A Wrapping Architecture for IR Systems to Mediate External Structured Document Sources

Yoshiharu Ishikawa  Takehiro Furudate  Shunsuke Uemura

Graduate School of Information Science
Nara Institute of Science and Technology (NAIST)
8916-5 Takayama, Ikoma, Nara 630-01, Japan
{ishikawa, takehi-h, uemura}@is.aist-nara.ac.jp

Abstract

With the growth of digital libraries and electronic publishing, many structured document sources are appearing and their effective mediation is an important research topic. In this paper, we propose a wrapping architecture for externally maintained structured document sources. Our wrapping target is information retrieval systems (IRSs) that provide access to structured documents. We describe a wrapper construction method for such IRSs with limited functionality. A constructed wrapper enhances retrieval facilities of the underlying IRS and provides an object database view to the mediator. We focus on determining whether the underlying IRS can support a given query. Then we discuss some research issues related to our wrapping architecture.

Keywords Document databases, interoperable databases, wrappers, mediators, structured documents, SGML, object databases

1 Introduction

Due to the rapid growth of the Internet, a large number of information sources are available online today, and their integration is becoming increasingly important [2]. Miscellaneous document sources are considered to be typical examples of such information sources. For example, bibliographic sources on the World-Wide Web contain useful information so that we may require extraction of the bibliographic contents and their integration with other existing databases or information sources. To integrate distributed and heterogeneous sources, the concept of a mediator has been proposed and examined [2, 16, 20]. A mediator integrates data from heterogeneous sources and provides applications with a clear and integrated view.


In recent years, many structured document sources have appeared alongside the growth of digital libraries and electronic publishing. Some digital library projects utilize SGML [11] for electronic document formats [7, 12] and some online bibliography collections use BibTex-formatted bibliographies [9]. The use of such structured document sources and their integration with other sources are important research topics.

Generally, an online structured document source maintains its documents and utility by itself; they may provide site-specific facilities for document access and retrieval. In prior research concerning structured document databases [3, 8, 14, 18], highly expressive data models and languages were proposed or assumed to handle document structures. However, we cannot generally rely on such powerful modeling/query facilities for external structured document sources. In our research, we assume information retrieval systems (IRSs) for structured documents (SGML documents) as external sources. The IRSs provide the Boolean retrieval facilities [10]; users can specify search conditions by arbitrarily connecting keywords via and, or, and not Boolean connectors. The IRSs also provide other retrieval facilities; for some predefined SGML elements (e.g., author, title for bibliographic sources), users can specify Boolean conditions in addition to conventional keyword-based conditions.

In this paper, we discuss a wrapper construction method for IRSs which store structured documents. A wrapper (translator) is a software component that translates data from disparate sources into a common model and provides a common query language for extracting information [16, 17]. In our context, a wrapper converts queries issued by the mediator into search expressions for the underlying IRS. When a wrapper receives a retrieved result from the IRS, it converts the result into common data model objects that the mediator can comprehend. We adopt
the ODMG-93 object data model [4] as the common data model; the wrapper receives OQL queries and returns ODMG objects.

As described above, in this paper, we discuss a wrapper construction method for IRSs which store structured documents. In particular, we focus on SGML documents because of their popularity and importance. In section 2, we explain the background and the objective of our research. In section 3, we describe assumptions upon which the underlying IRSs are based and introduce the functionality of a wrapper in our context. Then we discuss the wrapper construction method. In section 4, we describe the algorithm used to determine whether the underlying IRS can support a given query. In section 5, we discuss remaining issues, such as the problem of providing object identities and the improvement of performance and query power using views. Finally, in section 6, we offer our conclusions.

2 Background

2.1 Mediators and Wrappers

Research concerning the integration of heterogeneous sources is ongoing [2, 16]. Roughly speaking, our research is one of these attempts. Figure 1 shows a conceptual diagram of our distributed mediated environment. A mediator integrates heterogeneous sources and provides an integrated view to users. A wrapper offers a common data model and a query language to the mediator and converts given queries into search conditions which are applied to the underlying sources. The mediator accesses heterogeneous sources such as RDBMSs, ODBMSs, IRSs and World-Wide Web servers.

In this research, we focus on IRSs containing structured documents and wrappers for these IRSs. Our wrapper wraps such an IRS as if it was an ODBMS that conforms to the ODMG-93 standard [4] and provides an OQL query interface to the mediator. The wrapper translates OQL queries into search expressions which are applied to the underlying IRS and converts retrieved results into ODMG objects.

One of the important functions of wrappers is enhancement of the query facilities of the underlying sources [17]. For a source with limited functionality, we cannot always push or send query conditions given by the mediator into the source. Query conditions that cannot directly apply to the source will constitute a filter to the retrieved intermediate result. Formulation of search expressions which are applied to the underlying sources and generation of filters are crucial issues in the query processing of wrappers.

Figure 1: A Distributed Mediated Environment

Another important concept for wrappers is query support [17]. When a wrapper can formulate an appropriate search condition and a filter to fulfill a given query, we say the wrapper can support the query. Such a query is called a supported query. When a wrapper cannot generate an appropriate search expression or a filter, the wrapper cannot support the query and the query is called an unsupported query; in this case, the mediator cannot receive the desired result. It is also a key issue for wrappers to decide whether given queries can be supported or not.

2.2 Structured Documents and Databases

In the area of database research, much effort has been expended on structured documents. One of the main topics is data models and query languages for structured document databases [3, 8, 14, 18, 21]. Such data models and query languages are required to provide rich data modeling and query facilities to express or handle document contents and structures.

From the system-side view, there are also many proposals for architectures of structured document database systems and integration schemes for DBMSs and document retrieval/management systems. Roughly speaking, we can distinguish two approaches as follows [6].

Tight Integration: A DBMS directly manages documents and can evaluate all queries internally. The DBMS query processor can efficiently use its access methods and query processing facilities for document data to evaluate queries. This approach is further divided as follows.

1. Structured documents are physically stored in databases [8].
2. A DBMS has a parser to extract data objects from structured documents stored as conventional files [1]. Such a parser is considered to be an access method for documents.
Loose Integration: When a DBMS evaluates queries, it calls upon foreign functions provided by external document retrieval/management systems [3, 6]. Such integration of DBMSs and external IRSs is an important research topic [15]. A key issue is effective utilization of IR facilities (e.g., document ranking, probabilistic retrieval).

Our approach is similar to the loose integration approach with respect to a few points. However, our approach is different from the above approaches as follows.

First, many prior studies implicitly (or explicitly) assumed that databases and document retrieval/management systems are maintained within the same site or highly related sites. Such assumptions afford some advantages to DBMSs. For example, a DBMS may be allowed to access internal data structures and internal search facilities of the IRS; a DBMS may be notified when the documents in the document management system have been modified. However, we assume that the underlying IRS storing structured documents is a non-proprietary external source. Therefore, we cannot access the internal data structures or search facilities of the IRS and cannot receive any notification from the IRS. Moreover, we may not be able to form the simple query "retrieve all documents" due to the lack of such facilities. Although the IRS we assume carries many constraints, many existing online sources such as bibliographic search engines on the World-Wide Web belong to such category.

Second, we do not provide any expressive data models or query languages to manipulate structured documents. Our objectives are to provide a common view (namely, an ODMG-93 database view) to an existing IRS which stores structured documents, and to efficiently evaluate given OQL queries over the IRS. The wrapper translates OQL queries into search expressions for the IRS, and retrieved results are coerced to ODMG objects. Therefore, some structures and semantics of the retrieved documents will be lost.

3 Wrapping Architecture
3.1 Description of IR Facilities

As an example, we consider archaeological excavation reports written as SGML documents. A sample SGML document is shown in Fig. 2. The DTD (Document Type Definition) for this SGML document is shown in Fig. 3. The upper part of the report contains some bibliographic information (e.g., title, author) and archaeological information (e.g., district, era). The body (surrounded by <body>... </body>) contains textual information and a description of the findings. We assume that the target IRS provides document retrieval facilities for such excavation reports, and these reports are written according to the same DTD shown in Fig. 3.

```xml
<DOCTYPE report SYSTEM "report.dtd">
<report repid="R60021">
<title>XYZ Tumulus: The Excavation Report of Ancient Burial Mound in Kyoto, Japan</title>
<author><first-name>Taro</first-name> <last-name>Susuki</last-name></author>
<era>late kofun age</era>
<district>Kinki</district> <pref>Kyoto</pref>
<site>burial mound</site>
<affil>... </affil> <date>...</date>
<body>
<section><title>...</title>
<p>...</p>
</section><title>Finds</title>
<p><find id="ho0431">
  <name>bit</name>
  <class>horse equipment</class>
  <descr>
    oval plates.
    gilt bronze on iron ground.
    silver gilt rivet.
  </descr>
</find>
<p><find id="so0025">
  <name>sword</name>
  <class>arm</class>
  <descr>
    long iron sword.
    with dragon head handle.
  </descr>
</find> ....
</section>
</body>
</report>
```

Figure 2: A Document Instance

We assume that the underlying IRS supports simple Boolean retrieval facilities; we can specify one or more keywords by arbitrarily connecting them with the Boolean connectors (and, or, not). When a user specifies a search expression, the IRS retrieves SGML documents based on the contents of their body elements. Namely, for each stored SGML document, the IRS removes tags in the body element and converts the element into a simple text string; then the IRS indexes the text string for fast retrieval. The following is an example of simple Boolean retrieval facilities.

% search 'horse' and 'equipment'

The IRS retrieves all reports that contain the two words 'horse' and 'equipment' in their body elements. We assume that the search command of the IRS is case-insensitive. Other examples are shown below.

% search 'horse' or 'equipment'
Step 1: An OQL query is given to the query processor from the mediator.

Step 2: The query processor decides whether or not the IRS can support the query. The decision scheme for the support problem is described in section 4.

Step 3: If the query is supportable, the query processor generates a search expression for the underlying IRS, an extraction pattern, and a filter query for the retrieved result. Their use is described below.

Step 4: Given the search expression, the IRS retrieves SGML documents based on the expression. The retrieved SGML documents are sent to the SGML parser with the corresponding DTD.

Step 5: The SGML parser parses and checks each SGML document and sends the result to the extractor. The result is a parsed SGML document that contains tags without any omissions; with the assistance of the tag information, the following processes are simplified.

Step 6: The extractor extracts elements from the SGML documents based on the extraction pattern and carries out simple filtering processes when they can be performed at a low cost. This is achieved by a pattern matching function of the extractor, for example, as in the case of sgrep (structured grep) [19].

Step 7: The extractor assembles ODMG objects using the extracted SGML elements. In some cases, the assembly process will become very complex because costly structural operations such as grouping may be contained in the process.

When filtering SGML documents (step 6) and assembling objects (step 7), we must take object equality and object identity into account.
This issue is described in subsection 5.1.

**Step 8** Finally, the filter query is applied to the generated ODMG objects. The filter query is an OQL query which filters out objects that do not satisfy the query conditions. The result is returned to the mediator. Although it is better if we can perform all filtering processes in step 6, some filtering processes may require the implementation semantics of ODMG types. In such cases, we will need a filter query.

```java
interface Find
(extent Finds)
{
    attribute String name;
    attribute String class;
    attribute String district;
    attribute String pref;
    attribute String era;
    attribute String site;
    attribute Text descr;
    relationship List<Author> is-reported-by
    inverse Author::reports;
}
interface Author
(extent Authors)
{
    attribute String first-name;
    attribute String last-name;
    relationship Set<Find> reports
    inverse Find::is-reported-by;
}
```

Figure 5: An ODL Interface Definition

Fig. 5 shows an ODL definition for a wrapper in our example. The wrapper wraps the IRS containing structured documents that conform to the sample DTD in Fig. 3. In subsection 3.4, we describe how to relate each ODL attribute or relationship to the corresponding SGML element. The Text type is not contained in the ODMG-93 standard. We added the type for convenience to express raw texts and query conditions applied to them. The Text type provides a predicate contains that decides whether all the given words are contained in the Text attribute.

### 3.3 Example Query

In this subsection, we describe the query processing scheme with the aid of an example. Let us consider that the following query, Q1, is given to the wrapper from the mediator.

```
Q1: select f.name
    from Finds f
where f.is_reported_by.last_name = "Suzuki" and f.descr contains
    ("iron", "plates")
```

This query retrieves the names of finds which are reported by authors with the last name 'Suzuki', and the descriptions for the finds contain both of the words 'iron' and 'plates.' For a find object f extracted from a find element in a document, the path f.is-reported-by.last_name corresponds to the last-name element in the document. The path f.descr corresponds to the concatenation of raw texts extracted from the descr element (item elements) within the find element.

In query processing, the query processor first decides whether the IRS can support the query. As described before, the IRS provides basic Boolean retrieval facilities for raw texts in body elements and additional retrieval facilities for the specific elements title, first-name, and last-name. Therefore, the search expression

```
% search -ln 'suzuki'
```

can at least support the query. However, since the descr element is contained in the body element in the DTD definition, we can derive other search expressions such as those shown below.

```
% search 'iron' and 'plates'
% search -ln 'suzuki' and 'iron'
and 'plates'
```

Although all of the above search expressions can retrieve SGML documents that actually contain the required information, their query processing times and the number of mismatches in the retrieved results are different. However, such difference may be highly dependent on the internal processing and the statistical information of the underlying IRS. For such external sources, it is difficult or impossible to obtain such information. Therefore, we adopt a simple strategy for query processing.

**Strategy 1:**

*Push as many query conditions into the IRS as possible.*

In this example, the final search expression is given to the IRS based on this strategy.

The retrieval result from the IRS is SGML documents. The example document in Fig. 2 will be contained in the result of query Q1. These SGML documents are sent to the SGML parser and their document structures are analyzed and omitted tags are supplemented explicitly. Then the extractor extracts specified elements based on the extraction pattern. The extractor may need
to perform simple pattern matches. For example, consider the OQL query condition 
\`f . is-reported-by. last-name = "Suzuki"\` in query Q1. This condition assumes case-sensitive pattern matching. However, the underlying IRS is assumed to be case-insensitive. Since the retrieved result from the IRS will contain case-insensitive matches, the extractor must perform a case-sensitive search on the retrieved SGML documents.

Next consider the OQL query condition 
\`f . descr contains ("iron", "plates")\'. As described above, the search expression that is actually sent to the IRS searches documents that contain the words 'iron' and 'plates' in their bodies. However, the original request of query Q1 is to search finds such that descriptions for them contain both 'iron' and 'plates' together. Therefore, the extractor must specify more restrictive extraction conditions which reflect the original query request. For the sample SGML document in Fig. 2, the extraction result contains only the following find element.

```xml
<find id="ho0431"> <name>bitc/naae> <class>horse equipment> <descr> <i>oval plates.<i>gilt bronze on iron ground.<i>silver gilt rivet.<</descr> </find>
```

Since query Q1 requires the value of name elements in the retrieved find elements, the extractor extracts the value of the name elements from the intermediate result. In the above example, 'bit' is extracted.

### 3.4 Wrapper Generation

It is hard work to implement a wrapper from scratch. However, wrappers built according to our framework will contain many similar processes. Therefore, we need a wrapper generator tool for the purpose of wrapper construction with semi-automatic generation of components from declarative specifications.

For simple information sources (e.g., finger command), Papakonstantinou et al. [17] describe a framework for such a wrapper construction toolkit.

Now we describe the design of our wrapper generator tool under development. To construct a wrapper for an IRS, the following information and specifications are required.

1) The SGML DTD for documents stored in the target IRS.
2) An ODL definition that provides a wrapper interface for the mediator.

3) A description of mapping information that relates each attribute or relationship in the ODL type definition to an element of the corresponding SGML DTD.
4) A template tree describing the search conditions that the IRS can support. The use of a template tree to solve the query support problem is described in the next section.

Using this information, the wrapper generator generates skeletons for functions to be called from the wrapper. The implementor of the wrapper must supply the codes required for the functions.

For the integrated specification of 2) and 3), we provide the Wrapper Interface Definition Language (WIDL). This language is actually an extension of the ODL language. It provides conventional ODL definition facilities and additional constructs to specify the mapping information between an ODL attribute or relationship and the corresponding SGML element. An example of WIDL definition is shown in Fig. 6.

```xml
<dtd "report.dtd">
    interface Find @$f . find$> (extent Finds)
        <ident @$f . id$>
        
    interface Author @$a . author$> (extent Authors)
        <ident @$a . first-name, $a . last-name, $a . info. affil$>
        
    relationship List<Author> is-reported-by
        inverse Author::reports
    
</dtd>
```

Figure 6: A WIDL Definition

We explain the example and describe the syntax of WIDL informally. An interface definition for a type T has the form

```xml
interface T @$p . path$> 
    (...) 
</ident ...

(...)>
```

436
where additional constructs '<$p \text{ path}>' and '<ident ...>' are shown. The construct '<ident ...>' is explained in subsection 5.1. The construct '<$p \text{ path}>' means that a type $T$ object is virtually created for each element corresponding to path. $p$ is a path variable and we can use it as an alias for the path path within the interface definition of the type $T$. We can use an extended path expression to specify a path. For example, '<$f \text{ * .find}>' in Fig. 6 uses the extended path expression 'e.find'. We can use the symbol '*' to abbreviate a path. In this case, 'e.find' is equivalent to paths 'report.body.sect.p.finds-list.find' and 'report.body.sect.subsect.p.finds-list.find'. Another extension is denoted by the symbol '...'. When a path 'p.q' exists, the path 'q.p' means a backward traversal of the path 'p.q'. For example, the path '$f...report.*.author' denotes the author elements such that the authors write reports for the find $f$.

To specify mapping information between an ODL attribute and an element in the DTD, we use the notation '<attr attribute Text descr <= \$f .descr>' in Fig. 6. Generally speaking, there may be two or more item elements for a descr element. In such a case, we assume that all item elements are concatenated as an instance of the Text type.

4 Decision for Query Support

Given a query from the mediator, the query processor must decide whether the IRS can support the query. Additionally, the query processor must generate filtering information for the following filtering process. In this section, we show a decision algorithm for the query support problem.

4.1 Template Tree and Search-Pattern Tree

To illustrate the decision process, we now introduce two data structures: a template tree and a search-pattern tree. A template tree expresses the retrieval facilities that the underlying IRS provides. The tree T-IN in Fig. 7 is the template tree for our example IRS. T-IN expresses the specifiable paths in search expressions. Dotted arrows indicate indirect paths. The symbol '*' is merely used to simplify the expression; we should see as if there was a subtree at the point. Element names in italics are directly searchable elements.

The tree S-IN(Q1) in Fig. 7 is the search-pattern tree for the query Q1. A search-pattern tree expresses the search patterns specified in a query. A search-pattern tree is a graphical representation of a Boolean expression in the form called Disjunctive Normal Form (DNF). A Boolean expression is in DNF if it is the logical OR of clauses which are the logical AND of normal or negated words [5]. A search-pattern tree has an OR-node in the root, and the OR-node has one or more AND-nodes as its children. Each AND-node has one or more normal paths or negated paths. A negated path corresponds to a NOT search condition. In a search-pattern tree, element names in italics express the queried elements. S-IN(Q2) in Fig. 8 is a typical example of search-pattern trees.

In the example of Fig. 7, the decision process of query support is performed as follows.

1) The leftmost pattern 'report.*.last-name "suzuki"' in S-IN(Q1) perfectly matches the pattern 'report.*.last-name' in T-IN. Therefore, the IRS can support query Q1.

2) Based on Strategy 1, we must push as many search conditions as possible. The next pattern 'report.*.descr.*.iron" in S-IN(Q1) partially matches 'report.body' because the descr element is contained in the body element in the DTD definition. Thus, we can also push this condition into the IRS. However, we need a filtering process for this condition because the patterns do not match perfectly.

3) We can also push the rightmost pattern 'report.*.descr.*.plates" in S-IN(Q1) into the IRS because the pattern is equal to the case of 2) except for the constant.

We can generate a search expression based on the above process. However, we must perform filtering for at least two patterns...
'report.*.descr.*."iron" and 'report.*.descr.*."plates"'. If OQL and the IRS disagree with respect to the semantics of search conditions (e.g., case sensitivity), we may need an additional filtering for 'report.*.last-name."suzuki"'.

In the above example, when no pattern in S-IN(Q1) match those in T-IN, the query is unsupported.

4.2 Decision Algorithm

Before we discuss the decision algorithm, let us consider another query Q2.

Q2:
select f.name
from Finds f
where f.is_reported_by.author.first_name = "Taro" or f.pref = "Kyoto" and f.is_reported_by.author.last_name = "Suzuki" and f.descr not contains "horse"

This query is an example of queries containing or's and not's. The search-pattern tree S-IN(Q2) for this query is shown in Fig. 8.

![Figure 8: Search-Pattern Tree for Q2](image)

There are two AND-nodes in the search-pattern tree S-IN(Q2). Note that both of the two AND-nodes must contain matching patterns in order for query Q2 to become a supported query. The left AND-node clearly satisfies the condition. However, we cannot push the pattern 'report.*.pref."kyoto"' because it does not match any pattern in T-IN. Therefore, this pattern is used as a filter. We can push the remaining two patterns. Therefore, the IRS can support query Q2 and we can push three search conditions.

In Fig. 9, we show an outline of the process to decide whether the given query Q is supported. It also produces information for the following filtering step and push-list to generate a search expression.

In Check.And-Node(n), another strategy is used.

**Strategy 2:**

*If there are two or more matches for*
Strategy 3:
If there are two or more output formats, use the most specific one that fulfills the query requirement.

The decision for the most efficient output formats can also be made using the tree containment approach.

5 Discussion

5.1 Object Identity and Equality

In this paper, we have tried to provide an ODBMS view to an IRS using wrapper technology. In object databases, object identity is an important notion. How to supply object identities is a crucial problem in our context. For example, consider the following query.

```sql
select a.last-name, count(a.reports)
from Authors a
where a.reports.class = "arm"
```

In this query, the result depends on how the identities of author objects are determined.

For this kind of problem, we have proposed the method shown in Fig. 6. For example, we provided the Author type the following identity rule:

```xml
<ident $a.*.first-name, $a.*.last-name, $a..info.affil>
```

5.2 Use of Views

The mediator not necessarily need all of the structured documents that the IRS stores. In this case, it is possible to improve query efficiency. For example, the mediator may be only interested in archaeological information concerning only Nara and Kyoto prefectures. If the IRS provides search facilities for pref elements, we can define the view.

Define View Finds_of_Nara_Kyoto
```sql
select *
from Finds f
where $f.pref = "Nara" or $f.pref = "Kyoto",
```
and can query as follows:

```sql
select f.name
from Finds_of_Nara_Kyoto f
where $f.descr not contains "horse".
```

In this case, the query contains only the not condition, but the mediator can receive the desired response. Moreover, we could improve the query performance using the materialized view technique. The materialized view maintenance problem in our context is also considered to be an important research problem.

6 Conclusion

In this paper, we proposed a wrapping architecture for IRSs which store structured documents. We described a wrapper construction method for such IRSs with limited functionality. Our wrapper enhances query facilities of IRSs and provides an object database view to the mediator. We especially focused on the query support problem and provided an algorithm for decision. Our future work includes improvement of the decision algorithm using tree matching algorithms such as that described in [13], introduction of materialized view, trial to more practical IR systems as described in [5].

Acknowledgments

We are grateful to members of the Laboratory of Database and Media Systems (Uemura Laboratory), NAIST. We are especially grateful to the archaeologist Mituhiko Okayasu for his advice and thank him for sharing his extensive knowledge. We are also grateful to Professor Masatoshi Yoshikawa and Mr. Hiroyuki Kato at NAIST for their detailed suggestions. This work was supported in part by the Ministry of Education, Science, Sports and Culture, Japan under the Grant-in-Aid for Scientific Research No. 08244104 and No. 08780284.
References


