# AN EXAMPLE-BASED WEB-SITE CONSTRUCTION TOOL AND ITS IMPLEMENTATION

Atsuyuki Morishima
Dept. of Info. Sci. and Eng.
Shibaura Inst. of Tech.
Saitama, Saitama, Japan
amori@sic.shibaura-it.ac.jp

Takanori Mouri
Doctoral Program in Sys. and Info. Eng.
Univ. of Tsukuba
Tsukuba, Ibaraki, Japan
tmouri@kde.is.tsukuba.ac.jp

Hiroyuki Kitagawa
Inst. of Info. Sci. and Elec.
Univ. of Tsukuba
Tsukuba, Ibaraki, Japan
kitagawa@is.tsukuba.ac.jp

## Abstract

This paper presents the design and implementation of a novel tool to construct data-intensive Web-sites based on information sources. The key idea is to show the system some *examples* of Web pages and let it infer how to construct the entire Web-site. The problem is challenging, because the system allows XML to be one of information sources. Because XML data is semistructured, it is a non-trivial task to infer the intended operation from given examples. Another feature of the system is that it outputs XQuery queries, so that it can work with general XQuery engines.

**Key Words:** Web Publishing, XML, Authoring Tools, XQuery

## 1. Introduction

Today, *Web publishing* of data stored in information sources is indispensable in industries. While most of the information sources are relational databases, it is no doubt that XML is going to be one of the important information sources for data-intensive Web-sites, as XML has become the de facto format for data exchange and archive. A lot of techniques have been proposed in this area. For example, Web-site management systems such as Strudel [3] can create various web-views of heterogeneous information sources, based on a semistructured data model.

Technically, Web publishing requires two types of tasks, i.e., extraction of data from information sources and design of Web pages into which the data is incorporated. Accordingly, existing techniques and frameworks provide users with two different types of tools, and users have to learn and use the two types of tools for Web-site construction. In typical cases, users use text editors to write SQL-style queries for data extraction and drawing tools to design Web pages.

This paper explains a novel Web-site construction tool named AQUA (Amalgamation of QUery and Authoring). AQUA helps users construct data-intensive Web sites based on XML repositories in a very intuitive fashion. The key idea is to show the system how to construct some *examples* of Web pages and let it infer how to construct the entire Web-site. So AQUA looks like just a common authoring tool for HTML and SMIL [8] pages: Users are only required to drag and drop data objects for Web page authoring. The difference is that the authoring task implicitly specifies the entire Web-site construction process.

This problem is challenging, because we allow data in the data repositories to be XML, and inferring (generalizing) the intended operations is a non-trivial task. In relational databases, the data structure is regular and explicit, so domains of data objects are fixed in advance. QBE [10] and other QBE-style query languages are designed based on the property. However, XML is a type of semistructured data [1][2], and the data structure is often irregular and implicit. In our framework, the domain is defined dynamically according to the user's interaction with the system

We developed a prototype system [6], and found through experiments that even users without any knowledge on database-related concepts (such as database schemas, SQL-style queries and path expressions) can construct fairly complex Web-sites from a collection of XML documents, by following their intuitions.

A key feature of AQUA from the implementation aspect is that it can work with ordinary XQuery [9] engines. Because XQuery is becoming a standard query language for XML we expect that we will be able to get a number of robust and efficient XQuery engines. AQUA can be used as an easy-to-use Web-publishing tool for XML documents managed by XQuery engines, *without any special requirements*.

In this paper we explain the system's overview and its implementation. As we will see, it is not a straightforward task to develop a practical system that works with general XQuery engines. We explain our solutions.

## 2. Running Example

We assume that we have information on musicians and songs in an XML repository. The repository contains the following three documents (Fig. 1): (1) A song-information document, that has a list of song titles, artists, and video-clips (Fig. 1 (a)). (2) A music-label document that has a list of music labels with their lists of artists (Fig. 1(b)). We assume that the document is *semistructured* data. Fig. 2 shows the tree representation of a part of data in the example scenario, where the two subtrees in the dotted boxes correspond to two kinds of XML elements for music label information; one has a flat list of `artist` elements, and another `artist` elements grouped by genre. We explain Fig. 2 later in detail. (3) An artist-profile document, that contains a list of

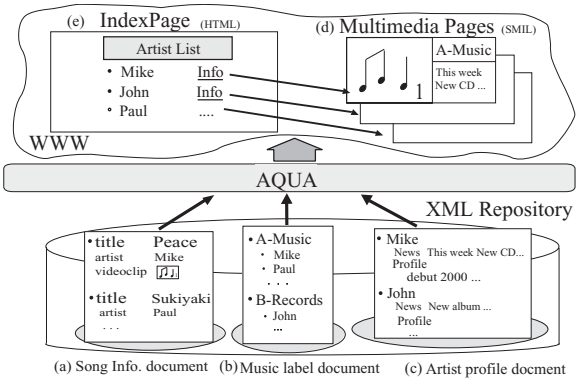



Figure 1. Example scenario

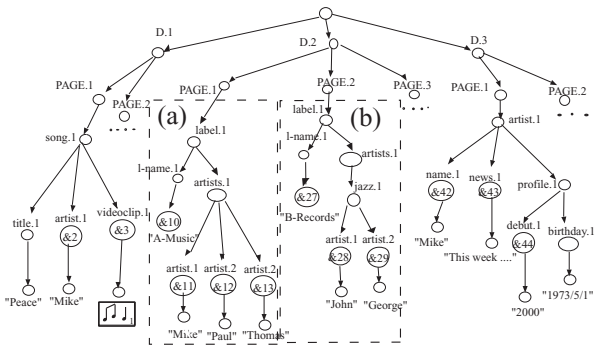



Figure 2. Tree representaion of data

artists (Fig. 1(c)). The information includes their names, the news on the artists, and profile information including their debut years.

The requirement here is to construct a Web-site, that has one Multimedia Web page (in SMIL) for each artist that made his debut in 2000 (Fig. 1(d)), and the index page for them (Fig. 1(e)).

## 3. The AQUA system

The system provides users with the following two types of windows for interaction:

**DataBox:** A *DataBox* is used to display data. Fig.3 shows DataBoxes in the running example. A DataBox shows a *page* at a time. In Fig. 3, a page corresponds to one of the followings[1]: (1) A `song` element (cf. Fig. 2) having information on a song ( is a reference to the video file), (2) a `label` element, and (3) an `artist` element. The user can click the Next and Previous buttons to browse other pages.

**Canvas:** The *Canvas* is a blank window onto which the user can drop data objects from DataBoxes.

### 3.1 Examples and Target Sets

Fig. 4 is a simple operation example[2]. *Drag-and-Drop* (D&D) is denoted by the dotted-lines. Here, we refer to

[1]The user can specify which element corresponds to a page.

[2]In the figure, multiple pages are shown simultaneously in a DataBox for explanation purposes. In reality, the Next and Previous buttons are used to view them.

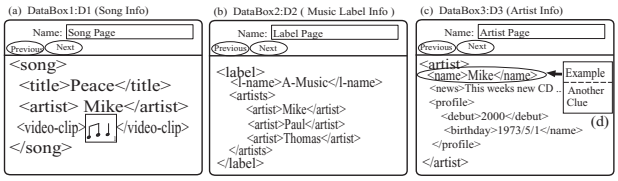



Figure 3. DataBoxes and a pop-up Menu

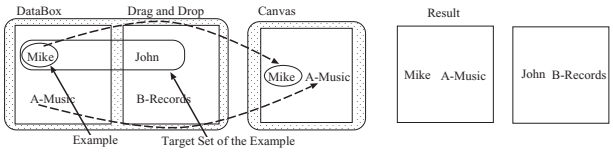



Figure 4. Manipulation of an example

the item being dragged and dropped as *an object*. XML elements and contents within the elements are objects. Objects dropped onto the Canvas appear in the result.

*An example* is denoted by the oval. The user can designate an object as an example by selecting the "Example" item on the pop-up menu (Fig. 3(d)) before the D&D operation.

An example has its *target set*, which is the set of objects the example represents. Manipulation of an example is interpreted as manipulation of the objects in its target set. Objects in a target set are highlighted in each DataBox.

As a default, the target set is defined as a set of objects that appear 'at the same position' on their pages as the example object. For example, in Fig. 3, the target set would be the set of artists' names in the artist-profile pages. "Another" and "Clue" menu items are used to modify the set.

The regular expression ( 'Example' ('Another' | 'Clue')* | 'D&D' )* shows the operation procedure in the AQUA system. Here, 'Example,' 'Another' and 'Clue' mean selections of respective menu items on an object. Intuitively, the system allows any combinations of the following operation patterns: (a) To designate an object as an example, and accept the default target set. (b) To designate an object as an example, and change the default target set by successive 'Another' and 'Clue' operations. (c) To drag-and-drop a non-example object (an object that the user did not designate as an example) onto the Canvas. In this case, only the object appears in the result. (d) To drag-and-drop an example object onto the Canvas.

If 'Another' operation is performed after 'Example' operation, the target set of the example is extended to include the 'Another' object and other objects that the system infers should be included into the target set. Intuitively, the system tries to generalize the relationship between the position of the example and that of the 'Another' object, and includes all the objects having the generalized relationship with the example into the target set.

After 'Example' operation, the user can choose another object and perform 'Clue' operation on it. The ob-

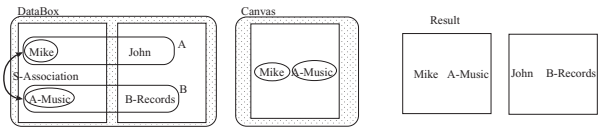



Figure 5. S-Association between A and B

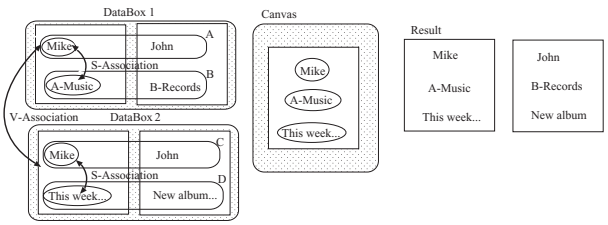



Figure 6. V-Association between A and C

ject is used to make the size of the example's target set smaller, by serving as a selection condition. For example, if he specifies 'Clue' operation on Mike's `<debut>` element and enters the condition "=2000," the Mike's target set is reduced to contain only the artists who made their debut in 2000.

## 3.2 Associations

When the user specifies multiple examples (and their target sets), there are usually *associations* among them. If an association occurs among target sets, only particular combinations of objects are qualified to be manipulated. Therefore, an association serves as a kind of join condition. Our framework supports two types of associations:

**Structural Association (S-Assoc.)** Two examples have an S-Assoc. when their positions have some special relationship. One of the simplest cases is that two different examples on the same page imply an S-Assoc. Suppose that the user wants to restructure pages in the DataBox in Fig. 5 so that the name and the music label are arranged side by side. If the user takes examples as in Fig. 5, an S-Assoc. is implied and he gets the intended result. In contrast, if he specifies 'John' as an example for target set A, the system considers that there is no S-Assoc. Therefore, the system outputs the Cartesian product ({(Mike, A-Music), (Mike, B-Records), (John, A-Music), (John, B-Records)}) of the two target sets.

**Value Association (V-Assoc.)** Two examples have a V-Assoc. when their values are the same. Suppose that we have two DataBoxes (Fig. 6), and that the user wants a set of pages, each of which contains an artist's name, the music label, and the news. If the user takes examples like those in Fig. 6, a V-Assoc. occurs and he gets the intended result. (Note that target sets A and B have S-Assoc., and so do target sets C and D.) If he specifies 'John' as an example for target set C, no V-Assoc. occurs and the system computes Cartesian product. The result would be { (Mike, A-Music, This week...), (Mike, A-Music, New album...), (John, B-Records, This week...), (John, B-Records, New album...)}.

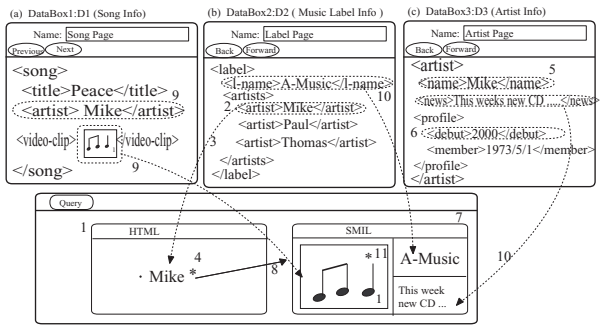



Figure 7. Manipulation for the scenario

## 4. Web-site Construction by Example

Fig. 7 illustrates the operation sequence to get the required result in the example scenario given in Sec. 2. We assume here that DataBoxes D1, D2, D3 contain the song-info. document, the music-label document, and the artist-profile document, respectively. `<a>` and `<n>` are abbreviations for `<artist>` and `<name>`, respectively.

**(1)** Open the Canvas and declare construction of an HTML page. (Then, the system opens a space for the HTML page on the Canvas.) **(2)** Designate '`<a>Mike</a>`' in D2 as an example (we call it $e_1$ here). The default target set includes artist names which appear first in music-label pages having the structure shown in Fig. 2(a). Next, specify that '`<a>Paul</a>`' (called $a_{11}$) in D2 is an 'Another' object. Then, press the Next button of the DataBox D2 to find another page that has the structure shown Fig. 2(b), and specifies that some artist (in this case '`<a>John</a>`') ($a_{12}$) is an 'Another' object. The system uses rules to generalize the relationship among positions of $e_1$, $a_{11}$, and $a_{12}$, so that the target set of $e_1$ is extended to include all the artist names. **(3)** D&D $e_1$ from D2 onto the Canvas. **(4)** Put a repetition mark (*) on 'Mike' on the Canvas. As a result, all the artists are listed in this page. Otherwise, a new page is produced for each artist. **(5)** Designate the '`<n>Mike</n>`' object ($e_2$) in D3 as an example. Note that the two target sets of $e_1$ and $e_2$ have V-Assoc. Therefore, this specifies an equi-join between their target sets. **(6)** Then designate the `<debut>` element in D3 ($c_2$) as the 'Clue' example for $e_2$, and enter the condition "=2000" so that the target set of $e_2$ includes only the names of artists who made their debut in 2000. **(7)** Declare construction of a SMIL page. The system opens a space for the SMIL page on the Canvas. **(8)** Connect a hypertext link from the HTML page to the SMIL page. **(9)**Designate the '`<a>Mike</a>`' element ($e_3$) and the video-clip ($e_4$) in D1 as examples. D&D $e_4$ onto the Canvas. **(10)**Designate the `<l-name>` element ($e_5$) in D2 and the `<news>` element ($e_6$) in D3 as examples. D&D them onto the Canvas. **(11)** Put the repetition mark (*) on the dropped video-clip. As a result, all of his video-clips are grouped and sequentially rendered in the same page. Otherwise, a new page is produced for each video clip.

## 5. Inside AQUA

This section explains how AQUA constructs Web-sites based on the user's interaction with the system.

As shown in Fig. 2, the system models the data as an object tree. Every node (object) is annotated with a *label*, which consists of a *label name* and a *label number*. Label numbers are sequentially assigned to sibling nodes with the same label name. Every object has an OID. Several OIDs are explicitly presented in the form of $\&n$ for explanatory purposes. $path(o)$ denotes the path from the root to the object $o$. For example, $path(\&11)$ =D.2→PAGE.1→label.1 →artists.1 →artist.1.

### 5.1 Computation of Candidate Sets

The system first computes *candidate sets* for example objects. Given an example $e$, $e$'s candidate set ($CS_e$) is defined as follows:

$$CS_e = \{o | o \in O \wedge C\text{-}Pred_e(o)\},$$

where $O$ is the set of all the objects in the object tree, and $C\text{-}Pred_e(o)$ is a *candidate predicate* incorporating a path expression. A path expression is similar to a path but may contain *wildcards*. $C\text{-}Pred_e(o)$ holds if and only if $path(o)$ conforms to the path expression. $C\text{-}Pred_e(o)$ is determined by the 'Example' and 'Another' operations as shown below.

**Example:** The following $CS_{\&11}$ gives the candidate set specified by Operation (2) in Sec. 4.

$$CS_{\&11} = \{o | \quad o \in O \wedge \text{D.2}\rightarrow\text{PAGE.?}\langle\text{PAGE.1}\rangle \rightarrow\text{label.1} \rightarrow\text{artists.1}\rightarrow{*}\langle\varepsilon\rangle \rightarrow\text{artist.?}\langle\text{artist.1}\rangle[o\langle\texttt{<a>Mike</a>}\rangle]\}.$$

The portions surrounded by "⟨" and "⟩" are *annotations*. Ignoring them, the path expression is D.2→PAGE.?→label.1→artists.1→*→artist.?. Given the source data shown in Fig. 2, $CS_{\&11}$ = {"<a>Mike</a>", "<a>Paul</a>", "<a>Thomas</a>", ... "<a>John</a>", "<a>George</a>", ...} (all artists of all the music labels).

The annotations give information on $path(e)$ and $value(e)$, that are used to derive the predicate. (In this case, $path(\&11)$ and $value(\&11)$. $\varepsilon$ denotes the null sequence.)

In general, the $C\text{-}Pred_e(o)$ is derived as follows:

(1) First, when the user specifies that the object $e$ is an example, the *default candidate predicate* $p[o\langle v\rangle]$ is derived. Here, $p$ is the same as $path(e)$ except that its PAGE.$i$ is replaced by PAGE.?⟨PAGE.$i$⟩, and $v$ is $value(e)$. For example, consider Operation (2) in Sec. 4. When the user specifies the object &11 (with its value "<a>Mike</a>") as an example, the default candidate predicate derived is D.2→PAGE.?⟨PAGE.1⟩→ label.1→ artists.1→ artist.1[$o\langle$<a>Mike</a>$\rangle$]. The predicate defines the set of objects appearing at the same position on different pages.

(2) An 'Another' operation modifies the candidate predicate to accept the 'Another' object. The system has a set of rules [5] that modifies the candidate predicate, according to the given 'Another' example $a$. The basic idea behind the rules is to place a wildcard at the position where $path(e)$ (the path incorporated as the annotation in the candidate predicate) and $path(a)$ conflict with each other.

For example, in Operation (2), the user specifies the object &12 (with its value "<a>Paul</a>"), and the object &28 (with its value "<a>John</a>") as two 'Another' objects. The system matches and generalizes the three paths from the root to three objects (Fig. 2), and outputs the predicate in the above $CS_{\&11}$.

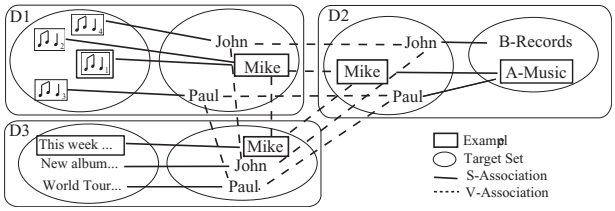



Figure 8. Target sets and associations

| Ex.Mike in D1 | Ex. ♫♩ in D1 | Ex.Mike in D2 | Ex. A-Music in D2 | Ex.Mike in D3 | Ex. This week... in D3 |
|---|---|---|---|---|---|
| Mike | ♫♩$_1$ | Mike | A-Music | Mike | This week... |
| Mike | ♫♩$_2$ | Mike | A-Music | Mike | This week... |
| Paul | ♫♩$_3$ | Paul | A-Music | Paul | World Tour... |
| John | ♫♩$_4$ | John | B-Records | John | New album... |

Figure 9. Target relation

### 5.2 Target Sets, Target Relation, and the Web-site

Target set $TS_e$ of an example $e$ is defined based on the Target relation. So we define the target relation first. Intuitively, a target relation $TR$ represents the target sets of examples and the associations among them. It is defined as $TR = \sigma_{p_0}(CS_1 \bowtie_{p_1} \ldots \bowtie_{p_{n-1}} CS_n)$ where $p_0$ represents selection conditions specified in 'Clue' operation, and $p_i$s are join conditions representing associations. For example, v-associations are represented by $value(e_1) = value(e_2)$ which means that values of objects $e_1$ and $e_2$ are the same. Fig. 9 shows $TR$ based on Fig. 8. Then, the target set $TS_e$ for an example $e$ is defined as follows:

$$TS_e = \pi_{attr(CS_e)}(TR)$$

where $attr(CS_e)$ denotes the attribute in $TR$ corresponding to $e$.

Then, the system applies the Nest and Projection operator [4] to $TR$, based on the position of repetition marks (*) on the Canvas. In the example in Fig. 7, the following expression produces the nested relation shown in Fig. 10.

$$\nu_{V=(\underline{\text{♫♩}})}(\pi_{\underline{Mike},\underline{\text{♫♩}},\underline{A-Music},\underline{Thisweek..}}(TR))$$

| Ex.Mike in D1 | V<br>Ex.  in D1 | Ex.A-Music in D2 | Ex. This week... in D3 |
|---|---|---|---|
| Mike |  | A-Music | This week.. |
| Paul |  | A-Music | World Tour.. |
| John |  | B-Records | New album.. |

Figure 10. Result nested relation

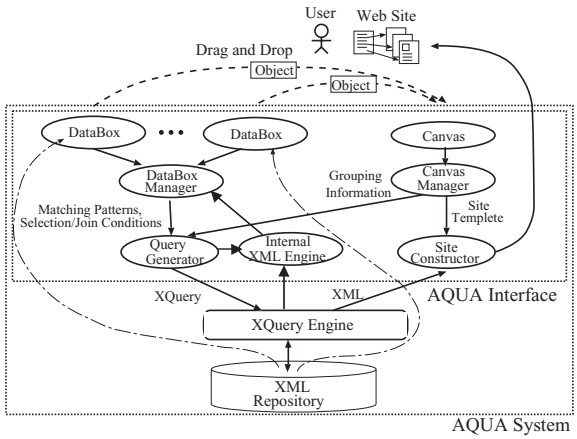



Figure 11. System architecture

The system outputs the result as a set of Web pages (Fig. 1(d)(e)).

## 6. Implementation

Fig. 11 shows the architecture of the AQUA system. Users communicate with the system through GUI components, namely, *DataBox* and *Canvas*. After a user drag-and-drops data objects from the DataBoxes to the Canvas in order to show the system some example Web pages, the DataBox Manager produces matching patterns (candidate predicates) and selection/join conditions (associations) against the underlying data, based on users' interaction with DataBoxes. The Canvas Manager produces *grouping information* and *site template*. The matching patterns, selection/join conditions and grouping information are fed into the *query generator* to produce an XQuery query. The internal XML engine is used to compute target sets. (Details are explained in Subsec. 6.2.) The *site constructor* embeds the query result from the XQuery engine into the site template and produces the result Web-site.

The system (Fig. 12) is implemented in Java (Java2 SDK1.3) and utilizes its drag and drop API. The code is about 21,000 lines. Quip [7] is used as an XQuery engine.

### 6.1 Query Generation

The AQUA system generates an XQuery query that computes the nested relation (in an XML format) explained in Sec. 5. The translation is always possible, because

Figure 12. AQUA prototype system

```
let $TR := <result>{
for $p1 in //D1/PAGE/song,
  $o1 in $p1/artist, $o2 in $p1/video-clip,
   $p2 in //D2/PAGE/label,
    $o3 in $p2/l-name, $o4 in $p2/artists//artist
     $p3 in //D3/PAGE/artist,
      $o5 in $p3/name, $o6 in $p3/news,
       $o7 in $p3/profile/debut
where $o1=$o3 and $o1=$o5 and $o7=2000
return <tr><o2>{$o2}</o2><c><o3>{$o3}</o3>
          <o4>{$o4}</o4><o6>{$o6}</o6></c></tr>
}</result>
let $result := <result>{
 for $ta in distinct-values($TR//tr//c)
 return <item>{$ta/o4}{$ta/o3}{$ta/o6}<rep>{
  for $tb in $TR//tr
  where $ta = $tb//c
  return <item>{$tb/o2}</item>
 }</rep></item>
}</result>
return $result
```

Figure 13. Generated query

the expressive power of XQuery includes path navigation, joins, and the Nest operation. One possible way to implement the computation in XQuery is to use nested `for` clauses and a `where` clause to construct the target relation, and then use nested subqueries to implement the Nest operation.

Fig. 13 shows the XQuery query that computes the nested relation shown in Fig. 10. (document, data, and text functions are omitted.) S-Associations are implemented by variables that keep least-common-ancestors of those objects that are associated with one another. For example, $\$p2$ is used to guarantee that the query makes correct pairs of an artist ($\$o4$) and a l-name ($\$o3$).

### 6.2 Problems and Our Solutions

Although the algorithm is well-defined at the data model level, it is not straightforward to construct a practical system based on it. As explained in Sec. 3., the system has to highlight objects in an inferred target set in order to ask the user to check if the target set is correct. Each time to compute objects in the target set, the system has to execute a query against the entire data instance, as explained in Subsec. 5.2. We have to implement the query in XQuery, but there are problems on performance and OIDs:

**(A) Performance.** The computation of target sets requires a global processing of data; The system has to scan the entire document in general, which can results in a long processing time to obtain target sets. This is undesirable, because the system provides a user interface where the response time is critical.

An important observation is as follows: We do not need the entire target set at the same time because what we can see in a DataBox is a small portion of data. The system takes advantages of this property. For example, assume that (1) the XPATH for a target set is `/labels/label/l-name`, (2) a page corresponds to a `label` element, and (3) the DataBox shows the third page now. Then, the system re-writes the XPATH in a query into `/labels/label[3]/l-name`. This additional condition leads to a very small result in general, and gives the XQuery engine a possibility of finding efficient plans.

**(B) OID Problem.** The data model assumes that all objects have OIDs. While the XQuery's semantics also assumes the property, OIDs are implicit, i.e., the result of the XQuery query has no explicit OIDs. Therefore, the system cannot know which objects in a DataBox are contained in the query result, because it is possible that more than one object in a DataBox have the same value.

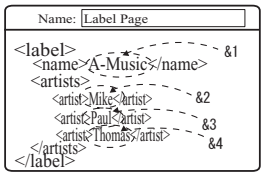
The internal XML query engine in Fig. 11 is used to connect objects in the result from XQuery engines to the objects shown in a DataBox. This is done as follows: Remember that the target relation $TR$ is the result of join of candidate sets based on associations (i.e., $TR = \sigma_{p_0}(CS_1 \bowtie_{p_1} \ldots \bowtie_{p_{n-1}} CS_n)$). Assume that we want to know which objects in a DataBox are contained in $TS_k = \pi_{attr(CS_k)}(TR)$ (the target set of example $e_k$). First, we define $TR_k^{s-assoc*}$ as follows:

$$TR_k^{s-assoc*} = CS_{k_1} \bowtie_{p_{k_1}} \ldots \bowtie_{p_{k_{m-1}}} CS_{k_m}$$

where $CS_{k_i}$ is a candidate set whose example is reachable from $e_k$ through S-Associations. In Fig. 8, if $e_k$ is "Mike" in D2, involved are the candidate sets for $e_k$ itself and "A-Music." Note that $TS_k \subseteq \pi_{attr(CS_k)}(TR_k^{s-assoc*})$ because $TS_k$ considers other associations and selection conditions. The idea here is to use the internal engine to (1) assign temporary, unique OIDs (none of which are the same) to objects in a DataBox (Fig. 14 (left)), (2) compute $TR^{s-assoc*}$ with the OIDs (Fig. 14 (right)), and (3) join the result with $TR$ computed by the outside XQuery engine. We can get the set of (temporary) OIDs in $TS_k$ (in the example, {&2, &3} ) with the following query:

$$\pi_{oid_k}(\pi_{oid_k,c}(TR_k^{s-assoc*}) \bowtie_c \underline{\pi_c(TR)})$$

where the underlined sub-expression is computed by the outside XQuery engine, and $c$ is the set of attributes that are used as join conditions between $TR^{s-assoc*}$ and $TR$. The framework works, because the $TR^{s-assoc*}$ is connected to $TR$ only through V-Associations. Note that the cost to compute $TR_k^{s-assoc*}$ is relatively small and so are the size of $\pi_{oid_k,c}(TR_k^{s-assoc*})$ and $\pi_c(TR)$, because we can use the technique explained in Problem (A).




| Ex.Mike | $Oid_1$ | Ex.A-Music | $Oid_2$ |
|---|---|---|---|
| Mike | &2 | A-Music | &1 |
| Mike | &2 | A-Music | &1 |
| Paul | &3 | A-Music | &1 |
| Thomas | &4 | A-Music | &1 |

Figure 14. Object identification mechanism

## 7. Conclusion

This paper has explained a novel tool for construction of data-intensive Web sites. It is unique in that unlike other tools, it requires users to show the system some examples of Web pages, and tries to infer his intention and construct the entire Web site. This is a non-trivial task, because the system can cope with XML (semistructured) data. The paper also explains the implementation. A key feature is that the system can work with general XQuery engines, exploiting the possible robust and efficient performance. We have explained mechanisms to cope with problems that arise when we want to implement a practical system. Future works include development of mechanisms to support (1) dynamic Web sites and (2) reuse of the past operations.

## References


[1] S. Abiteboul. Querying semi-structured data. *Proc. 6th International Conference on Data Theory (ICDT'97)*, pp. 1-18, 1997.

[2] Peter Buneman. Semistructured data. *Proc. 16th ACM Symposium on Principles of Database Systems (PODS'97)*, pp. 117-121, 1998.

[3] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the Boat with Strudel:Experiences with a Web-Site Management System. *Proc. ACM SIGMOD'98*, pp.414-425, 1998

[4] P. C. Fischer and S. J. Thomas. Operators for non-first-normal-form relations. *Proc. IEEE COMPSAC83*, pp. 464-475, 1983.

[5] A. Morishima, S. Koizumi and H. Kitagawa. Drag and Drop: Amalgamation of Authoring, Querying, and Restructuring for Multimedia View Construction. *Proc. 5th IFIP 2.6 Working Conference on Visual Database Systems(VDB5)*, pp. 257-276, 2000.

[6] A. Morishima, S. Koizumi, H. Kitagawa, and S. Takano. Enabling End-users to Construct Data-intensive Web-sites from XML Repositories: An Example-based Approach. *Proc. 27th VLDB Conf.*, pp. 703-704, 2001.

[7] Software AG. QuiP. http://www.softwareag.com/.

[8] W3C. Synchronized Multimedia Integration Language(SMIL). http://www.w3c.org/AudioVideo, 2002.

[9] W3C. XQuery 1.0: An XML Query Language. X3C Working Draft, http://www.w3.org/TR/xquery, 2002.

[10] M. M. Zloof. Query-by-Example: a Data Base Language. *IBM Systems Journal*, Vol. 16, No. 4, pp. 324-343, 1977.