Navigator-based Query Processing in the World Wide Web Wrapper

Kazunori Katoh† Atsuyuki Morishima†† Hiroyuki Kitagawa††

†Doctoral Program in Engineering, University of Tsukuba
††Institute of Information Sciences and Electronics, University of Tsukuba

Abstract

Integration of heterogeneous information sources has been one of the most important issues in recent advanced application environments. We are developing an information integration environment for the World Wide Web (or shortly, Web), relational databases, and structured document repositories. In this environment, manipulation of the information sources is performed through software modules called wrappers. In this paper, we present design and implementation of the Web wrapper. In general, Web page manipulation may cause very large data transfer cost if all the necessary pages are transferred to the Web wrapper. To alleviate this problem, we propose a query processing scheme which uses distributed agents, named navigators. The navigators cooperatively take part of the wrapper's functions at Web server sites, to reduce the cost of Web page transfer. This paper includes experimental results on data transfer costs in this scheme.

1 Introduction

Today, a huge number of information sources are available via computer networks. Thus, integration of heterogeneous information sources has been one of the most important issues in recent advanced application environments [2][3][7]. In particular, with the broad acceptance of the World Wide Web (or Web), there is a great deal of demand for integration of the Web and other information sources.

We are developing an information integration environment for the Web, relational databases, and structured document repositories [8][11][12][13]. Web pages are usually written in XML or HTML. Structured document repositories are assumed to contain documents written in SGML, XML, or HTML.

This environment uses software modules called *mediators* and *wrappers* [17][19] to attain integration (Figure 1). This is one of the promising approaches to integrate heterogeneous information sources. Many integration systems follow this approach [4][16].

In this approach, wrappers are associated with information sources, and the mediator acts as a coordinator. Integration of information sources is attained as follows. First, wrappers translate information sources into a common data model (in our environment, WebNR/SD data model), so that users can query and browse data uniformly through the mediator. If users submit a data manipulation request to the mediator, the mediator analyzes the request, decomposes it into local processing requests, and sends them to wrappers. Each wrapper translates requests into local commands, and issues them to the local information source. Local commands may be simple requests to get particular data items such as Web page fetches, or complicated queries which utilize querying capability of local information sources, such as SQL queries. The wrapper receives the intermediate result, translates it into the common data model, and sends it back to the mediator. Finally, the mediator collects data from the wrappers and produces the final result, which is returned to the client.

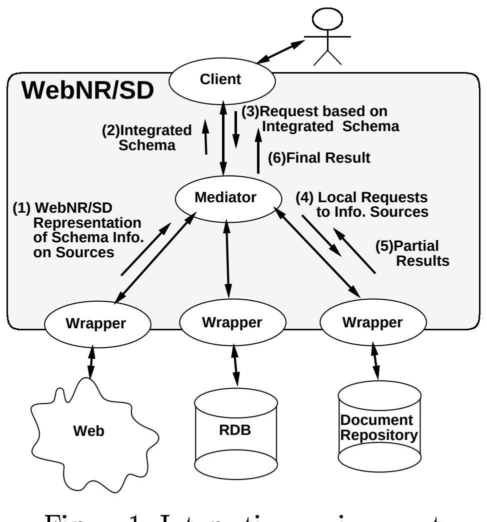
WebNR/SD
Client
(2)Integrated Schema
(3)Request based on Integrated Schema
(6)Final Result
Mediator
(1) WebNR/SD Representation of Schema Info. on Sources
(4) Local Requests to Info. Sources
(5)Partial Results
Wrapper
Wrapper
Wrapper
Web
RDB
Document Repository

Figure 1. Integration environment

In this paper, we describe an architecture of the Web wrapper. As mentioned in [1] and [14], *navigation* is one of the essential operations for the Web. Navigation is to find particular paths (or pages) by traversing hyper-text links from a given starting point. Many Web query languages [9][10][15] support *navigational queries*, which find particular paths (or pages) based on a given linking pattern (such as a regular expression and logical constraints).

In our environment, the Web wrapper is responsible for Web operations including navigation (and navigational queries). To implement the operations, the Web wrapper sometimes has to do a lot of work, since the Web itself only provides the very basic page fetch mechanism and has no query processing capability. For example, navigational queries need to successively follow hyper-text links occurring in Web pages. Therefore, all the Web pages traversed in the navigational queries need to be transferred to the wrapper from distant Web server sites. In general, this causes very large data transfer cost.

To alleviate this problem, we propose a query processing scheme which employs distributed agents named *navigators*. A navigator is dispatched to a Web server site and is responsible for obtaining qualified (partial) paths in the scope of the server. It is only qualified path information that navigators transfer to the Web wrapper rather than the whole contents of accessed pages. Thus, navigators can bring about reduction of the data transfer cost.

To the best of our knowledge, no efficient implementation scheme for Web navigational queries has been proposed, and no experimental result has been reported.

The remainder of this paper is organized as follows. In Section 2, we explain WebNR/SD, the common data model in our integration environment. This section also gives description of how navigational queries can be specified in WebNR/SD. In Section 3, first, we describe functions of the Web wrapper, and then, we propose a query processing scheme which uses navigators to implement navigational queries. In Section 4, we report and discuss experimental results. The experiments shows that the proposed scheme can considerably reduce the amount of data transfer, and that, in many cases, it can reduce the execution time of navigational queries. Section 5 is the conclusion.

2 WebNR/SD

In this section, we overview WebNR/SD focusing on Web-related issues. More detailed description of WebNR/SD is given in [12]. WebNR/SD is a data model which introduces abstract data type concept into nested relations. It has an abstract data type named *SD type* (structured document type) to deal with raw structured documents (SGML, XML, and HTML documents). Also, it has an abstract data type named *Hlink type* to represent hyper-text links which appear in XML and HTML documents. Figure 2 shows a sample relation in WebNR/SD. The domains of attributes A and D are String and Integer, respectively. The domain of B is SD type, and that of E is Hlink type.

A	B	C	
		D	E
abc	`<table><dep>` `Department...`	1	`<a href="http://..">...`
		2	...
def	...	3	...
		4	...

Figure 2. Sample relation in WebNR/SD

WebNR/SD provides the nested relational algebra operators, and a number of functions associated with SD type to retrieve text elements contained in documents. Also, WebNR/SD has operators, called *converters*, to dynamically convert structured documents into nested relational structures and vice versa.

Moreover, WebNR/SD has a number of operators for Web integration. *Navigate* operator is used to realize navigational queries in the Web. *Import* operator is used to fetch the contents of Web pages as SD type values from the outside Web world on demand. In contrast, *Export* operator exports SD type values stored in relations as Web pages to the outside Web world.

Integration of Web, structured documents and relational databases is achieved by combining the above operators.

2.1 SD type and Hlink type

A value of SD type is a pair of a *DTD* (Document Type Definition) and text in which tags are embedded according to the DTD. We call a value of SD type an *SD value*[1].

Figure 3 shows an example SD value. The DTD in the upper box represents the document structure. Inside the lower box is the tagged text. A tagged text is divided into *elements* surrounded by a *start tag* <*g*> and an *end tag* </*g*>.

```
report   = seq(title, authors, body, ref)
authors  = rep(author)
author   = seq(name, homepage)
homepage = hlink
body     = rep(section)
section  = seq(sectitle, rep(para))
   ...
```
```
<report><title>On Web Integration</title>
<authors><author><name>Thomas</name>
<homepage href="http://T.ac.jp/p1.xml">His home
page</homepage></author> ... </authors>
<body><section><sectitle>Introduction</sectitle>
<para> ...
```

Figure 3. Sample SD value

Elements with **hlink** structure are called *linking elements* and represent hyper-text links to Web pages just as anchor elements in HTML documents. The **hlink** structure is associated with the attribute "href," whose value designates a URL. For example, "`<homepage`

[1]The DTD part can be NULL when the SD value corresponds to an HTML or XML document without DTD.

`href="http://T.ac.jp/p1.xml">His home page </homepage>`" in Figure 3 is a linking element.

A value of Hlink type (an *Hlink value*) is defined as an SD value which consists of only one linking element.

2.2 Web-Related Operators

Export and Import Operators

Export (**E**) exports SD values stored in a relation to the Web world as Web pages. Import (**I**) imports Web pages residing in the Web world into a relation as SD values. Figure 4 gives an example of Export and Import, where

$$r_2 := \mathbf{E}_{A,U,L,G}(r_1),$$

and

$$r_1 := \mathbf{I}_{A,U,L,G}(r_2).$$

$\mathbf{E}_{A,U,L,G}(r_1)$ creates Web pages whose URLs are given in attribute U. The Web pages' contents correspond to SD values stored in attribute A of r_1. Attribute A in the result relation r_2 has Hlink values referring to those pages and containing character strings originally stored in attribute L. $\mathbf{I}_{A,U,L,G}(r_2)$ works in the opposite direction. Attribute A in the result relation r_1 obtains SD values corresponding to Web pages which are referred to by Hlink values stored in attribute A of r_2.

r_1:

ID	A	U	L	G
1	⟨ e:**seq**(f:**rep**(g),h), "`<e><f><g>T4</g> </f><h>T5</h></e>`"⟩	http://T.ac.jp/p.xml	T Univ	a

r_2:

ID	A
1	⟨ a:**hlink**, "`<a href="http://T.ac.jp/p.xml"> T Univ </a>`" ⟩

Figure 4. Example of Export and Import

Navigate Operator

Navigate (**N**) realizes navigational queries. That is, this operator navigates through the Web according to a *path regular expression* specified as a parameter. In path regular expressions, hyper-text links are represented by $\rightarrow$ (a local link whose destination and source pages reside in the same Web server) and $\Rightarrow$ (a global link whose destination and source pages reside in different Web servers), while Web pages are represented by character strings (*labels*) and a period. For example, $B \rightarrow . \Rightarrow C \rightarrow D$ is a path regular expression that represents the set of paths that start with a Web page B in a Web server X, followed by a page in the same server X and a page C in a different server $Y(\neq X)$, and end with a page D in the server Y.

Path regular expressions can contain alternation and repetition structures. For example, $A(\rightarrow .)* \rightarrow B \;|\; A \Rightarrow B$ represents the set of paths which start with a page A, followed by one or more local links leading to B, or from a page A to B via a global link. '$*/n_1..n_2/$' can be used as syntactic sugar. For example, $A(\rightarrow .) * /2..3/ \rightarrow B$ is equivalent to $A \rightarrow . \rightarrow . \rightarrow B \mid A \rightarrow . \rightarrow . \rightarrow . \rightarrow B$. Path regular expressions can also specify word containment conditions on Web pages[2]. The condition is placed just after a label or a period. Figure 5 shows the BNF specification of path regular expressions[3].

```
path_reg_expr ::= label p_expr {'|' label p_expr}
p_expr        ::= link page [ '[' condition ']' ]
                | p_expr { p_expr }
                | p_expr { '|' p_expr }
                | p_expr '*' ['/'num'..'num'/']
                | '(' p_expr ')'
link          ::= '− >' | '=>'
page          ::= label | '.'
```

Figure 5. Path regular expression syntax

Suppose that there is a hyper-text link structure shown in Figure 6 in the Web. In this figure, each Ui stands for a URL, α, β, γ are Web server names, and lower-case letters show linking elements. Figure 7 gives the result of the following expression:

$$r_4 := \mathbf{N}_{A(\rightarrow.)*\rightarrow.\Rightarrow B(\rightarrow.)*\rightarrow C[''paper''],D}(r_3).$$

Labels in path regular expressions are used to associate Web pages with relational attributes. The second parameter of Navigator operator (in this case, D) specifies that attribute D of the result relation contains the set of paths specified by the path regular expressions. Figure 7 shows two important points: (1) Each page on the paths is represented by an Hlink value referring to the Web page instead of the contents of the page itself. (2) Links to those pages corresponding to the period do not appear in sub-relations of attribute D.

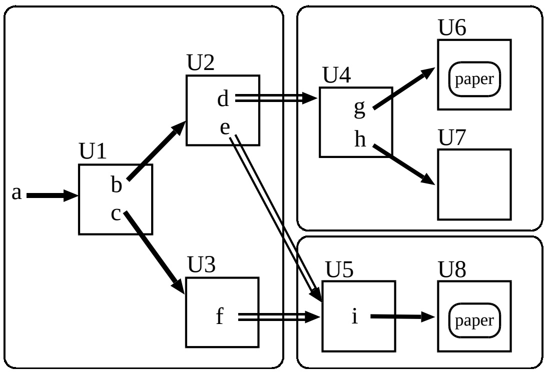

Figure 6. Example hyper-text link structure

[2]Actually, the conditions can be more complex ones based on region algebra. Details are omitted here.

[3]We assume that every path specified by a path regular expression gives non-empty binding to all the labels. $A \rightarrow B \mid A \rightarrow C$ is an example which violates the restriction.

r_3:

ID	A
1	⟨ a:**hlink**, "<a href="U1">a</a>" ⟩

r_4:

ID	A	D	
		B	C
1	⟨ a:**hlink**, "<a href="U1"> a</a>" ⟩	⟨ a:**hlink**, "<a href="U4"> d</a>" ⟩	⟨ a:**hlink**, "<a href="U6"> g</a>" ⟩
		⟨ a:**hlink**, "<a href="U5"> e</a>" ⟩	⟨ a:**hlink**, "<a href="U8"> i</a>" ⟩
		⟨ a:**hlink**, "<a href="U5"> f</a>" ⟩	⟨ a:**hlink**, "<a href="U8"> i</a>" ⟩

Figure 7. Example of Navigate operator

The path regular expressions are essentially the same as those of WebSQL queries[4].

3 Web Wrapper

In the integration environment, the Web wrapper is placed on the site where the mediator works. The Web wrapper provides the mediator with the following functions.

1. It creates relational views on top of the Web: In WebNR/SD, the Web is modeled as a set of collections of Hlink values (i.e. unary relations with an Hlink type attribute). For example, bookmarks in Web browsers can be used to construct such unary relations. The Hlink values work as anchor points to fetch necessary Web pages with Navigate and Import operators.

2. It executes Export, Import, Navigate operators: When the mediator applies these operators to relational views provided by the Web wrapper, the Wrapper is responsible for their execution[5].

We focus on execution of the Navigate operator here. The most basic scheme that executes the Navigate operator would be that the Web wrapper performs everything. In the following discussion, first, we show the basic scheme and its problem. Then, we propose a more sophisticated scheme which utilizes distributed agents.

3.1 Basic Scheme for the Navigate Operator

According to the path regular expression specified as a parameter of Navigate operator, the wrapper can construct a finite state automaton. The wrapper fetches Web pages and uses the automaton to determine if each path is qualified for the specified path regular expression.

Figure 8 shows the finite automaton for path regular expression $A(\rightarrow .)* \rightarrow . \Rightarrow B(\rightarrow .)* \rightarrow C[''paper'']$. In this figure, states are annotated with character strings or numbers. The character strings correspond to the labels in the path regular expression. State transition can have three types of conditions. They are local links, global links, and word containments, represented by $\rightarrow$, $\Rightarrow$, and $\overset{word}{\rightharpoonup}$, respectively. Their meanings are the same as the counterparts in path regular expressions.

In the basic scheme, the Web wrapper can obtain the set of qualified paths by fetching all the Web pages along the candidate paths and examining whether each path is accepted by this automaton. However, this scheme causes very large data transfer costs since all the Web page contents are transferred from distant Web servers to the Web wrapper.

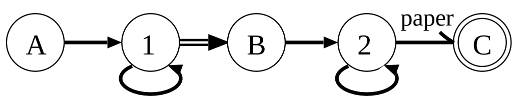

Figure 8. Automaton corresponding to $A(\rightarrow .)* \rightarrow . \Rightarrow B(\rightarrow .)* \rightarrow C[''paper'']$

3.2 Utilization of Distributed Agents

To alleviate this problem, we propose a more sophisticated scheme which employs distributed agents.

The scheme uses agents named *navigators*. A navigator is dispatched to a Web server site and is responsible for obtaining qualified (partial) paths in the scope of the server (Figure 9). In contrast to the basic scheme, it is only qualified path information that navigators transfer to the Web wrapper. Therefore, this can reduce the data transfer cost.

In order to realize this scheme, the wrapper decomposes the finite state automaton for the whole path regular expression into a number of sub-automatons. The decomposed automatons are incorporated into different navigators, so that each navigator can determine if paths are qualified for the partial path regular expression in the scope of a single Web server.

[4]WebSQL permits other link types such as an interior link. And it permits no word containment condition in path regular expressions. However, such differences have no effect on our discussion.

[5]In case the mediator applies these operators to relations materialized in the mediator, it calls the Web wrapper functions to perform required data transformation tuple-at-a-time.

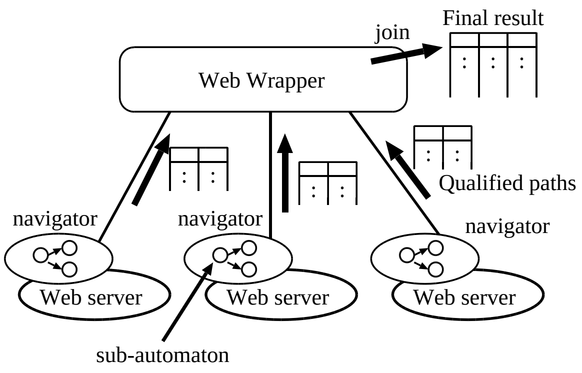

Figure 9. Utilization of navigators

Decomposition of an Automaton

We represent the automaton as X and a sub-automaton as X_i. Let $I = \{IS(X)\} \cup GS(X)$, where (1) $IS(X)$ is the initial state in X, and (2) $GS(X)$ is the set of states gs in X such that gs is the destination of some global link in X. The states in I are used as initial states in sub-automatons. Also, let $F = GS(X) \cup AS(X)$, where $AS(X)$ is the set of accepting states in X. The states in F are used as accepting states in sub-automatons. Then, X_i is defined as an automaton such that (1) its initial state is some state in I, and (2) it has states in X reachable from its initial state through state transitions which do not go over any states in F (as mentioned above, states in F become accepting states in X_i). If an accepting state in X_i does not belong to $AS(X)$, we call it an *intermediate accepting state*. Also, we refer to the sub-automaton which has $IS(X)$ as the *root automaton*.

Figure 10 shows the result of decomposition of the automaton in Figure 8. In this case, $IS(X) = A$, $GS(X) = \{B\}$, $AS(X) = \{C\}$, $I = \{A, B\}$, and $F = \{B, C\}$. As a result, two automatons X_1 and X_2 are obtained by the decomposition. X_1 is the root automaton, and B is an intermediate accepting state.

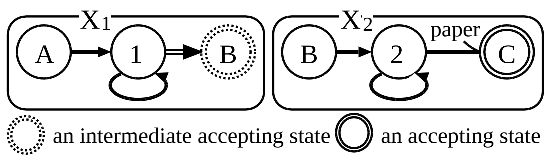

Figure 10. Sub-automatons

Navigation Algorithm

Now, we show how to create and dispatch navigators. First, when a Navigate operator is executed, sub-automatons are constructed by decomposing the automaton for the path regular expression. Then, the Web wrapper creates a navigator which has the root automaton. The Web wrapper dispatches it to Web server α which manages the starting Web page (i.e. the page referred to by a starting Hlink value). The navigator obtains the set of qualified paths originated by the starting Hlink value in the scope of α, and it records the result paths in relations (tables). If the result paths contain a path which reaches an intermediate accepting state s in the root automaton, it indicates that the next Web page is managed by a different Web server $\beta (\neq \alpha)$ (note that the final transition is always caused by a global link if the accepting state is intermediate one). The navigator sends this information to the Web wrapper. Then, the wrapper creates another navigator whose automaton has s as its initial state, and dispatches it to the site of Web server β. In case that the site already has a navigator associated with the same automaton, the navigator is reused and no new navigator is created. Then, the new (or reused) navigator receives an Hlink value as a starting point, and works in the same way. When a path reaches some accepting (not intermediate) state, it is considered to be qualified for the whole path regular expression. This process terminates when there is no further hyper-text link to follow[6].

Note that before a navigator records some partial path into a relation, it checks whether there already exists the same partial path in the relation. If duplicates exist, no redundant traversal of the path is launched. Similarly, infinite iteration is avoided when a navigator works on a Web server. Therefore, the algorithm terminates even if cyclic structures appear in the Web.

Example

We illustrate the above algorithm by an example. Consider $\mathbf{N}_{A(\rightarrow.)*\rightarrow.\Rightarrow B(\rightarrow.)*\rightarrow C[''paper''],D}(r_3)$ (r_3 is given in Figure 7). Then, sub-automatons are X_1 and X_2 in Figure 10. The starting Hlink value in r_3 refers to a Web page at URL U1. Assume that relevant Web pages are linked and managed as shown in Figure 6.

Figure 11 shows how the Web wrapper creates navigators and passes Hlink values to them in this example. In this figure, α, β, and γ are Web servers. Navigators are represented by circles. X_1 and X_2 are sub-automatons associated with the navigators.

First, the Web wrapper creates a navigator with the root automaton X_1, at Web server α where a starting web page exists. While obtaining the paths starting with Hlink value a in the scope of Web server α, the navigator finds that the paths referred to by Hlink values d, e, and f must be checked by automaton X_2. Because d refers to a Web page in Web server β, and because e and f refer to a Web page in Web server γ, the Web

[6]Of course, we can set some restriction on the navigation process by specifying time limit or resource consumption limit. In this paper, we omit the discussion on this.

wrapper dispatches navigators with automaton X_2 to β and γ, and passes the Hlink values to them. The two navigators check paths in order to obtain the final result.

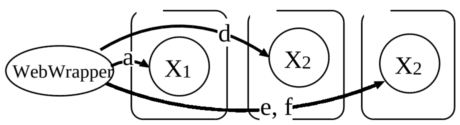

Figure 11. Creation of navigators

As mentioned before, each navigator manages relations to store the paths qualified in the scope of the corresponding Web server it resides. One relation is created for each accepting state in a sub-automaton. Figure 12 shows the relations managed in the above example. Consider relation $\alpha\_X_1\_B$. It is managed by the navigator, with sub-automaton X_1, which is dispatched to Web server α. It stores the set of paths accepted by intermediate accepting state B. Other two relations ($\beta\_X_2\_C$ and $\gamma\_X_2\_C$) are managed in a similar way. Attribute names of the relations fall into two groups: (1)Those which correspond to labels of states in sub-automatons, and (2) Join attributes (X_1-X_2 in this example). The former attributes have Hlink values which cause transition into states annotated with the corresponding labels. Join attributes have Hlink values which cause transition into intermediate accepting states (in the case of the right-most attribute), or values used as starting Hlink values to compute paths (in the case of the left-most attribute).

The final result (the set of paths qualified for the whole path regular expression) is obtained by joining these relations.

$\alpha\_X_1\_B$

A	B	X_1-X_2
a	d	d
a	e	e
a	f	f

$\beta\_X_2\_C$

X_1-X_2	C
d	g

$\gamma\_X_2\_C$

X_1-X_2	C
e	i
f	i

Final result

A	B	C
a	d	g
a	e	i
a	f	i

Figure 12. Relations managed by navigators and the final result

4 Experiments

In this section, we compare the proposed navigator-based scheme with the basic scheme in terms of the amount of data transfer and execution time.

The Web wrapper and the navigator have been written in Java. JDK1.1.5 is used as the Java virtual machine. Apache 1.2.6 is used as the Web server. Communication between the Web server and the Web wrapper/navigator is performed based on HTTP. On the other hand, communication between the wrapper and the navigator is performed through HORB (Hirano's Object Request Broker) 1.3.b1, which provides the Java-based distributed computing environment [5][6]. We use two computers (C1 and C2) located at distant places. C1 is at University of Tsukuba, Ibaraki, Japan, and C2 is at Kanagawa Institute of Technology, Kanagawa, Japan. They are the same type of workstations (Sun Ultra 1 with Ultra Sparc 143MHz CPU, and Solaris 2.6). C1 has 64MB memory and C2 has 32MB memory.

4.1 Experiment Data and Procedure

For the experiments, we construct eight sets of Web pages. They differ from each other in the total number of non-root Web pages (50, 100, 200, 400, 800, 1200, 1600, or 2000 pages in each set). In each set, Web pages form a tree whose height is three (Figure 13). It has one root page which has local links to a number of intermediate pages. Each intermediate page has 49 local links to leaf pages. Each leaf page has no link.

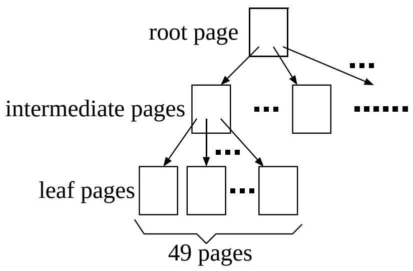

Figure 13. Structure of a set of Web pages

We run the Web wrapper at one site, and place the Web server and the Web pages in the other site. Then, we request the Web wrapper to process Navigate operator applied to a relation which only contains an Hlink value referring to the root Web page. The Navigate operator has the path regular expression $A(\rightarrow .)* \rightarrow B$ as its parameter. Therefore, the result includes all the paths starting with the root page and ending with an intermediate page or a leaf page.

The experiments were done under the basic scheme and the proposed navigator-based scheme with one navigator (Figure 14). We measured the amount of data transfer and execution time. The amount of data transfer was obtained by measuring the total size of packets which the Web server or HORB daemon transfers. The execution time was calculated from the start time and the end time of Navigate operation. We disabled the caching in the Web server.

We performed the same experiment in the following configurations: (a) Site A: C_1 at Univ. of Tsukuba, Site B: C_2 at Kanagawa Institute of Technology (b) Sites A and B are exchanged. There was no significant difference between the two configurations. The experimental results about the execution time shown below are the average of the two configurations.

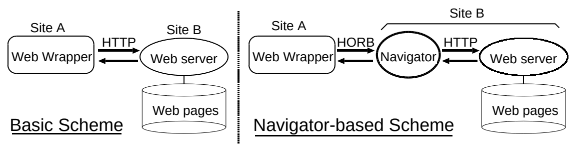

Figure 14. Experiment schemes

4.2 Results and Discussions

4.2.1 The Amount of Data Transfer

Figure 15 shows the amount of data transfer in the basic scheme and the navigator-based scheme. It shows that utilization of the navigator can considerably reduce the amount of data transfer. The data transfer in the navigator-based scheme is only 3.4% of that in the basic scheme on average.

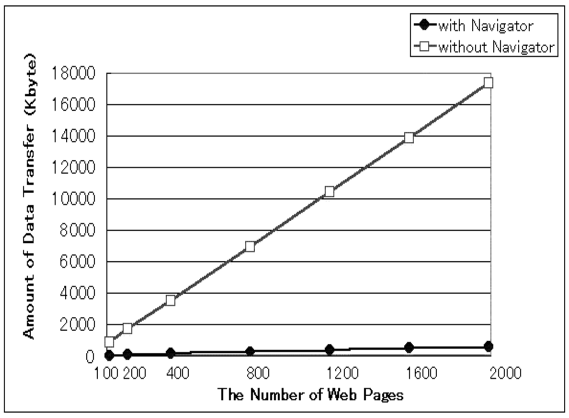

Figure 15. Data transfer cost

4.2.2 Execution Time

Considering difference in the traffic condition of the network, we measured the execution time in the daytime and at midnight on weekdays. Each point in the graphs shows the average of six trials (three trials each in configurations (a) and (b)).

Daytime Result

Figure 16 shows that the navigator-based scheme is superior to the basic scheme regardless of the number of Web pages read from the Web server. The average reduction in execution time is 30%.

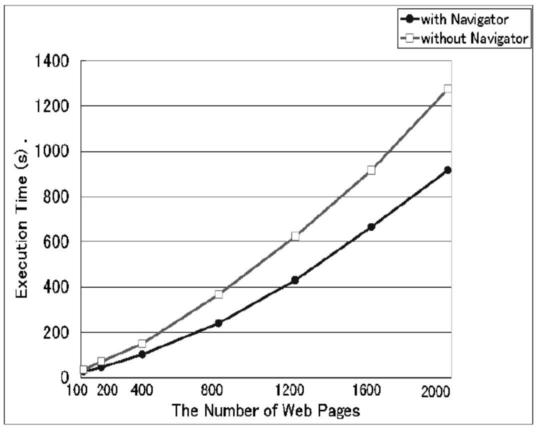

Figure 16. Execution time (in the daytime on weekday)

Midnight Result

Figure 17 shows that the average reduction at midnight is 22%, which is smaller than that in the daytime. The reason is that the network load is light at midnight compared with the daytime, and the data transfer takes less execution time.

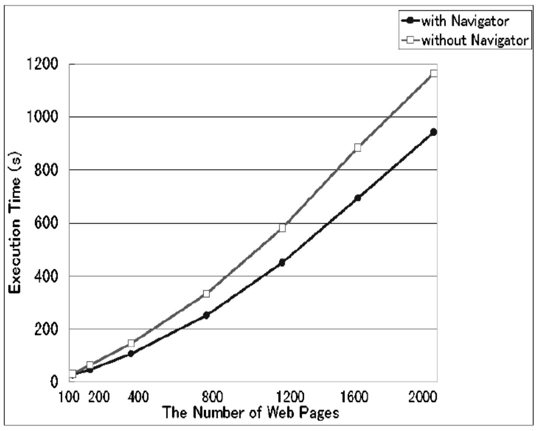

Figure 17. Execution time (at midnight on weekday)

Average of Daytime and Midnight Results

Figure 18 shows the average of the above experimental results. The average of reduction rate is 26%.

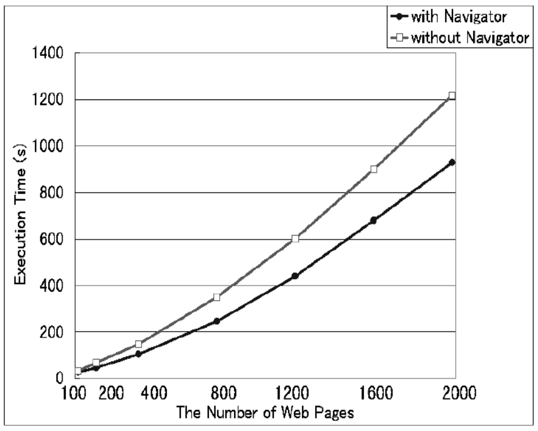

Figure 18. Average of daytime and midnight results

Navigation Referencing a Small Number of Pages

We also measured the execution time in case that the number of Web pages read from the Web server is less than or equal to 10. The data sets used in this experiment are different from those of the above experiments. If a data set has $n(> 1)$ Web pages, it consists of one root page, one intermediate page, and $n - 2$ leaf pages. Figure 19 shows the result. When the number of pages is less than or equal to 5, the execution time in the basic scheme is superior to that in the navigator-based scheme.

The reason is that the overhead for dispatching the navigator is larger than the reduction of page transfer. The overhead is constant when the same number of navigators are dispatched. Therefore, as the number of Web pages grows, the advantage of navigator utilization becomes more evident.

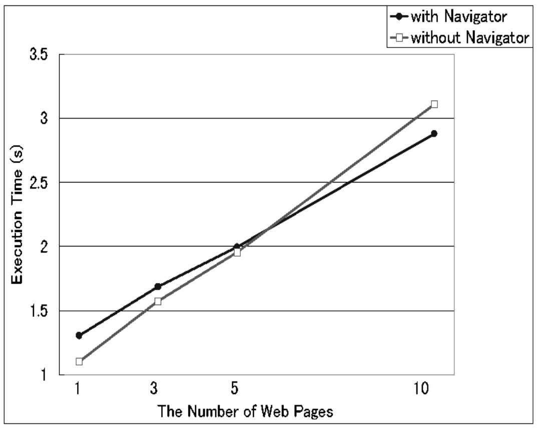

Figure 19. Navigation referencing a small number of pages

5 Conclusions

In this paper, we have presented design and implementation of a Web wrapper in an integration environment for the World Wide Web, relational databases, and structured document repositories. In particular, we have proposed a sophisticated scheme which uses distributed agents named navigators to execute Web navigational queries. We have reported the experimental results in terms of the amount of data transfer and the execution time. The results have shown that the navigator-based scheme can considerably reduce the amount of data transfer, and that, in many cases, the proposed scheme is superior to the basic scheme which uses no navigator.

As shown in the experimental results, difference in the execution time between the basic and navigator-based schemes may be affected by the network load and the number of pages to be referenced. In order to get good execution time, we have to construct a dispatching algorithm of navigators which takes them into account. Also, if we can speed up internal processing in the Web wrapper and navigator, the data transfer reduction caused by navigators would contribute more to the execution time reduction.

In the experiment, we used only one Web server (and only one accompanying navigator). When more Web servers are involved in navigational queries, the proposed scheme is expected to work better, since parallel processing of navigators is possible.

Future research issues include evaluation of the proposed scheme in situations where (1) multiple Web servers exist and (2) real Web sites are involved. They also include utilization of other mechanisms for agent dispatching and communication, such as servlets [18]. This is important in terms of applicability[7] and security[8]. Improvement of the navigation algorithm is another important issue. For example, we need to develop dynamic dispatching algorithms incorporating factors such as network load, and consider optimization of traversing order of paths. These issues will be discussed in forth-coming papers.

Acknowledgement

The authors are grateful to Takayuki Suzuki, Knagawa Institute of Technology, for his support in our experiment. This work was supported in part by the Ministry of Education, Science, Sports and Culture, Japan through the Grand-in-Aid for Scientific Research (#08244101, #09680321), and by the Telecommunications Advancement Foundation.

References

[1] S. Abiteboul and V. Vianu. "Queries and Computation on the Web," *Proc. 6th Intl. Conf. on Data Theory (ICDT'97)*, pp 262-275, 1997.

[2] P. Buneman, S. B. Davidson, K. Hart, and C. Overton. "A Data Transformation System for Biological Data Sources," *Proc. 21st VLDB Conf.*, pp. 158-169, 1995.

[3] A. K. Elmagarmid and C. Pu, (eds.). "Special Issue on Heterogeneous Databases," *ACM Computing Survey*, vol. 22, no. 3, 1990.

[4] Mary F. Fernandez and Daniela Florescu and Jaewwoo Kang and Alon Y. Levy and Dan Suciu. "STRUDEL: A Web-site Management System", *Proc. SIGMOD Conference*, pp. 549–552, Tucson, May 1997.

[5] S.Hirano. "HORB Home Page," `http://ring.etl.go.jp/openlab/horb-j/WELCOME.HTM`.

[6] S. Hirano. "HORB: Distributed Execution of Java Programs," *Proc. Intl. Conf. on Worldwide Computing and It's Applications '97 (WWCA'97)* Tsukuba, pp. 29-42, Mar. 1997.

[7] A. R. Hurson, M. W. Bright, and S. Pakzad, (eds.). "Multidatabase Systems: An Advanced Solution for Global Information Sharing," *IEEE Computer Society Press*, 1994.

[8] K. Katoh, A. Morishima, and H. Kitagawa. "Integration of Heterogeneous Information Sources with WebNR/SD – Design and Development of a Query Processing Scheme for WWW –," *Proc. 56th National Convention on Information Processing Society of Japan (IPSJ)*, pp. 252-253, Mar. 1998 (in Japanese).

[9] D. Konopnicki and O. Shmueli. "W3QS: A Query System for the World-Wide Web," *Proc. VLDB Conf.*, pp. 54-65, 1995.

[10] L. Lakshmannan, F. Sadri, and I. Subramanian. "A Declarative Language for Querying and Restructuring the Web," *Proc. 6th Intl. Workshop on Research Issues in Data Eng. (RIDE'96)*, Feb. 1996.

[7]Currently, servlets are supported by many types of Web servers.

[8]Although what navigators do to web site resources is only to "read" web pages through HTTP, HORB allows inappropriate access to resources if navigators have implementation errors. In contrast, behavior of servlets can be explicitly limited by configuration files of Web servers. Therefore, Web server site administrators can protect their sites.

[11] A. Morishima and H. Kitagawa. "A Data Modeling and Query Processing Scheme for Integration of Structured Document Repositories and Relational Databases," *Proc. 5th Intl. Conf. on Database Systems for Advanced Applications (DASFAA'97)*, pp. 145–154, Melbourne, Apr. 1997.

[12] A. Morishima and H. Kitagawa. "Integrated Querying and Restructuring of the World Wide Web and Databases," *Proc. Intl. Symp. on Digital Media Information Base (DMIB'97)*, pp. 261–271, Nara, Japan, Nov. 1997.

[13] A. Morishima, and H. Kitagawa. "NR/SD+ Data Model and Its Query Procesing — For Integration of Structured Documents and Relational Databases," *Trans. Information Processing Society of Japan (IPSJ)*, Vol. 39, No. 4, pp. 954-967, Apr. 1998 (in Japanese).

[14] A. O. Mendelzon and T. Milo. "Formal Models of Web Queries," *Proc. 16th ACM PODS'97*, pp. 134-143, 1997.

[15] A. O. Mendelzon, G. A. Mihaila, and T. Milo. "Querying the World Wide Web," *Proc. Symp. on Parallel and Distributed Information Systems (PDIS'96)*, pp. 80-91, Dec. 1996.

[16] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. "Object Exchange Across Heterogeneous Information Sources," *Proc. 11th DE Conf.*, pp. 251-260, Mar. 1995.

[17] Mary Tork Roth and Peter M. Schwarz. "Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources", *Proc. VLDB Conf.*, pp. 266–275, Athens, Greece, 1997.

[18] Sun Microsystems, Inc. "Servlets," `http://jserv.javasoft.com/products/java-server/servlets/index.html`.

[19] G. Wiederhold. "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, pp. 38-49, Mar. 1992.