Building HyperView Wrappers for Publisher Web Sites

Lukas C. Faulstich*,[1], Myra Spiliopoulou[2]

[1] Institut für Informatik, Freie Universität Berlin,
<http://www.inf.fu-berlin.de/~faulstic>
[2] Institut für Wirtschaftsinformatik, Humboldt-Universität zu Berlin,
<http://www.wiwi.hu-berlin.de/~myra>

Abstract. Electronic journals are becoming a major source of scientific information. Researchers interested only in certain topics do not have time to scan all possibly relevant journals on a regular basis. A digital library can assist them by providing a uniform, search-able interface for electronic journals. To this purpose, a catalogue of metadata on the available journals such as authors and titles of articles must be established by the digital library. If there is no cooperation with journal publishers, this metadata must be extracted from the publishers' Web Sites, overcoming the intrinsic heterogeneity problems.
Within the framework of the ongoing Natural Sciences Digital Library project at the Free University of Berlin, we have designed a wrapper-mediator mechanism that copes with the heterogeneity problems of automatic metadata acquisition. It is based on our generic HyperView methodology for integration of Web Sites. From this methodology it inherits two elegant and effective features. First, the structure of the publisher site is specified with abstract graph-schemata, instead of being hard-coded in scripts for data acquisition. Second, a powerful view concept based on declarative graph-transformation rules is used for information extraction.

1 Introduction

The number of electronic journals is rapidly increasing. Publishers currently offer several thousand electronic journals. For researchers, the advent of online journal editions has made life much easier since all information is reachable immediately from the own desktop. On the other hand, this huge amount of heterogeneously structured information makes conventional methods for dealing with it such as book-marking and browsing of publisher Web Sites inadequate.

University libraries have started to provide access to electronic journals. In most cases, however, these are essentially collections of links to the journal home pages at the publishers' Web Sites. In some cases, these links are also included in the library catalogs. Examples for electronic journal link collections are (among many others) the Stanford University Libraries [27], Simon Fraser University

* Supported by the German Research Society, Berlin-Brandenburg Graduate School on Distributed Information Systems (DFG grant no. GRK 316)

Electronic Library in Computing Science [25], and the Electronic Journals collection [29] at the Stony Brook University Libraries.

In the case of the *Elektronische Zeitschriftenbibliothek* [28] at the University of Regensburg, a search-able catalogue of electronic journals is provided. It allows to focus the search to certain scientific disciplines or to journals available in print at a specific Bavarian university library. For each journal, links to the journal home page, to the catalogue entry, as well as an indicator about the electronic availability of the full text are presented.

The DBLP computer science bibliography maintained by Michael Ley [19] offers tables of contents of many conference proceedings as well as of about 150 computer science journals. For some journals, links to the electronic editions of the articles are provided. Each author name occurring in these tables of contents is linked with the bibliography of this author. However, most publishers do not provide metadata. So, the majority of the bibliographic data for journals has to be entered *manually by librarians* into a database. The HTML pages of this service are generated in advance as a materialized database view.

Another service that relies mostly on manually entered bibliographic information is the British "national information service for the higher education community" BUBL [6]. BUBL provides search-able abstracts and tables of content for about 200 journals. Unfortunately, links to the corresponding electronic journal editions are missing, and only links to publisher home pages are provided.

There are other projects which import the full text of articles into the digital library, thus supporting search facilities on these articles. JSTOR [16] is an archival project for older volumes of journals which are scanned, indexed using text recognition software, and stored in image form. A search-able WWW interface to these journals is available to the participating universities.

In the DeLIver [24] project, publishers provide articles of selected electronic journals in SGML form. These articles are stored in the digital library and indexed using so-called *concept spaces*, thus supporting advanced search facilities. In contrast to this close cooperation, only minimal cooperation of the publishers is assumed in our DigLib project.

From all these projects we can see that catalogues of electronic journals which go beyond collections of hyper-links require the acquisition of metadata on the included journals. In the cases we have seen, this is still done manually by librarians. If publishers do not provide this information, active methods for gathering metadata from the available WWW sources are needed.

In principle, all information needed for a catalogue of electronic journals is present at the publisher's Web Site. However, the form of this information varies extremely: it depends on the overall design of a site, on the policy of updating or dynamically generating pages and of displaying new data. To correctly design a mechanism importing metadata from a publisher's site, those particularities must be taken into account. On the other hand, a generic method is necessary to describe the site and the import activities it calls for at an abstract level. Otherwise, any change in a site would break the existing import scripts and require costly re-implementation.

In this study, we present a mechanism for the import of bibliographical data into the local repository of a digital library. We use an adaptive import scheduling strategy which estimates the publication dates of new journal issues. At each estimated due day, the "HyperNavigator" is triggered to import bibliographical data on the new issue by navigating the publisher's Web Site. The HyperNavigator is a module based on our HyperView methodology for wrapping and integrating heterogeneous Web Sites. It is based on high-level declarative view specifications and a demand-driven materialization strategy. Its formal framework is the subject of a companion paper [11]. Here we demonstrate how our methodology can be applied to the problem of wrapping publisher Web Sites.

Our import mechanism for bibliographical data is used in the Natural Sciences Digital Library project (Free University Berlin), which we describe in Section 2. The import scheduling strategy is discussed in Section 3. In Section 4, we present the HyperNavigator, first as a generic architecture and then in the context of our project. Related work is presented in Section 5. We conclude our study in the last section.

2 Overview of the DigLib Project

The DigLib project at the Free University Berlin establishes a digital library for the natural sciences. This will enable the university library to provide Web-based access to electronic journals and university publications like dissertation theses and technical reports. Within the framework of DigLib, a search-able catalogue of electronic journals is being developed. The contents of the journals are accessible at the site of their publishers, reachable by links kept in the catalogue. The current DigLib prototype provides only links to journal home pages.

2.1 Why using the HyperView methodology in DigLib?

By applying the HyperView methodology to wrap publisher Web Sites, we can provide not only links to the current issues, but also links to articles together with bibliographic data such as the titles and authors. These data can become part of the catalogue. Furthermore, we can use HyperView to wrap existing search facilities of the publisher Web Sites in order to support distributed full-text search on abstracts or even articles. Hence, the user may issue content-related queries on the catalogue and obtain links to relevant journal issues and articles.

The information retained on each electronic journal must be automatically extended, as new issues appear. The publication data on these new issues should be incorporated into the catalogue and become available to the users for querying. This raises the issue of discovering when a new issue of a journal becomes available at the publisher's Web Site, and the challenging issue of importing metadata on it to the local repository containing our catalogue.

These Web Sites are intrinsically heterogeneous in their structure and in the way they are updated. For example, one publisher may decide to keep each volume of a journal on a separate page, while another places all issues of the journal on a single page. Some publishers provide links to the current issue of a journal, others require the user to pick the newest issue from a list of all issues. If a link to the current issue exists, its target URL may be constant or variable etc. etc. Since the target size of the DigLib corpus will exceed the one thousand journals from many different publishers, we have to provide a solution for automatic metadata import in spite of that heterogeneity. The solution we propose in Section 4 is based on the HyperView methodology.

2.2 The Architecture of DigLib

The abstract architecture of DigLib is shown in Fig. 1. It can be seen as a configuration of the generic "Warehouse for Internet Data" (WIND) architecture presented in [10]. The WIND architectural model foresees a set of local repositories, in which database objects and textual documents from heterogeneous sources can be stored. In the framework of DigLib, only one relational database system (RDBMS) is needed.

This RDBMS stores the search-able catalogue of bibliographic records together with other, administrative data like user profiles. The end-user interface for query formulation is a HTTP server. The PHP server-side HTML embedded scripting language [23] is used for database access. An extended interface is foreseen for the librarian, for operations like insertion of new journals and registration of users.

The Repository Manager coordinates the maintenance of the catalogue data and of the user profiles in the RDBMS. One of its services is the Import Scheduling Service ("ISS"), which periodically schedules the import of the latest journal issues. The import itself is performed by the HyperNavigator. The HyperNavigator has two levels: the publisher-specific HyperView wrappers and the publisher-independent mediator. The mediator guarantees access transparency, in that all access requests performed by the Repository Manager refer to journals without knowledge of their actual HTML representation. The activation of the wrapper responsible for each publisher is performed internally by the mediator.

2.3 The Modules Responsible for Information Import

As new journal issues become available at the publishers' sites, we need to import their description in our local repository. We have designed an "Import Scheduling Service" (ISS). The ISS selects daily the journals of which the latest issue should be imported, and puts them into a ToDo list. This list is processed by the HyperNavigator, which inserts the descriptions of the new issues into a private relation. At some off-peak hour, this relation is used to update the DigLib catalogue. This approach ensures that the activities of preparing and performing the import do not affect the availability of database tables used by other services.

An overview of the import mechanism is depicted in Fig. 2. The ISS and the HyperNavigator are described in the next sections.

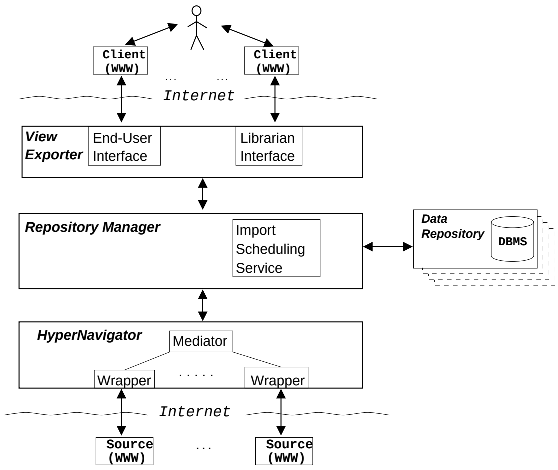

Fig. 1. The DigLib architecture

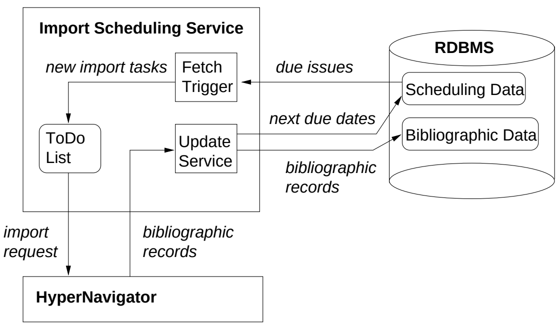

Fig. 2. The import activities in DigLib

3 The Import Scheduling Service

The Import Scheduling Service deals with a specialized instance of the *materialized view maintenance* problem. In the terminology of [30], a publisher Web Site wrapped by the HyperNavigator can be seen as a *query-able source* on which a view is established using an *eager* approach. In our case, changes in the source can be detected relatively efficiently, since journal issues appear with a known frequency and form a running sequence. Therefore we can estimate the best time for polling a source and have to watch only a limited number of HTML pages for discovering new journal issues. For a detailed overview on the general problem of materialized view maintenance see [14].

The Import Scheduling Service (cf. Fig. 2) consists of two (sub)services: (i) The “fetch trigger” identifies the journals, whose next issue should be fetched, and inserts them into a ToDo list. (ii) The “update service” obtains data on each newly imported issue from the HyperNavigator and uses them to compute the date at which the next issue should be fetched.

3.1 The Schedule Data of the ISS

The private relation ScheduleData of the ISS contains the data necessary to decide when the new issue of each journal must be imported. The signature of ScheduleData is shown on Table 1.

ScheduleData	
Description	*Attribute name*
Journal identifier	`jid`
Publication rate	`rate`
Next issue due date	`next`
Number of issues until new approximation	`max_imports`
Counter of issues until new approximation	`count_imports`
Minimum delay encountered thus far	`min_delay`
Days in advance for import start	`days_in_advance`

Table 1. The internal relation ScheduleData used by the ISS

The journal identifier is a foreign key to the journal entries in the DigLib catalogue. The `rate` must be specified on an annual basis. The `next` date specifies the day of the next import attempt. If the given `next` is already past upon insertion, the "fetch trigger" will place the journal right into the ToDo list.

The second group of attributes in Table 1 concerns the approximation of the `next` due date by the "update service" of the ISS. If no exact `next` date is provided by the publisher, `next` is incremented after each import by 1 year / `rate` days. This estimation can be subject to errors so that either the import is attempted too early and has to be repeated, or it is too late, and users have to wait unnecessarily long for the new issue.

The ISS therefore follows a conservative strategy in the sense that `next` is adjusted by the ISS `days_in_advance` days before the actual date of publication. The ISS keeps a `min_delay` statistic and re-adjusts its approximation after `max_imports`. The librarian may prevent this readjustment by setting the value of `max_imports` to NULL.

3.2 Importing Journals using the ToDo List

The "fetch trigger" of the ISS inserts into the ToDo list an entry for each new issue to be fetched by the HyperNavigator. Each entry is comprised by the journal identifier `jid`, a `priority` value, and access information for the new issue. This access information is obtained from the DigLib catalogue and consists of publisher, journal URL, number and volume of the current issue. The priority is computed from statistics kept in the catalogue. It takes into account the number of users registered with DigLib for notifications on this journal and and the number of requests for the journal.

If the new issue has not yet appeared or the available time span for the import run is over, the import fails and the journal remains in the ToDo list. In this case, its priority is increased. Otherwise, the journal is removed from the list and the publication data are inserted into the private relation of the HyperNavigator. The `jid` is also returned to the "update service" of the ISS which uses this information to compute the `next` due date for this journal.

3.3 Computing the Due Date of the Next Issue

For each new issue imported, the "update service" of the ISS resets the `next` and `current_issue` values of the journal. If the value of `max_imports` is set, we first compute the minimum delay encountered thus far and increase the counter of the issues imported till now. We then increase the value of `next` by the publication period of the journal, as computed from the journal's annual `rate`.

Thereafter, we test whether `max_imports` have already been performed. If yes, then it is time to readjust the `next` value against the `min_delay` encountered thus far. This implies shifting the `next` by as many days as the `min_delay`, so that premature attempts to import the issue are avoided in the future. If it is likely that the journal appears earlier than assumed by `next`, the administrator may specify that the `next` is shifted backwards by a number of `days_in_advance`. Once a satisfactory approximation of `next` is reached, the `days_in_advance` should be reset to zero. Hence, the new value of `next` is set by shifting the old one forward by `min_delay` days and backward by `days_in_advance`.

3.4 Obtaining Notifications from the Publishers

The ISS is designed to cope with the delivery of new issues, for which no exact publication date is available. If the publisher can provide this information, the ISS must replace the `next` date of the affected journals. For the delivery of this information, two different scenarios are possible: (i) If the publisher actively delivers this date, e.g. via a eMail service, then the ISS must be extended by a module receiving and processing such notifications. Alerting services based on other technologies [9], e.g. channels, may be used as well. (ii) If the publisher just announces the date passively on its Web pages, then the HyperNavigator can be used to poll the publisher site for this information.

4 The HyperNavigator

4.1 The HyperView Approach

The HyperView methodology is intended for the integration of well-known Web sources and the acquisition of information from them. We used it in the framework of DigLib to build the HyperNavigator, our module for the import of journal metadata from the Web Sites of the publishers.

The HyperView approach is based on the architecture shown in Fig. 3. Each Web Site is encapsulated in a site-specific wrapper. Information from different wrappers is integrated in a mediator. From the conceptual viewpoint, we use different levels of abstraction to model Web Sites. These levels are reflected by the layers of our architecture. The HTML layer models single HTML pages, the ACR layer the abstract structure of each Web Site, and the database layer the integrated domain-oriented view of all Sites.

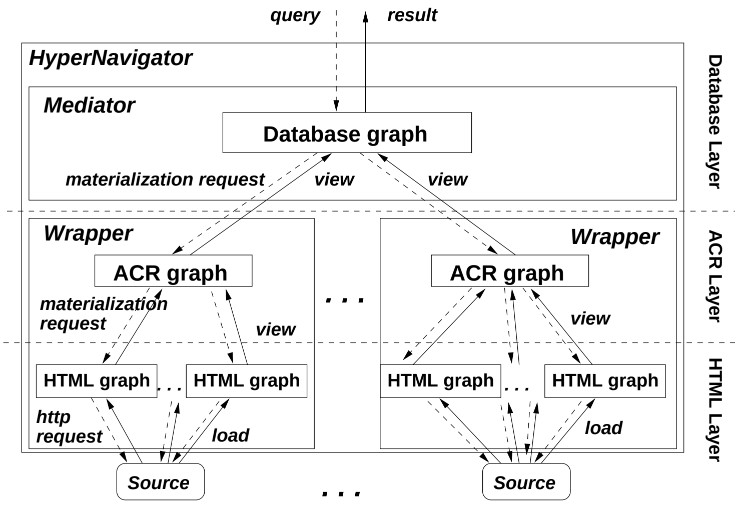

Fig. 3. The layers of the HyperNavigator

Each layer consists of labeled directed graphs which are conceived as abstractions of the graphs in the preceding layer. Each graph conforms to a given graph schema. Inter-layer edges connect abstract data items with the data items from which they abstract. Thus, a large layered data graph is formed.

The layered design is intended to provide transparency and encapsulate site-specific constructs. The graph of each of the higher layers is computed as a view of the layer immediately below it. These views are defined by declarative graph-transformation rules which are invoked on demand to materialize new graph elements at the higher layer, by matching graph elements and thus extracting information from the lower layer. On the HTML layer, HTML pages are loaded, parsed, and represented as directed labeled graphs on demand.

In the current prototype for the DigLib project, the information in the HyperNavigator database layer is computed in response to queries from the import scheduling service and inserted into the relational database system of DigLib.

4.2 Modeling Web Sites

When setting up an HyperView system, two kinds of schemata must be developed: a global, domain-oriented schema for the database layer, and for each Web Site a site-specific *Abstract Content Representation* (ACR) schema. The ACR schema reflects the logical schema of the underlying Web Site. HTML pages are modeled by a common generic schema.

A set of site specific rules has to be defined which is used to match the data contained in these HTML graphs and to create an intermediate data layer conforming to the ACR schema of the source.

Another set of rules has to be defined which restructures the information stored in the ACR graphs and to integrate it on a third layer into a common graph database conforming to the database schema. In DigLib, the database schema defines concepts for electronic journals, volumes and issues of journals, and articles, along with their usual attributes.

The advantage of using rules for information extraction compared with other more character-oriented techniques proposed in the literature (see Section 5) lies in the fact that its rules are declarative, easy to maintain, and (to some extend) robust against minor deviations and restructurings of pages. If structural changes in the Web Sites break the existing views, it is often sufficient to change only the rules for the ACR graph, without affecting the higher layers.

The HyperNavigator Database schema. In order to wrap Web Sites for electronic journals, we first have to set up a database schema, as shown in Fig. 4, which models the domain of electronic journals independently of the heterogeneous structure of the publisher Web Sites.

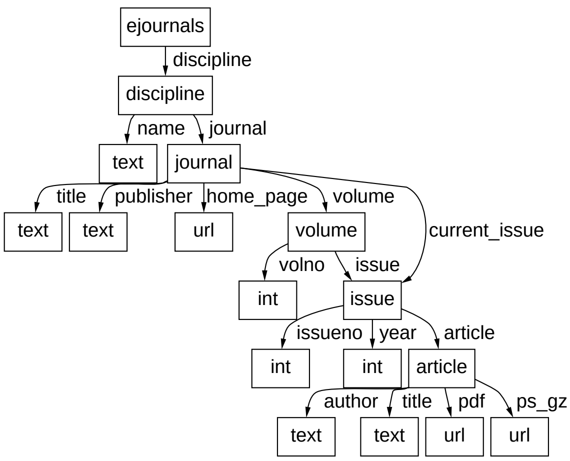

Fig. 4. ejournal database schema

We categorize all electronic journals by discipline. Each journal consists of several volumes, each having a number of issues. Each issue contains several articles, for which the authors, title, and links to the PostScript/PDF version are retained. It should be stressed that this is a preliminary schema that will further evolve during the lifetime of DigLib. A more complex schema graph, probably containing cycles, can be modelled as easily.

ACR Schema of an Example WWW Source. For each Web Site, an ACR schema has to be developed which reflects the conceptual structure of this Site. In this paper we present as example the Link information service <http://link.springer.de> of the Springer-Verlag Berlin Heidelberg New York. It provides several houndred electronic journals grouped by scientific discipline into so-called "online libraries". Each journal has a home page and a "Contents" page listing each volume together with the issues belonging to this volume. For each issue there is a table of contents, where the authors and title of each article can be found, across with links to the abstract and to the full paper in PostScript and PDF format.

The conceptual structure of the Springer-Verlag server is depicted in the ACR schema of Fig. 5. Note that this schema only models information related to electronic journals. Other data made available by the server need not to be modeled.

Each node of the ACR schema models a class of pages or page fragments. The edges correspond to hyper-links between pages or to part-of relations between page fragments. Due to structural differences between journals published by the German and the American branch of Springer-Verlag, we had to establish a separate subschema for the journals appearing in the American branch, whose nodes are indicated by the suffix "_us".

This example ACR schema has a simple tree structure. More complex structures, including cyclic ones, are also supported. An ACR schema with cycles would for instance be needed to model articles whose citations' list is electronically available and contains links to other articles. This is the case for articles published in the VLDB conferences and for scientific reports in HTML format.

4.3 Extracting Information from Web Sites

Parsing of HTML Pages into HTML graphs. HTML pages are analyzed in the HyperView system by a generic parser and turned into a tree-structured graph. As an example, in Fig. 6, the graph representing the "Contents" page of the International Journal on Digital Libraries is shown.

Each subdocument enclosed between an opening tag and its closing counterpart is represented as a subtree. The root of this subtree is labeled with the name of the tag. Tag attributes are converted into edges emanating from the root, which are labeled with the attribute name and point to a node labeled with the attribute value. The content of the subdocument is represented as a sequence of children nodes containing either untagged text (called PCDATA in

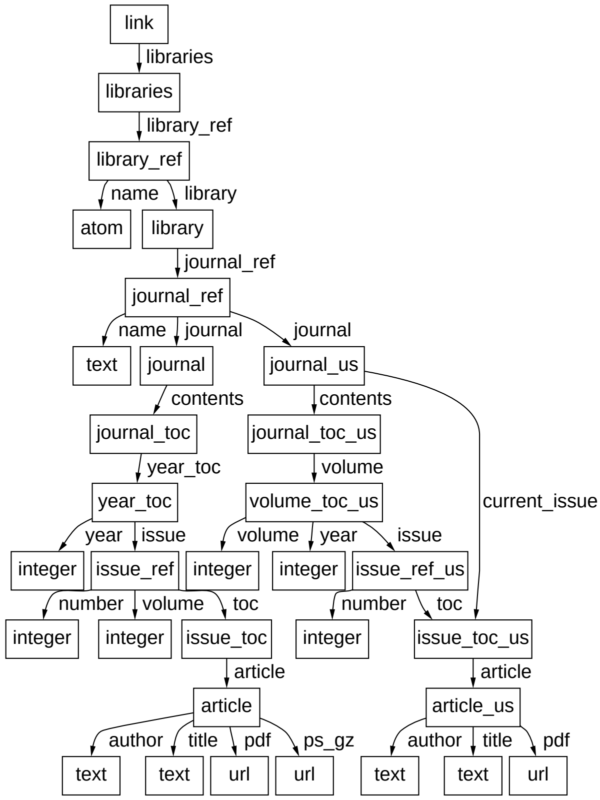

Fig. 5. ACR schema for Springer-Verlag. American journals indicated by suffix "_us".

HTML terminology) or root nodes of further tagged subdocuments. From the root node, numbered *#(i)* edges are pointing to these children nodes.

Defining a View over the HTML Graphs. We now describe the establishment of the ACR graph over the HTML pages of a Web Site. Each ACR graph is a view defined by a set of rules developed and maintained by the DigLib administrator. In the case of the Springer-Verlag, the complete ruleset consists of 15 rules; we explain two of them in detail.

As mentioned before, the "Contents" page of a journal lists all available issues, ordered by year. Suppose that the HyperNavigator has already loaded this page and there is a journal_toc node with a *source* edge pointing to the root node of this page in the ACR graph. The internal HTML representation of this page (with some icons and navigational links omitted) has been shown in Fig. 6.

Rule get_year_toc shown in Fig. 7 computes a *year_toc* edge leading from the journal_toc node to a new year_toc node in the ACR graph which corresponds to a page fragment listing all journal issues for a particular year. In the

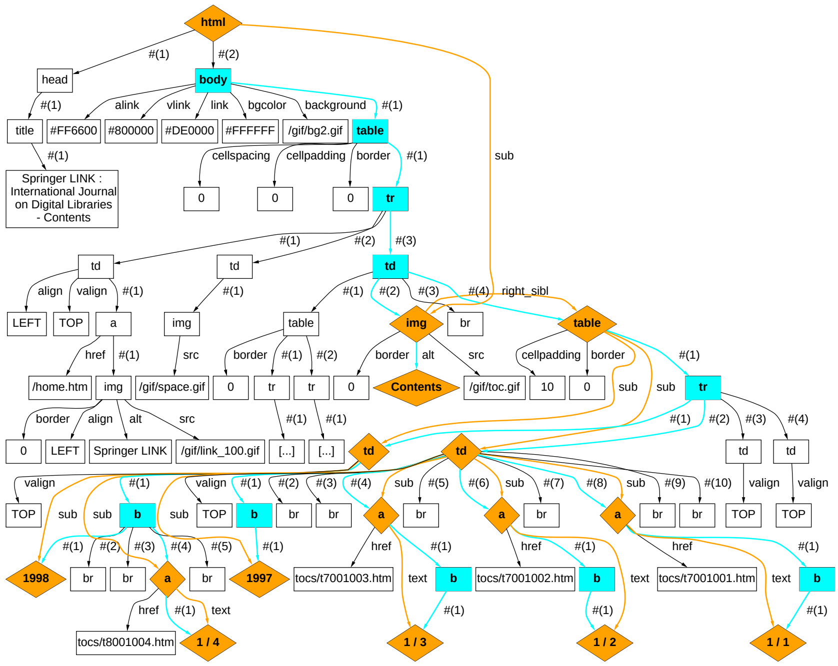

Fig. 6. HTML graph of the IJODL "Contents" page. Elements relevant to rule application are highlighted. Nodes in diamond-shape are direct matches for the nodes occurring in the rules discussed below. Relevant virtual *sub* and *right_sibl* edges are included.

existing HTML graph of the content page, the rule matches a path from the root <html> node to a <td> tag which contains a string matching a year number Year. This path follows derived *sub* edges expressing a logical subnode relation. This relation is implemented by alternative rules, depending on the type of the source node. Another derived edge, *right_sibl*, allows direct access to the right sibling of a node. A link to this <td> node and an year attribute with value Year is added to the new year_toc node then.

The second rule, get_issue_ref, computes an *issue* edge from the year_toc node (created by rule get_year_toc) to a issue_ref node with volume number Volume and issue number Number as matched by the HTML part of the rule. A *source* edge points to the anchor containing a hyper-link to the table-of-contents page of that issue. Further similar rules can be used now to follow this link, match entries for articles and their components, and extract information on author, title, and links to the abstract and the full document in PostScript or PDF format.

Defining a View over the ACR Graphs. We describe now how we establish the database graph over the ACR graphs of different publisher Web Sites. The

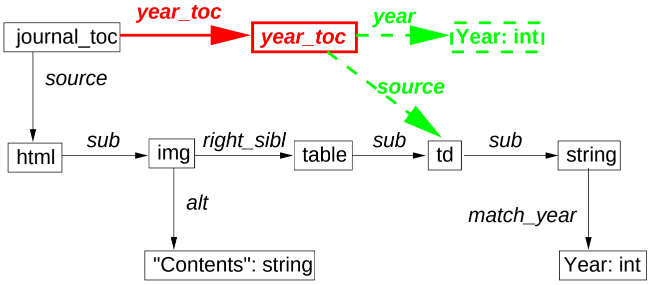

Fig. 7. Rule get_year_toc. **Notation** (also used for the other rules): (i) the upper part relates to the result graph of the view (here: the ACR graph), the lower part to the source graph (here: an HTML graph). (ii) Existing graph items to be matched are shown with normal-width (black) lines. (iii) Graph items which are materialized by the rule on demand are shown with bold (red) lines and boldface (red) font. (iv) Additional graph items produced as a side-effect of the rule are shown with bold dashed (green) lines and boldface (green) font.

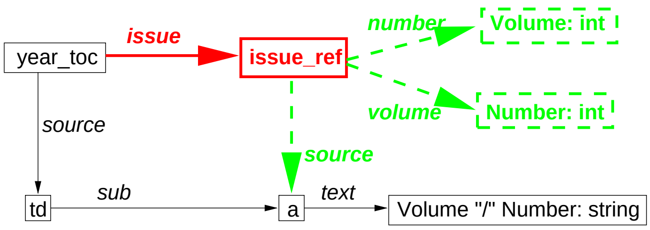

Fig. 8. Rule get_issue_ref

database graph is a view defined by a set of publisher-specific rules. From the ruleset for the ACR graph of the Springer-Verlag, we select the rule get_issue as an example. This rule is shown in Fig. 9.

In the ACR subschema for the German branch of Springer-Verlag, issue_ref nodes are organized by year. In the database graph, we have issue nodes organized by volume. The purpose of the rule get_issue is to map the issue_ref nodes to issue nodes.

Note that the issues comprising a volume of a journal are not necessarily identical with the issues having appeared in a certain year. Some journals may have two or more volumes per year. Volumes may start at any time of the year. With our graphical notation, this fact is specified in an intuitive way.

For a given journal node, the corresponding "Contents" page (represented by the journal_toc node) is found. For each issue_ref node that belongs to volume *VolumeNo*, has number *IssueNo*, and can be reached through a year_toc node for year *Year*, the rule creates a volume node and an issue node in the database and supplies them with the attribute values of *VolumeNo*, *IssueNo*, and *Year*.

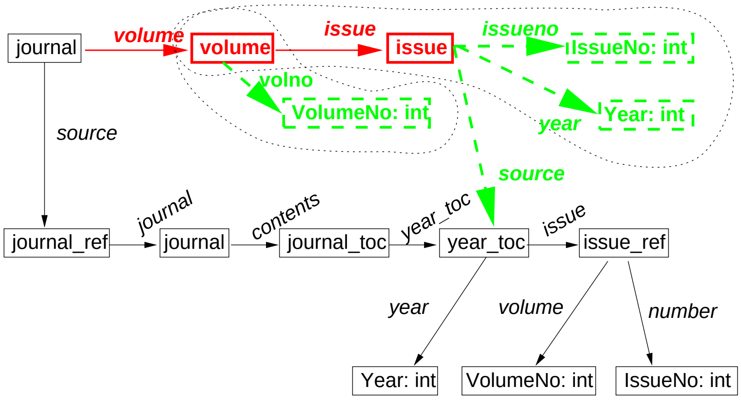

Fig. 9. ACR rule get_issue. Notation: *Reuse Subgraphs* are indicated by dotted loops.

In order to prevent the creation of multiple volume nodes for the same *VolumeNo*, we use the notion of *Reuse Specifications* [11]: Elements of a rule can be grouped together to form a *Reuse Subgraph* (indicated by a dotted loop). On applying the rule, a new instance of a reuse subgraph will be created only if the database does *not* contain an instance of this subgraph yet. In the case of rule get_issue, this implies that a volume node is identified by its *volno* attribute, and a issue node is identified by the volume node to which it is belonging, together with its *issueno* and *year* attributes.

Querying the HyperNavigator. The HyperNavigator provides a simple graph-based query language which is used to specify the information to be imported into the DigLib catalogue. A query is a rule that does not produce new graph elements, but just returns for each match a binding for the variables occuring in the rule. HyperView rules can be supplied with arbitrary computable constraints on the variables in order to restrict the set of matches. For each admissible match, we return the variable, or – in the case of the digital library – insert it as a new tuple into the database of the DigLib server. In the case of a stand-alone HyperView server which has a user interface of its own, rules will be used to generate HTML pages which contain the results of a query in a generic or customized format.

In Fig. 10 we show an example of a query retrieving volume, issue number, and title of all articles containing the word "metadata" in their title and having appeared after 1996 in any computer science journal on "Digital Libraries" (string in the journal name).

In DigLib, we use information from the digital library database to formulate import queries that avoid unnecessary page accesses. In particular, the publisher, the URL of a journal and the volume and number of the last imported issue are given as parameters. Hence, the HyperNavigator does not have to navigate from

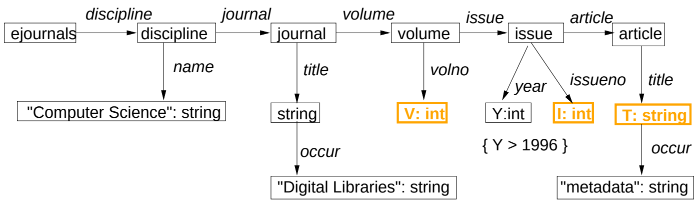

Fig. 10. Example Query: all articles on "metadata" having appeared in a computer science journal on "Digital Libraries" after 1996. The vertices corresponding to the retrieved attributes are shown in boldface.

the publisher's home page to the journal page but can access it directly in order to check for a new issue.

Fig. 11 shows the import query for the next issue of the Intl. Journal on Digital Libraries. From the scheduling data in the catalogue we know that the current issue is the 4th one of the 1st volume. The Import Scheduling Service reads this information and passes it to the HyperNavigator when establihing the ToDo list (see subsection 3.2). The result of this query is inserted by the HyperView into the DigLib catalogue.

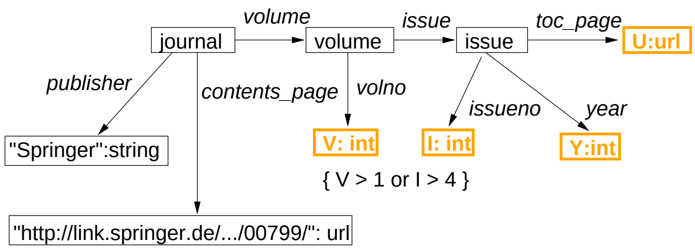

Fig. 11. Import Query for the next issue of the IJODL

4.4 Implementation of the HyperNavigator

A prototype of the HyperNavigator has been implemented in SWI Prolog. Schema and instance graphs are stored in the Prolog fact database. We implement HyperView rules as Prolog predicates by using a special syntax for expressing rules. This special syntax is translated into plain Prolog code by a preprocessor.

For rule get_year_toc (depicted in Fig. 7), this syntax and the result of the translation are shown in Fig. 12.

The prolog predicate for a rule can be registered with a schema graph by specifying that it is to be used to materialize edges with a certain label in an instance graph of this schema.

```
get_year_toc(ACR, TOC, Year):--
    TOC @ ACR: journal_toc ->
    source = _ @ HTML ->
    sub = Img: img -> [
        alt =  'Contents',
        right_sibl =  _ : table ->
        sub = YearNd: td ->
        sub-> match_year = Year ] -->
    TOC @ ACR -> year_toc = YearTOC ->
    [   source =  YearNd @ HTML,
        year =  Year ]).
```

```
get_year_toc(ACR, TOC, YearTOC):-
    edge(ACR, TOC, source, Root @ HTML),
    edge(HTML, HTML, sub, IMG),
    vertex(HTML, IMG, img),
    edge(HTML, IMG, alt, 'Contents'),
    edge(HTML, IMG, right_sibl, Table),
    vertex(HTML, Table, table),
    edge(HTML, Table, sub, YearNd),
    vertex(HTML, YearNd, td),
    edge(HTML, YearNd, sub, String),
    edge(HTML, String, match_year, Year),
    add_vertex(ACR,YearTOC,year_toc),
    add_edge(ACR,TOC,year_toc,YearTOC),
    add_edge(ACR,YearTOC,source,YearNd@HTML),
    add_edge(ACR,YearTOC,year,Year).
```

Fig. 12. Prolog notation and translation for rule get_year_toc

Once there is a request for edges with that label emanating from a certain source node, the Prolog fact database will be queried first. If this fails, each predicate registered for that kind of edge is called in turn to materialize the requested edges. This approach overcomes the inefficiencies inherent to most graph-rewriting techniques by always applying rules only on demand, and focusing the rule application to a given vertex in the graph.

Without being particularly optimized, the prolog prototype performs fast enough: the runtime is dominated by the page load time which depends on the network throughput and the performance of the publisher's HTTP server.

5 Related Work: Wrapping semistructured data sources

In digital libraries, wrappers are used mostly to adapt an existing search facility of a source to a certain search protocol, say, Z39.50 [20]. However, little has been reported about the use and construction of WWW wrappers in the field of digital libraries.

In the Stanford Digital Library Project, wrappers (here called "proxies") are used to access distributed, autonomous services, but also for providing metadata on attribute sets of search proxies [5]. The actual construction of wrappers is not covered in this work. For building WWW wrappers, the WebExtractor [15] (discussed below) has been developed in the context of this project.

In the German MeDoc [8] project, publisher servers are wrapped in order to support a common search protocol and to provide term frequency profiles which are used to preselect sources for distributed queries. Due to the close cooperation with publishers in this project, these wrappers did not have to be built on top of WWW sources.

Wrappers for WWW sources have to provide three functionalities: navigation, information extraction, and information integration. In the HyperView system, these aspects are combined into a seamless graph-based approach.

Web query languages, such as WebSQL [21], W3QS [17], WebLog [18], and WebOQL [2] focus primarily on ad-hoc queries. With the exception of WebOQL, they all ignore the syntactical structure of HTML pages and consider only the included hyper-links. Still, these approaches provide valuable insight into the navigational aspect of Web Site wrapping.

InfoExtractor [26], a tool developed by INRIA/Bull, and the Web Extractor [15] built by the TSIMMIS [13] group at Stanford both use a hierarchy of regions defined by regular expressions to analyze the structure of a document and extract parts of it as result. These methods are completely character-oriented and do not take advantage of the structure provided by the HTML syntax. This makes it difficult to write wrappers that are robust with respect to reordering of page elements or small variations in the HTML text.

The wrapper generation toolkit [3] of Ashish and Knoblock works in a semi-automatic way by presenting guesses on the structure and interesting regions of example pages to the user. Again, this tool is character-oriented and sensitive to details of the HTML code. Furthermore, the issue of Web navigation is not covered.

NoDoSE [1] aims at extracting structured data from semistructured text documents in general, not only for HTML pages. The user has to decompose hierarchically an example document and specify the type of data to be extracted from each part, with respect to a user supplied object-oriented schema. The system comprises a mining component which supports this decomposition by inferring grammar rules.

The YAT [7] system is a tool for specifying data conversions. The YAT data model represents documents by ordered labeled trees which can be linked by references. Tree models form an instantiation hierarchy. YATL is a declarative, rule based language featuring pattern matching and restructuring operators. Each rule matches one or more (sub-)trees and produces a new tree. Currently, the major limitation of YAT is its missing support for incremental data conversion on demand. This means that all data has to be imported and converted once, making it less suitable for Web views. Furthermore, pattern matching across hyper-links which is important for Web Site navigation is not supported.

EDITOR is a procedural language [4] for extraction and restructuring of text from arbitrary documents. It combines a set of elementary text manipulation commands with loop constructs. This approach seems to be quite low-level imposing negative consequences for the legibility and maintenance of EDITOR programs. EDITOR is used in the context of the ARANEUS project which provides a tool box for the management of data from the Web. The EDITOR language is used for extracting data from semistructured HTML pages which are modeled in the hierarchical ("page-oriented") ARANEUS Data Model (ADM). The language ULIXES is used to specify and navigate within the link structure of the pages and to build nested-relational views over the extracted data. The generation of Web pages from these views is specified in the language PENELOPE. Compared to the seamless, graph-based HyperView approach, ARANEUS uses three different languages and different data models (ADM, relational) within the same system.

Strudel [12] is a system for the management of Web Sites. It assumes the underlying data to be available in a graph representation, but does not deal with the construction of wrappers which could retrieve this data from external Web Sites.

6 Conclusion and Outlook

The increasing number of electronic journals calls for effective access. Digital libraries can meet this demand by establishing search-able catalogues integrating the bibliographic data of electronic journals and providing uniform access to the articles stored at heterogeneous publisher Web Sites.

In this paper we deal with the problem of metadata acquisition for such a catalogue, in the context of the ongoing Natural Sciences Digital Library project at the Free University of Berlin. We propose an adaptive import strategy implemented in the Import Scheduling Service.

The HyperNavigator is triggered by the Import Scheduling Service in order to carry out the metadata import. It is based on the HyperView methodology for establishing views over heterogeneous Web Sites. The main advantages of our methodology are its graph-based approach and the uniform use of views defined by declarative rules at all levels of the system.

We have demonstrated the feasibility of this approach for the *Link* information service of the Springer-Verlag publishing house. We have presented the required schemata and have discussed detailed examples of the rules defining the views. We have also covered briefly the implementation of the HyperNavigator.

We are currently integrating the HyperNavigator with the DigLib server being under development. An interesting extension of this work is to use the HyperNavigator for wrapping also the existing search interfaces of publisher WWW servers, thus supporting distributed full-text retrieval.

Acknowledgments: We wish to express our special thanks to Ulf Leser, Agnes Voisard, Jana Lewerenz, and Daniel Faensen for their helpful suggestions and comments on this paper.

References

1. B. Adelberg. NoDoSE: A tool for semi-automatically extracting semi-structured data from text documents-brad adelberg. In *SIGMOD Conference 1998*, 1998.
2. G. Arocena and A. Mendelzon. WebOQL: Restructuring documents, databases and webs. In *Proc. of 14th. Intl. Conf. on Data Engineering (ICDE 98)*, 1998.
3. N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. In *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.

4. P. Atzeni and G. Mecca. Cut & paste. In *PODS'97*, pages 12–15, Tucson, Arizona, 1997.
5. M. Baldonado, C.K. Chang, L. Gravano, and A. Paepcke. The stanford digital library metadata architecture. *International Journal on Digital Libraries*, 1(2):108–121, 1997.
6. BUBL (British National Information Service for the higher education community). <http://bubl.ac.uk/admin/purpose.htm>.
7. S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *SIGMOD Conference 1998*, pages 177–188, 1998.
8. M. Dreger et al. Medoc information broker - harnessing the information in leterature and full text databases. In N. Fuhr J. Callan, editor, *Proc. SIGIR workshop on Networked Information Retrieval*, 1996.
9. D. Faensen, A. Hinze, and H. Schweppe. Alerting in a digital library environment – do channels meet the requirements. In *ECDL'98*, 1998.
10. L.C. Faulstich, M. Spiliopoulou, and V. Linnemann. WIND: A warehouse for internet data. In *Advances in Databases – Proceedings BNCOD 15*, number 1271 in LNCS, pages 169–183. Springer, 1997.
11. Lukas C. Faulstich. Integrating web sites using HyperView. Submitted for publication., 1998.
12. Mary Fernandez, Daniela Florescu, Jaewoo Kang, Alon Levy, and Dan Suciu. Catching the boat with Strudel: experiences with a web-site management system. In *SIGMOD*, pages 414–425, 1998.
13. H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Integrating and accessing heterogeneous information sources in TSIMMIS. In *AAAI Symposium on Information Gathering*, pages 61–64, 1995.
14. A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques and applications. *IEEE Quarterly Bulletin on Data Engineering; Special Issue on Materialized Views and Data Warehousing*, 18(2):3 – 18, 1995.
15. J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.
16. JSTOR. <http://www.jstor.org/>.
17. D. Konopnicki and O. Shmueli. W3QS : A system for WWW querying. In *ICDE'97*, pages 586–586, April 1997.
18. L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the Web. In IEEE, editor, *RIDE'96*, pages 12–21. IEEE Computer Society Press, 1996.
19. M. Ley. Die Trierer Informatik-Bibliographie DBLP. In *GI Jahrestagung 1997*, pages 257–266, 1997. <http://dblp.uni-trier.de>.
20. C.A. Lynch. The Z39-50 information retrieval protocol: An overview and status report. *ACM Computer Communication Review*, 21(1):58–70, 1991.
21. A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *International Journal on Digital Libraries*, 1(1):54–67, 1997.
22. P. Merialdo P. Atzeni, G. Mecca. To weave the web. In *VLDB '97*, pages 206–215, 1997.
23. PHP3 manual. <http://www.php.net/manual/>, 1998.
24. B.R. Schatz, W.H. Mischo, T.W. Cole, J.B. Hardin, A.P. Bishop, and H. Chen. Federating diverse collections of scientific literature. *IEEE Computer*, 29(5), 1996.
25. Simon Fraser University Electronic Library in Computing Science. <http://fas.sfu.ca/projects/ElectronicLibrary/Collections/CMPT/>.

26. D. Smith and M. Lopez. Information extraction for semi-structured documents. In *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.
27. Stanford University Libraries – Electronic Journals Collection. <http://www-sul.stanford.edu/collect/ejourns.html>.
28. Elektronische Zeitschriftenbibliothek, Universität Regensburg. <http://www.bibliothek.uni-regensburg.de/ezeit/ezb.phtml>.
29. Stony Brook University Libraries – electronic journals. <http://www.sunysb.edu/library/ldeljour.htm>.
30. J. Widom. Research problems in data warehousing. In *4th International Conference on Information and Knowledge Management*, pages 25 – 30, 1995.