be treated as a reliable backing support for the PAT optimizer; it is capable of performing object-oriented optimization on structured-document queries. This means that the host query system may compensate to a certain degree for the possible imperfectness of the preceding PAT optimizer. This relieves us a lot of concerns about the likely incompleteness of the PAT optimizer – a slight incompleteness of the PAT optimizer as a preprocessor does not matter seriously. In another word, using a sound OODBMS as a platform, from another aspect, warrants our *all deterministic transformation* strategy.

Consequently, our optimization system always aims at the current state of a PAT expression during transformation, while previously generated and less interesting alternatives need never be maintained for further consideration, and uninteresting candidates will never be produced.

Rule-based approach to query optimization has long been criticized for its uncontrollable behavior, which is crucial for performance. This problem is ideally solved in our scenario through the incorporation of determinism into the system. Actually, the determinism in a transformation system can be either *rigid* or *less-rigid*. A less-rigid deterministic system maintains a reduced number of alternatives for final choice-making, while a rigid system like our PAT optimizer focuses always on the current alternative. In both cases, determinism is typically attained through the use of appropriate heuristics.

In the sense of rigid determinism, the optimization of a query expression becomes a linear sequence of invoking applicable transformation rules. The finally reached candidate is treated as the "optimal". Choice-making among multiple candidates is only implicitly reflected by the order of rules in the rule system, especially when they share the same input pattern and the same precondition.

Toward deterministic optimization, various heuristics according to diverse PAT expression patterns and dozens of deterministic transformation rules have been developed in our testbed. The system uniquely leads each input query expression step-by-step, eventually reaching a significantly improved one.

4. The Rule System

The application of the optimizer which we envisage is in a highly dynamic environment, like *ad-hoc* querying or distributed query processing on the WWW. Thus the efficiency of the optimizer is a predominant goal as stated. Therefore the strategy we choose is to perform exclusively deterministic transformations on the query expressions, rather than to evaluate different alternatives. These transformations are required to lead to step-by-step improvements of the query expressions until a final expression is reached. The criteria which we use to determine the improvements are heuristics-based. This is achieved by exploiting available index structures and interesting DTD properties at the PAT query level, because SGML-specific semantics can be easily exploited at this level.

4.1. The rule system organization

The main objective of the PAT query optimizer is to identify opportunities where by exploiting knowledge on the DTD, dramatic improvements in the efficiency of query evaluation may be achieved.

Toward this objective, in addition to semantic optimization, selected standard query transformations, such as straightforward simplification and selection pushing-down (in the term of a tree-like representation of the expression), are also required. All these rules are organized within three different phases. The first phase normalizes the operator order of the incoming query and performs simplification; the second phase accomplishes the desired semantic transformations with the primary goal of exploiting structure indices; and a third phase performs a final cleaning-up of the transformed expression. The finally achieved expression is

treated as the "optimal" one and passed on to the underlying query processing system.

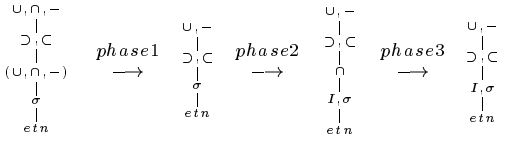
For a concise presentation, we introduce the following shorthand notations:

"$\begin{smallmatrix}\cup\\ \cap\end{smallmatrix}$" – stands for set operator $\cup$ and/or $\cap$

"σ_R" – for selection operator σ_r and/or $\sigma_{A,r}$

"$\begin{smallmatrix}\supset\\ \subset\end{smallmatrix}$" – for structure operator $\subset$ and/or $\supset$

Now we discuss the three phases and the corresponding strategy behind.



(1) (2) (3) (4)

Figure 2. Illustration of transformation strategy

Phase 1: Query normalization. The first phase normalizes incoming queries according pattern (2) of Figure 2. The algebraic properties of the PAT query algebra can be most favorably exploited for achieving the following goals:

1. Isolate the different types of operators at different levels in the expression.
2. Eventually eliminate $\cap$ operators.
3. Perform the selection operators, σ and $\begin{smallmatrix}\supset\\ \subset\end{smallmatrix}$, first.

Pushing the σ_R operators to the bottom level simplifies the identification of potential attribute and content indices. In addition, as a heuristics, we assume that selection operators are the most selective and efficient operators to evaluate. The isolation of the $\begin{smallmatrix}\supset\\ \subset\end{smallmatrix}$ operators into the next upper level is an important prerequisite to enable the identification of opportunities for exploiting structure indices. Collecting the remaining set operators at the top level is concluded according to the heuristics of "performing unselective operations as late as possible". We expect that incoming queries will have, in general, the form of pattern (1) in Figure 2 that is already quite close to the form of pattern (2) achieved by normalization, in particular selection predicates will be specified more early while set operators are typically applied to the selection results. In the first phase, a number of simplifications are also performed, like simplifying the expression "$E \cup E$" to a single E, to make the subsequent semantic transformation phase more efficient.

Phase 2: Semantic optimization. Our second phase starts at pattern (2), ends at pattern (3), and typically with *index* operators being interpolated, intersections may be reintroduced again. Identification of the use of attribute and content indices is fairly simple. For selection operators, the availability of indices can be checked straightforwardly. The potential use of structure indices, either be using general types of equivalences, like associativity laws, or particular properties of the DTD, like the existence of *entrance locations* is explored next. If the application of structure indices is not feasible, reduction of the traversal path into the documents' structure is still possible, which again exploits the particular structural properties extracted from the DTD, and is achieved through appropriate transformations.

Phase 3: Query simplification. As semantic transformation rules, either cooperatively or separately, may introduce new operators, in particular, new element types, containment operators, and intersection operators, thus a further run of the simplification rules of the normalization phase for a bottom-up simplification is required.

In the following we give several representative rules used in each of the three phases, together with a discussion of the underlying heuristics. Most of the rules will later be used in the transformations performed on our running example.

4.2. Representive rules

Transformation rules are distinguished from the more general *equivalences* in that transformation rules target at deterministic improvement on an input expression; the improvements are determined using heuristics. But the correctness of each such transformation rule comes from the equivalence that the rule maintains between the two sides of it. A large set of appropriate equivalences have been carefully identified and described in a related paper [6]. Transformation rules, being *unidirectional*, take the form "$(E1) \implies (E2)$". An additional precondition may be added to some rules to determine the applicability of the rule to an input expression.

4.2.1 Normalization rules

Query normalization covers three aspects: DTD-constraint based simplification (like $\mathcal{R}1$), operator reordering (like $\mathcal{R}2$ - $\mathcal{R}7$), and a bottom-up simplification (like $\mathcal{R}8$ - $\mathcal{R}11$). Following are just several representatives of the rules in this group.

$\mathcal{R}1.\ (E1 \subset E2) \Longrightarrow \phi$ if $\tau(E2)$ doesn't contain $\tau(E1)$
$\mathcal{R}2.\ (E_1 \cap (E2 \cup E3)) \Longrightarrow ((E1 \cap E2) \cup (E1 \cap E3))$
$\mathcal{R}3.\ ((E_1 \cup E2) \cap E3) \Longrightarrow ((E1 \cap E3) \cup (E2 \cap E3))$
$\mathcal{R}4.\ (\sigma_R(E1) \cap E2) \Longrightarrow (\sigma_R(E1 \cap E2))$
$\mathcal{R}5.\ (E1 \cap \sigma_R(E2)) \Longrightarrow (\sigma_R(E1 \cap E2))$
$\mathcal{R}6.\ (E1 \supseteq_{\subset} E2) \cap E3 \Longrightarrow (E1 \cap E3) \supseteq_{\subset} E2$
$\mathcal{R}7.\ E1 \cap (E2 \supseteq_{\subset} E3) \Longrightarrow (E1 \cap E2) \supseteq_{\subset} E3$
$\mathcal{R}8.\ (E \cap E) \Longrightarrow (E)$
$\mathcal{R}9.\ ((E1 \supseteq_{\subset} E2) \supseteq_{\subset} E2) \Longrightarrow (E1 \supseteq_{\subset} E2)$
$\mathcal{R}10.\ ((E1 \supseteq_{\subset} E2) \cup E1) \Longrightarrow (E1)$
$\mathcal{R}11.\ (E1 \cup (E1 \supseteq_{\subset} E2)) \Longrightarrow (E1)$

In stead of giving proof, we provide brief explanation of above rules. $\mathcal{R}1$ directly comes from the DTD-constraint used. $\mathcal{R}2$ and $\mathcal{R}3$ are exactly the $\cap$ distribution laws of set theory. As $\supseteq_{\subset}$ operations restrict the first argument's extension by imposing a containment relationship with the elements of the second argument, we can more simply consider $\supseteq_{\subset}$ operations as a further kind of selections. In this sense, $\mathcal{R}4$ through $\mathcal{R}7$ are communications of intersection with a certain selection; their correctness are self-evident. $\mathcal{R}8$ is yet another basic law of set theory. $\mathcal{R}9$ is to delete redundant restriction imposed by the containment relationship with the same argument. $\mathcal{R}10$ and $\mathcal{R}11$ both hold because $(E1 \supseteq_{\subset} E2)$ is a subset of $E1$.

4.2.2 Semantic rules

Semantic rules are developed with the predominant goal: enabling the exploitation of structure indices during optimization, which in most cases is not readily achievable, rather, relying on a deep exploration of DTD-knowledge such as *obligation*, *exclusivity*, and *entrance location* with regard to the two element types connected through a *containment* operation.

Six different cases have been identified for exploiting structure indices into a query (*cf.* [6]). The simplest case is to directly use an available structure index between the two element types involved:

$\mathcal{R}12$. $E1 \supseteq_{\subset} E2 \Longrightarrow I_{\tau(E1)}(E2) \cap E1$
if a structure index between $\tau(E1)$ and $\tau(E2)$ is available

$I_{\tau(E1)}(E2)$ denotes a structure index operation defined between $\tau(E1)$ and $\tau(E2)$, where the subscription $_{\tau(E1)}$ also indicates the result type of the operation.

This rule is based on an index substitution equivalence[2], interpolating an index operation into an expression. On the right-hand side, the newly introduced intersection with the $E1$ subexpression is necessary because the interpolated structure index reflects only the restriction of the $E2$ on the result type's extension, but not the original selection imposed by the $E1$.

[2]The equivalence takes the general form, $(E1 \supseteq_{\subset} E2) \iff (I_{\tau(E1)}(E2) \cap E1)$.

The second case is designed to reveal the applicability of a potential structure index that is not directly available. The corresponding rule combines the $\supseteq_{\subset}$ *commutativity* and *associativity* laws (related equivalences are detailed in [6]) into a single transformation:

$\mathcal{R}13$. $(E1 \supseteq_{\subset} (E2 \supseteq_{\subset} E3)) \Longrightarrow ((I_{\tau(E1)}(E3) \cap E1) \supseteq_{\subset} E2)$
if a structure index between $\tau(E1)$ and $\tau(E3) exists$

Proof.
$(E1 \supseteq_{\subset} (E2 \supseteq_{\subset} E3))$
$\Longrightarrow ((E1 \supseteq_{\subset} E2) \supseteq_{\subset} E3)$ ($\supseteq_{\subset}$ associativity) (1)
$\Longrightarrow ((E1 \supseteq_{\subset} E3) \supseteq_{\subset} E2)$ ($\supseteq_{\subset}$ commutativity) (2)
$\Longrightarrow ((I_{\tau(E1)}(E3) \cap E1) \supseteq_{\subset} E2)$ ($\mathcal{R}12$) (3)
∎

The four other cases require a deeper exploration of structural properties of the documents. Here we show just one of them:

$\mathcal{R}14$. $(E1 \subset E2) \Longrightarrow (E1 \subset E3)$
if $\tau(E2)$ is an entrance location for $\tau(E1)$ and type $E3$;
$\tau(E2)$ is exclusively contained in $E3$ and $free(E2)$ holds;
and a structure index between $E3$ and $\tau(E1)$ is available.

As a precondition, this rule asks for additional structural properties being held, i.e., entrance location and exclusivity, which are deducible from the DTD (*cf.* [6] for the algorithms), and the condition $free(E2)$ that holds iff $ext(E2) = ext(\tau(E2))$ – typically this means $E2$ is a plain element-type-name and does not impose any selection condition on the extension of this element type. The proof (*cf.* [6]) of this rule is based on the definitions of these structural properties, and is omitted here due to space limitation.

Rules simply applying available σ_R indices are also collected in the semantic rule group. In addition, if applying indices into a query is not feasible, a less favorable opportunity – reducing the length of required traversal into the documents' structure (as no structure index is available), will be explored.

4.2.3 Simplification rules

The third phase recollects all the simplification rules of Phase 1. But those applying DTD-constraints to query expressions are not considered since semantic transformations cannot introduce new inconsistencies with regard to the DTD. In addition, two new rules are added to cope with the simplification of the newly introduced index operator $I_{\tau(E1)}$ and $\cap$.

$\mathcal{R}15.\ (I_{\tau(E1)}(E2) \cap E1) \Longrightarrow (I_{\tau(E1)}(E2))$ ***if*** $free(E1)$ holds
$\mathcal{R}16.\ (I_{\tau(E1)}(E2) \cap \sigma_R(E1)) \Longrightarrow \sigma_R(I_{\tau(E1)}(E2))$ ***if*** $free(E1)$ holds

If a structure index and a selection index both are available, $\mathcal{R}16$ give the precedence to the structure index as structure navigation into the document database is assumed the most expensive and thus should be by-passed as soon as an appropriate structure index is available.

4.3. An optimization example

We now present the transformations performed on our running example query given in Section 2.2. The main goal of these transformations is to enable the application of the structure indices $I_P(A)$ and $I_P(S)$ as depicted in Figure 1 ('P', 'A' and 'S' are used as short notations of 'Paragraph', 'Article', and 'ShortPaper', respectively).

First, we reformulate this query for easy presentation. Let σ_1 and σ_2 stand for "$\sigma_{r=`OLAP'}$" and "$\sigma_{r=`multidimension'}$", respectively, and rewrite 'Paragraph' as 'P', 'Article' as 'A' and 'ShortPaper' as 'S' for short. The transformations performed are illustrated as follows:

$(((\sigma_1(P) \subset A) \cup (\sigma_1(P) \subset S)) \cap$
$((\sigma_2(P) \subset A) \cup (\sigma_2(P) \subset S)))$
$\stackrel{step1}{\Longrightarrow}$ ($\cup$ pushing up by $\mathcal{R}3$)
$(((\sigma_1(P) \subset A) \cap ((\sigma_2(P) \subset A) \cup (\sigma_2(P) \subset S))) \cup$
$((\sigma_1(P) \subset S) \cap ((\sigma_2(P) \subset A) \cup (\sigma_2(P) \subset S))))$
$\stackrel{step2}{\Longrightarrow}$ (again $\cup$ pushing up by $\mathcal{R}2$)
$((((\sigma_1(P) \subset A) \cap (\sigma_2(P) \subset A)) \cup$
$((\sigma_1(P) \subset A) \cap (\sigma_2(P) \subset S))) \cup$
$(((\sigma_1(P) \subset S) \cap (\sigma_2(P) \subset A)) \cup$
$((\sigma_1(P) \subset S) \cap (\sigma_2(P) \subset S))))$
$\stackrel{step3}{\Longrightarrow}$ ($\cap$ pushing down via $\mathcal{R}6$ and $\mathcal{R}7$)
$(((((\sigma_1(P) \cap \sigma_2(P)) \subset A) \subset A) \cup$
$(((\sigma_1(P) \cap \sigma_2(P)) \subset A) \subset S)) \cup$
$((((\sigma_1(P) \cap \sigma_2(P)) \subset S) \subset A) \cup$
$(((\sigma_1(P) \cap \sigma_2(P)) \subset S) \subset S)))$
$\stackrel{step4}{\Longrightarrow}$ (redundant $\subset$ deletion by $\mathcal{R}9$)
$((((\sigma_1(P) \cap \sigma_2(P)) \subset A) \cup (((\sigma_1(P) \cap \sigma_2(P)) \subset A) \subset S)) \cup$
$((((\sigma_1(P) \cap \sigma_2(P)) \subset S) \subset A) \cup ((\sigma_1(P) \cap \sigma_2(P)) \subset S)))$
$\stackrel{step5}{\Longrightarrow}$ ($\cap$ pushing down and deleting by $\mathcal{R}4$, $\mathcal{R}5$ and $\mathcal{R}8$)
$(((\sigma_1(\sigma_2(P)) \subset A) \cup ((\sigma_1(\sigma_2(P)) \subset A) \subset S)) \cup$
$(((\sigma_1(\sigma_2(P)) \subset S) \subset A) \cup (\sigma_1(\sigma_2(P)) \subset S)))$
$\stackrel{step6}{\Longrightarrow}$ ($\cup$ simplification via $\mathcal{R}10$ and $\mathcal{R}11$)
$((\sigma_1(\sigma_2(P)) \subset A) \cup (\sigma_1(\sigma_2(P)) \subset S))$
$\stackrel{step7}{\Longrightarrow}$ (index introduction by $\mathcal{R}12$)
$((I_P(A) \cap \sigma_1(\sigma_2(P))) \cup (I_P(S) \cap \sigma_1(\sigma_2(P))))$
$\stackrel{step8}{\Longrightarrow}$ (giving precedence to structure-index by $\mathcal{R}16$)
$(\sigma_1(\sigma_2(I_P(A))) \cup \sigma_1(\sigma_2(I_P(S))))$

In above transformation, $I_P(A)$ and $I_P(S)$ are used as short notations of index operations. Step 1 to step 6 perform normalization (note that $\cap$ is pushed down, which appears not to be a very intuitive strategy but enables the final elimination of all $\cap$); step 7 interpolates structure indices into the query; and step 8 confirms the precedence of the introduced index operations in this query (also cleaning up the extra $\cap$ operators).

One may argue that an alternate formulation for above example, $((\sigma_1(P) \cap \sigma_2(P)) \subset (A \cup S))$, might be more reasonable to most human users. But the adopted one is, in any sense, a practical one; it is even more exemplary with simple form-based query interfaces that support query combination to formulate more complex queries.

5. Previous Work

In this section, we briefly refer to previous related work, including our own experience in applying an open extensible OODBMS to the management, especially query processing, of structured documents

As structured document management becomes increasingly important, much work has been done on the modeling and store of structured documents into a database system. With regard to choosing an OODBMS as a platform, targeting at SGML documents, and focusing on declarative query mechanisms, just a few work are related.

[2] is most related to ours regarding the adoption of a readily available OODBMS as a platform: they select O2 while we use VODAK. But, toward almost the same goal – to store and query SGML documents in an OODBMS, the both undertook from different approaches. The basic idea in [2] is to extend both the query language and the underlying algebra (and the data model). Whereas in our case, the query optimization exploiting the specific features of structured documents is separated as a preprocessor for the underlying VODAK query optimizer. Another major difference is that we chose an extended PAT algebra as query language. This turns out to be a valuable choice, as we got not only a declarative, user-friendly, and expressive document query language, but also the potential of applying SGML-compliant semantics directly to query optimization owing to the well-suited features of the PAT algebra.

Being part of a large project conducted by five institutions in Canada, the work reported in [16, 17] aims at the developing of an object DBMS to store and manage SGML/HiTime-compliant multimedia documents. To our knowledge, their work regarding object-oriented SGML document database management is mostly on inherent modeling support. Stressing on the central use of OODBMS technology, sophisticated type systems for multimedia documents were reported in both [16] and [17].

As for structured-document query optimization, the work reported in [7] tries to replace a query-algebra operator with a cheaper one whenever the DTD allows. The DTDs considered are simpler than the ones of SGML, and the authors do not look at the different grammar constructors. The optimization in [7] makes use of a special cost model, and the results are thus not directly transferable to our application scenario.

Other generally related work includes [15, 12, 22, 7, 5, 13, 14, 11]. Among them, [15] and [12] are the two more recent and interesting ones. In [15] a model for querying document databases via both content and structure is presented; while in [12] an OO model and query language for hypermedia and multimedia documents are proposed with the emphasis on multiple complex structures.

Finally, we shed light on the previous approach we adopted in the HyperStorM testbed for the processing/optimization of structured document queries. The lessons we drew from our own experience, to a large extent, motivated this ongoing effort.

In the past several years, we have been practising the application of an open extensible OODBMS for the management of structured documents. The HyperStorM [3, 4] was developed as a testbed based on the VODAK OODBMS [19]. Extending existent facilities of the underlying OODBMS for modeling and querying structured documents was identified and adopted at the inception of our research. *Extensibility* ever motivated intensive research for *extensible* DBMS [10]. Our experience with structured-document query optimization reveals that a plain application of the "extensible DBMS" approach does not work well in practice as expected. In our case, we extended the VODAK query optimizer by adding quite a number of new transformation rules exploiting the specific features of structured documents into the rule system of it. We arrived at the following conclusions:

(1) The specific features of structured documents are better characterized using a commensurate high-level algebra, like our PAT; query transformation is better performed on the expressions of this high-level algebra to most facilitate exploitation of the specific semantics of structured-documents on query optimization (The semantics otherwise can easily get lost during the mapping of the query expressions to the lower level ones complying with the underlying object algebra.)

(2) The "pre-processing" approach is more appropriate for the optimization of structured document queries, and the host OODBMS can be more directly and efficiently used for the management and querying of structured documents.

6. Summary

In this paper, we presented a new approach to the query optimization in a structured-document database. The approach is strongly heuristics-based. In this work, we choose SGML-compliant documents as targets and adopted an extended PAT algebra as the user query language. One major objective of this work is to achieve fast optimization for input queries, which is then passed on to a host OODBMS used as a platform. The application of this approach which we envisage is in a highly dynamic environment, like *ad-hoc* querying or distributed query processing on the WWW. Fast processing/optimization of declarative queries for structured documents is critical for large DL systems and diverse document repositories as available on the WWW.

The proposed approach greatly facilitates the exploitation of the specific semantics of structured-documents, and of the various structural knowledge and the structure indices of the documents. The approach consists of two processing stages, and organizes the query expression optimization into three transformation phases, which performs exclusively *deterministic transformations* on the expressions.

The characteristics of our approach are highlighted below:

- The host query system is directly put into effect for the processing of structured-document queries.
- The various DTD-knowledge, structure indices and specific semantics of SGML documents are adequately exploited for the purpose of query optimization.
- The proposed approach performs exclusively deterministic transformations on query expressions.
- The implementation of this approach is simple because of both the direct use of the host system and the deterministic transformation strategy; this simplicity contributes additional efficiency to the processing of structured document queries.

Although our approach is proposed under the framework of HyperStorM that is based on an OODBMS platform and adopts an extended PAT algebra as query language, but it is not much strategically dependent on a specific OODBMS nor on the PAT algebra. In this approach, PAT (expressions) is only used as a representation of queries to perform optimization. The techniques we developed in this work for the optimization of structured-document queries can equally well apply to other declarative query languages as long as they can adequately characterize the specific semantics of structured documents. The OODBMS used in our approach as a platform can also be substituted for by other kind of document repositories (not necessarily being based on OODBs).

Currently, we have a limited implementation of this new approach, in which transformation rules are experimented using a C++ implementation. Our planed work for the near future includes: (1) obtain a fully polished implementation of the current rule system, (2) integrate our approach with the XML-QL [9] query language to support fast query optimization for XML document databases.

Acknowledgments. We gratefully acknowledge the contribution of Dr. Böhm and Professor Özsu to our work. The valuable previous work made by Dr. Böhm in this institute with the HyperStorM project serves as a good basis for this ongoing effort; and the fruitful cooperation of the

authors with Professor Özsu in a related paper [6] is highly regarded as very helpful for this writing. We also appreciate very much the help of Dr. Yangjun Chen for the final polishing of this paper, and the helpful comments of the anonymous reviewers.

References

[1] Karl Aberer, Dunren Che, Klemens Boehm. *Rule-based Generation of Logical Query Plans with Controlled Complexity*. In Proc. of the Fifth Int. Conf. on Deductive and Object-Oriented Databases (DOOD'97), Montreux, Switzerland, December 8-12, 1997.

[2] Serge Abiteboul, Sophie Cluet, Vassilis Christophides, et al. *Querying Documents in Object Databases*. Digital Libraries. No. 1, 1997, pp. 5-19.

[3] Klemens Böhm, Karl Aberer, Erich J. Neuhold, et al. *Structured Document Storage and Refined Declarative and Navigational Access Mechanisms in HyperStorM*. The VLDB Journal, Vol. 6, No. 4, November 1997, pp. 296-311.

[4] Klemens Böhm, Karl Aberer, M. Tamer Oezsu, et al. *Query Optimization for Structured Documents Based on Knowledge on the Document Type Definition*. In Proc. of IEEE Int. Forum on Research and Technology Advances in Digital Libraries (ADL'98), pp.196-205, Santa Barbara, California, April 22-24, 1998.

[5] Surajit Chaudhuri and Luis Gravano. *Optimizing Queries over Multimedia Repositories*. In Proc. the 1996 ACM SIGMOD Int. Conf. on Management of Data, pp. 91-102, Montreal, Canada, June 1996.

[6] Dunren Che, Karl Aberer, M. T. Özsu, et al. *Query Processing and Optimization in Structured Document Databases* (a pre-publishing version prepared for VLDB Journal).

[7] Mariano Consens and Tova Milo. *Optimizing Queries on Files*. In Proc. of the 1994 ACM SIGMOD Int. Conf. on Management of Data, pp. 301-312, Minneapolis, Minnesota, May 1994.

[8] Tuong Dao. *An Indexing Model for Structured Documents to support Queries on Content, Structure and Attributes*. In Proc. of IEEE Int. Forum on Research and Technology Advances in Digital Libraries (ADL'98), pp.88-97, Santa Barbara, California, Aprill 22-24, 1998.

[9] Alin Deutsch, Mary Fernandez, Daniela Florescu, et al. *XML-QL: A Query Language for XML*. 1998 A submission version to the World Wide Web Consortium 19-August-1998 (http://www.w3.org/TR/NOTE-xml-ql/).

[10] G. Gräfe and W.J. McKenna. *The Volcano Optimizer Generator: Extensibility and Efficient Search*. In Proc. of 9th ICDE, pp. 209-218, Vienna, Austria, April 19-23, 1993.

[11] G.H. Gonnet, R.A. Baeza-Yates, and T. Snider. *Information Retrieval – Data Structures and Algorithms*. New Indices for Text: PAT trees and PAT arrays, Prentice hall, 1992.

[12] Kyuchul Lee, Yong Kyu Lee and P. Bruce Berra. *Management of Multi-Structured Hypermedia Documents: A Data Model, Query Language, and Indexing Scheme*. Multimedia Tools and Applications, Vol. 4, No. 2, march 1997, pp. 199-224.

[13] Atsuyuki Morishima and Hiroyuki Kitagawa. *A Data Modeling and Query Processing Scheme for Integration of Structured Document Repositories and Relational Databases*. In Proc. of the Fifth Int. Conf. on Database Systems for Advanced Applications, Melbourne, Australia, April 1-4, 1997.

[14] Atsuyuki Morishima and Hiroyuki Kitagawa. *A Data Modeling Approach to the Seamless Information Exchange among Structured Documents and Databases*. In Proc. of 1997 ACM System on Applied Computing, San Jose, Feb. 1997.

[15] Gonzalo Navarro and Ricarrdo Baeza-Yates. *Proximal Nodes: A Model to Query Document Databases by Content and Structure*. ACM Transaction on Information Systems, Vol. 15, No. 4, October 1997, pp. 400-435.

[16] M. T. Özsu, D. Szafron, G. El-Medani, et al. *An Object-Oriented Multimedia Database System for a News-on-Demand Application*. Multimedia Systems, No. 3 , 1995, pp. 182-203.

[17] M. T. Özsu, P. Iglinski, D. Szafron, et al. *An Object-Oriented SGML/HiTime Compliant Multimedia Database Management System*. In Proc. of Fifth ACM Int. Multimedia Conf. (ACM Multimedia'97), pp. 239-249, Seattle, WA, November 1997.

[18] A. Salminen and F. W. Tompa. *PAT Expressions: an Algebra for Text Search*. Acta Linguistica Hungarica 41 (1994), no.1, pp.277-306.

[19] *VODAK V 4.0 User Manual*, Tech. Report 910, GMD-IPSI, April 1995, St. Augustin.

[20] *The NCSA Beginner's Guide to HTML* (http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html).

[21] *The SGML/XML Web Page* (http://www.sil.org/sgml/sgml.html) copyright (c) Robin Cover 1994-1998.

[22] Tak W. Yan and Jurgen Annevelink. *Integrating a Structured-Text Retrieval System with an Object-Oriented Database System*. In Proc. of the 20th VLDB Conf., pp. 740-749, Santiago, Chile, 1994.