Chapter 2

HyperView by Example: Wrapping Publisher Web Sites

In this chapter, the HyperView methodology is demonstrated by presenting a case study on electronic journals. The key concepts of HyperView are introduced informally and examples for their application to the integration of publisher Web sites are presented. This chapter is based on [Faulstich and Spiliopoulou, 1998, Faulstich and Spiliopoulou, 1999] which describe the import modules for bibliographic data in the DARWIN project that have been implemented using HyperView.

2.1 Digital Libraries of Electronic Journals

Publishers are currently offering several thousands of electronic journals, and this number is rapidly increasing. For researchers, the availability of online journal editions has made life much easier since all information is reachable immediately from the researcher's 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 (cf. [SUL, 1998], [SFU, 1996], [USB, 1998]).

The DBLP computer science bibliography maintained by Michael Ley [Ley, 1997] offers tables of contents of about 150 computer science journals. For some journals, links to the electronic editions of the articles are provided. However, most publishers do not provide bibliographical data. 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.

From these examples and further projects discussed in [Faulstich and Spiliopoulou, 1999] 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 studied, 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 a lot: 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 requires. Otherwise, any change in a site would break the existing import scripts and require costly re-implementation.

In this chapter, the HyperView based solution for integrated access over electronic journals of multiple publisher Web sites, as realized in the DARWIN project, is presented.

2.1.1 The DARWIN project

The DARWIN project at the Free University Berlin establishes a digital library for the natural sciences. This project will enable the university library to provide Web-based access to electronic journals and university publications like dissertations and technical reports. Within the framework of DARWIN, a searchable catalogue of electronic journals is being developed. The current issues of the journals are accessible at the site of their publishers, *reachable by links kept in the catalogue*. This feature distinguishes DARWIN from other projects which just provide links to publisher or journal home pages. A further distinction is in the fact that the integration of information from Publisher Web sites is achieved without requiring cooperation or feedback from the publishers. For the envisaged corpus size of DARWIN, (5000+ journals from 500+ publishers) a tight cooperation is impossible for pragmatic reasons.

Rather than deploying an ad hoc solution to implement mediator and wrappers, it therefore makes sense to use the HyperView methodology which allows to model publisher Web sites and

supports query-able views on these Web sites defined by declarative rules.

2.1.2 Use cases

We envisage the following use cases for the DARWIN catalogue:

Browsing: the user explores the available journals via a list of scientific disciplines or publishers. The user inspects current or past issues of certain journals, some articles and their abstracts in these issues, and sometimes downloads the article's full text.

Looking up a bibliographic reference: the user has a complete bibliographic reference for an article in a journal. (S)he selects the journal and the issue from the digital library by submitting a search mask and may then choose to download the article's full text.

Searching with incomplete information: the user specifies the known bibliographical information in a search mask and receives from the digital library all articles matching the request. The user then inspects the returned results and decides to refine the search and / or to download some of the articles.

Personalization: the user selects a personal list of interesting journals which is maintained as part of the user profile. Moreover, the user may define a search profile that characterizes interesting articles.

Alerting: the user is notified of new issues of occurring in his / her personal journal list or articles matching the search profile. The notification can take the form of an HTML page presented to the user at login, a personalized CDF channel, or periodic e-mails.

These use cases suggest that bibliographic data on journal issues and the articles in these issues have to be collected and homogenized from the underlying publisher Web sites. The virtual Web site of the catalogue should support both browsing and search with possibly incomplete information. Journal issues should be presented in a uniform way that saves the user from coping with the heterogeneities of the publisher Web sites. In this case, links to full-texts at the publisher Web sites have to be provided. Alternatively, links to the external issue and article pages can be provided.

2.2 Modeling publisher Web Sites

2.2.1 Generic approach

When setting up a HyperView system, the graphs occurring at each layer of the HyperView architecture have to be described by schemata.

First, a global, domain-oriented schema for the database layer has to be developed. This schema captures the information relevant to the intended application.

Then for each Web site, a site-specific "*Abstract Content Representation*" (ACR) schema has to be developed. The ACR schema reflects the logical schema of the underlying Web site.

For HTML pages, there exists a fixed, generic schema that can be used for all Web sites. This schema describes HTML pages as syntax trees which are connected via edges representing hyperlinks.

If a virtual Web site is to be built using the HyperView System, then the structure of this Web site is to be specified by a "*User Interface*" (UI) schema similar to the ACR schema. Alternatively, it is possible to use the generic UI schema of the HyperView browser discussed in Section 6.5 and use the customization features of this database browser. In the framework of DARWIN, the HyperView System is used without the UI layer.

2.2.2 Graph Schemata

A *graph schema* is a graph that describes a class of *data graphs* conforming to this schema. A node of a schema graph represents a class of entities. Nodes of a data graph conforming to this schema node represent instances of this entity class.

A schema edge represents a navigable binary relationship of arbitrary cardinality between instances of the entity classes connected by the schema edge. Edges of a data graph conforming to a schema edge connect nodes conforming to the source and target of this schema edge.

The labels of graph schemata are type names representing sets of admissible instance labels. The sets of admissible labels may either be the value sets of atomic types (such as Integer, String), application specific types (such as journal ISSNs), or singleton sets denoting attribute or class names.

A data graph conforms to a schema if there is a structure-preserving mapping to a schema graph such that the label of each data graph element belongs to the type of its corresponding schema element. A mapping is structure-preserving if all paths in the data graph are mapped to paths in the schema graph.

This schema concept has the advantage that it supports attributes of unpredictable cardinality which are typical for semistructured data. Moreover it supports cyclic schemata and instances in a natural way, without requiring a special reference mechanism needed in other tree-based approaches [Cluet *et al.*, 1998, Buneman *et al.*, 1997].

Since in the CGDM data model of HyperView all graphs form clusters of a global, modularized graph, graph schemata similarly form clusters of the global, modularized graph schema of HyperView. In the following, the term “schema” means a cluster of this global schema.

2.2.3 The HyperView Database Schema

Before we can start to wrap Web sites, we first have to set up a global database schema. This schema does not need to describe all information available in the underlying Web sites. Rather, it depicts the data relevant for the application that is to be built on top of this database.

In DARWIN, the database schema describes electronic journals. We use the schema shown in Figure 2.1. This schema reflects the application requirements of DARWIN and is independent of the structure of each publisher Web site.

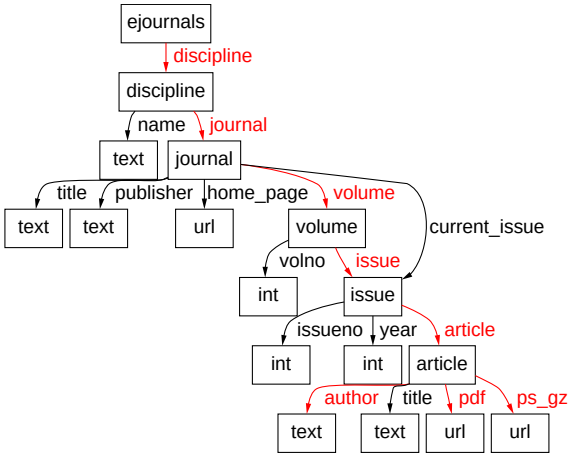

Figure 2.1: ejournal database schema

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.

2.2.4 ACR Schemata of Example Web Sources

In the classical problem of integrating multiple databases, the local schemata would be readily available. However, when integrating Web sites, that do not participate in the integration activities, the local schemata are not available. Hence, for each Web site, an ACR schema which reflects the conceptual structure of this site has to be derived from the HTML sources of the available Web pages in a kind of re-engineering step.

The development of an ACR schema is guided by the database schema which specifies which information offered by a web site is relevant to the planned information integration. Hence the ACR schema models only those parts of a Web site which are either relevant or which are necessary navigation paths leading to the relevant parts. The rest of a web site can be ignored by the schema designer.

In the ACR schema, a node models a group of pages or page fragments. Nodes are connected by labeled, directed edges which model hyper-links between pages or part-of relationships between pages and page fragments. Traversing of hyper-links and loading of the referenced pages is done transparently at the HTML level.

An ACR schema can be a simple tree or a cyclic graph. The latter case occurs for instance, when a chain of similar pages connected by hyper-links has to be modeled.

In general, we need an ACR schema per Web site that is to be integrated. We now present in detail three of the publisher Web sites which are wrapped in DARWIN.

Springer-Verlag Berlin, Heidelberg, New York offers the Link information service that can be found at <http://link.springer.de>. It provides several hundred electronic journals grouped by scientific discipline into so-called “online libraries”.

Each journal has a home page and a “Contents” page listing the issues of each year by volume and issue number. For each issue there is a table of contents, where the authors and title of each article can be found, together with links to the abstract and to the full text of the paper in a PostScript and PDF version.

For journals produced by the American branch of Springer-Verlag, a different structure is used which provides a link to the current issue and organizes the issues of a journal by volume (and not by year). Due to this differences between journals published by the German and the American branch of Springer-Verlag, a separate subschema modeling the journals appearing in the American branch had to be developed.

The conceptual structure of the Springer-Verlag server is depicted in the ACR schema of Figure 2.2. Note, that this schema only models information related to electronic journals. Other data made available by the server need not to be modeled for the application in DARWIN.

The ACM Digital Library at <http://www.acm.org/dl/> hosts the journals published by ACM. The schema of its web site is shown in Figure 2.3. Since ACM is dedicated to computer science, this Web site is not organized by scientific discipline.

Wiley-VCH again offers journals in several scientific disciplines on its Web site at <http://www.wiley-vch.de>. The schema of this Web site is depicted in Figure 2.4. Similar to Springer Verlag, this Web site shows heterogeneities in the structure of the offered journals. This requires a separate rule set for each of the alternative subschemata.

Discussion. Even though we find common domain-specific concepts such as journal, issue, article etc. in all ACR schemata, we can see that they are organized differently. For instance, issues can be organized by volumes or by year and the information can be distributed on one or more pages.

These example ACR schemata show a simple tree structure. More complex structures, including cyclic ones, are supported by the HyperView methodology as well. An ACR schema with

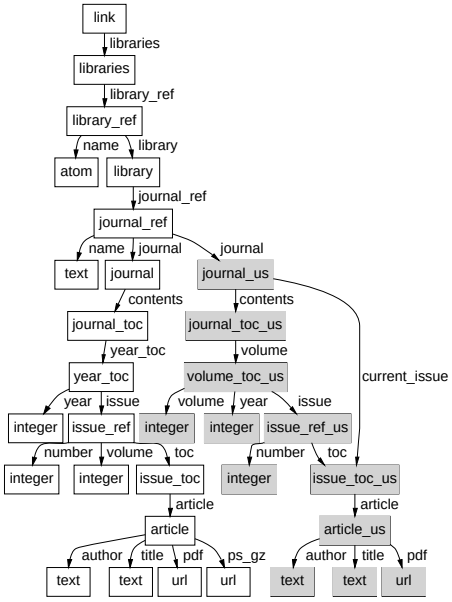

Figure 2.2: ACR schema for Springer-Verlag. Subschema for American journals indicated by filled nodes and the suffix "_us".

cycles would be necessary for instance to model a Web site providing information on electronically available articles referencing other articles. This is the case for articles published in the last VLDB conferences and in general for articles and reports published in HTML format.

2.2.5 Representing HTML Pages as 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 Figure 2.5, the graph representing the "Contents" page of the International Journal on Digital Libraries is shown.

Each subdocument enclosed between opening and closing tag is represented as a subtree. The root of this sub-tree is labeled with the name of the tag. Tag attributes are converted into edges starting 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 child nodes containing either untagged text (called PCDATA in HTML terminology) or parent nodes of further tagged subdocuments. From the parent node, numbered *#(i)* edges are pointing to its child nodes in textual order.

On HTML graphs, additional virtual edges modeling, for instance, the transitive part-of relationship sub or the connection between headlines and the subdocuments logically belonging to them are available. These virtual edges are implemented by a set of rules acting on the HTML graphs. Table 5.3 on page 71 lists and describes these edges.

2.3 Building Views on publisher Web Sites

Before we are able to pose queries formulated against the database schema defined above, the contents of the database graph have to be defined. This database graph is a "virtual" graph in the sense that it is a view on the ACR graphs which in turn are views on the HTML graphs of pages loaded from the underlying Web sites.

In our approach, virtual graphs are defined by HyperViews consisting of graph-transformation rules. They are materialized in a demand-driven way as response to queries posed against them.

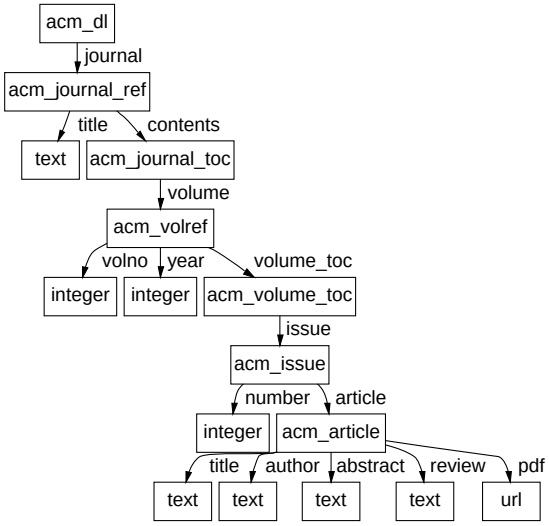

Figure 2.3: ACR schema for the ACM digital library.

As a consequence only the pages needed to answer a query are loaded from the underlying Web sites.

To create a view over the HTML pages of a particular Web site, a set of site specific rules has to be defined which is used to match the data contained in these HTML graphs and to materialize the intermediate ACR data layer conforming to the ACR schema of the source.

To set up a view over this ACR layer, another set of rules has to be defined. It is used to restructure the information stored in the ACR graphs and integrate it on a third layer into the database graph which conforms to the database schema. When the database graph is queried, requested information which is not yet available there is extracted from the underlying ACR layer and materialized in the database graph.

Before we describe in Section 2.3.2 and 2.3.3 how to define HyperViews over publisher web sites and in Section 2.3.4 how to query these HyperViews, we present the HyperView concept of queries and rules.

2.3.1 Queries and Rules

A query in HyperView is a graph that is to be matched against a data graph. The query graph may contain variables that are bound by a match to the labels of the corresponding data graph elements. Additional constraints on the values of these variables may be specified in the query as well.

The data graph against which the query is to matched may be a virtual graph defined by a HyperView as a mapping from one or more input graphs. Thus, a HyperView executes a query against a virtual data graph by matching the root of the query graph against some already materialized part of the virtual data graph. This initial match is stepwise extended by matching outgoing query edges to outgoing edges of the corresponding data graph nodes. However, a conventional matching algorithm would fail to match a query graph edge against a not yet materialized edge of the virtual graph. In our approach, the HyperView intercepts such an “edge fault” and materializes the missing edges on the fly.

To materialize missing edges, a HyperView contains a set of graph-transformation rules that are associated with schema edges. If a query edge corresponding to a certain schema edge is requested, the rules associated with this schema edge are triggered. When a rule is executed, it issues a query against the input graphs of the HyperView. For each match of this query, the rule extends the output graph with new graph elements. In particular, it adds an edge that is returned

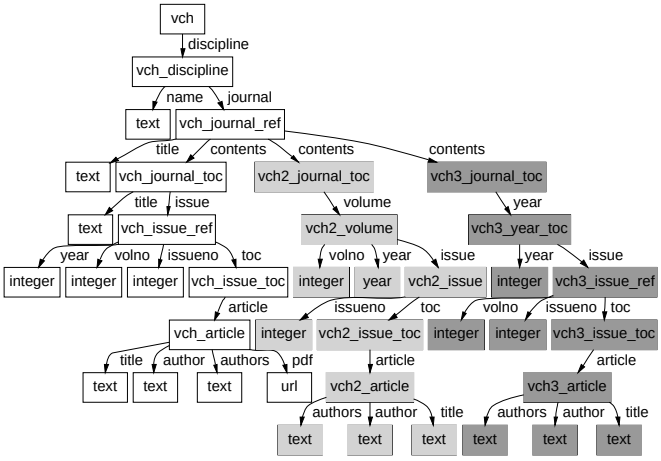

Figure 2.4: ACR schema for Wiley-VCH. Alternative subschemata for journals are indicated by different coloring.

as a match for the query edge that caused the edge fault.

The graph elements referenced in a HyperView rule may belong to different graphs. To be more precise, they may belong to different *clusters* of a global modularized CGDM graph. Each of these clusters conforms to a cluster of the global modularized schema. Informally, we call these schema clusters "schemata". HyperView rules are typed: for each graph element of a rule a corresponding schema element is specified. In particular, rule graph elements are assigned to schema clusters.

HyperView rules are a special kind of graph-transformation rules. Typically, graph-transformation rules consist of two graphs, the left-hand-side (LHS) and the right-hand-side (RHS). A rule is applied to a data graph by finding a match for the LHS and replacing the matching subgraph with the RHS. Several formal approaches for defining the semantics of such rules exist [G. Rozenberg Montanari *et al.*, 1997]. We use the "Algebraic Single Push Out Approach" [Löwe, 1993] as formal basis of our concept of rules. For more details see Chapter 3.

In HyperView, the LHS is a subgraph of the RHS. This means that rules can only add new graph elements, but cannot delete existing graph elements. We call the part of the RHS that is not contained in the LHS the *Update Part* of the rule. We require that there is a single edge leading from the LHS into this Update Part. This edge is called the *Primary Edge* of the rule. The rule is associated with the schema edge corresponding to this primary edge. When the rule is triggered, the LHS is used as a query against the data graph and for each match, the variables occurring in the rule are bound and copies of the elements of the Update Part (including the Primary Edge) are added to the data graph.

Sometimes it is important to reuse a part of a data graph instead of adding new elements to it. This is the case if we want to represent a certain real world entity (e.g., a person) by a unique node of the data graph. To implement this requirement, we have extended our rule concept with "*Reuse Subgraphs*" which designate subgraphs of the update part that are to be added by the rule only if there does not exist a matching data subgraph that can be reused instead. We show an example of applying reuse subgraphs in Section 2.3.3.

We use the notation depicted in Table 2.1 to represent HyperView rules graphically.

2.3.2 Defining a View over the HTML Graphs

We now describe how to establish HyperViews defining the ACR graphs that represent the relevant contents of the underlying Web sites. Each ACR graph is a view over the HTML pages of a single Web site, represented as HTML graphs. Thus these rules reference elements of different

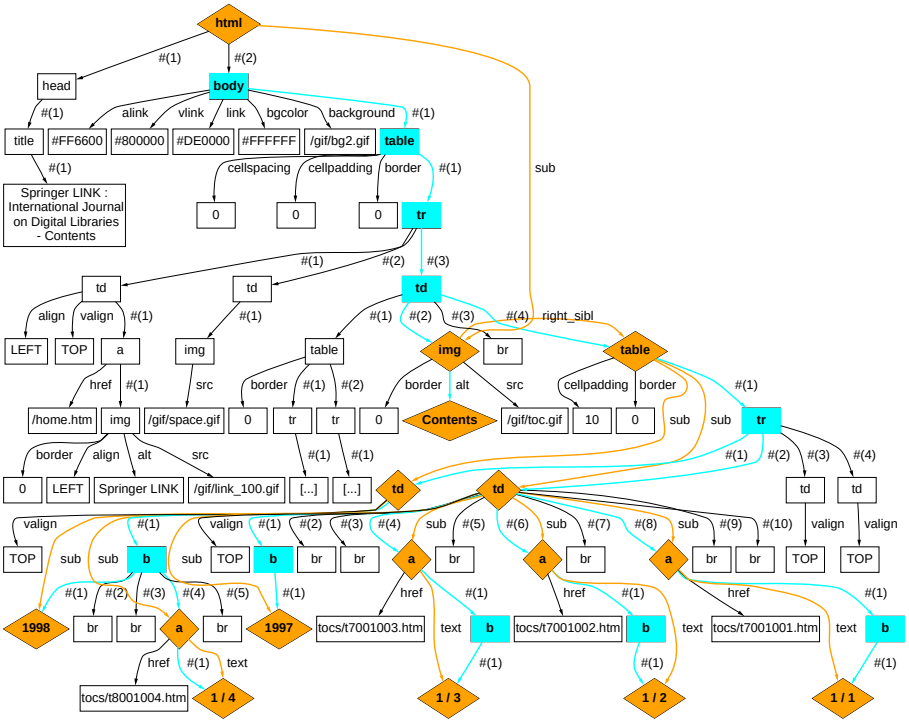

Figure 2.5: HTML graph of the IJODL "Contents" page. Elements relevant to rule application are shaded. Nodes in diamond-shape are direct matches for the nodes occurring in the rules discussed below. Relevant virtual *sub* and *right_sibl* edges are included.

graphs: the HTML input graphs and the ACR output graph of the HyperView. This HyperView is to be developed and maintained by the administrator of the integrated virtual site.

In the example case of the Springer-Verlag, a rule set comprises 15 rules. We explain the view definition mechanism using two of them: Rule get_year_toc creates the annual table of contents as part of the table of contents of a given journal. Rule get_issue_ref creates a reference for each issue that appears in the annual table of contents built by the previous rule.

Building an annual table of contents for a journal. 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 navigation links omitted) has been shown in Figure 2.5.

Rule get_year_toc shown in Figure 2.6 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 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 does not follow the immediate containment relation in the HTML graph, but rather uses virtual *sub* edges expressing a logical sub-node relation. This relation is implemented in a polymorphic way by alternative rules, depending on the type of the source node. Another virtual edge, *right_sibl*, allows direct access to a right sibling of a node.

On application of the rule get_year_toc, a *source* edge pointing to this `<td>` node and an *year* attribute with value *Year* are added to the year_toc node created by the rule.

Rule Part	Style	Color	Graphical Representation
Vertex			Label : Type Type / Label :
Left Hand Side (LHS)	normal	black	
Constraint	normal	black	{ . . . }
Cluster	dashed		SchemaName
Update Part	**bold**	green	
Reuse Subgraph	dotted	green	

Table 2.1: Graphical notation of HyperView rules. Note, that both label and type of a vertex can be omitted. The label can be either a variable name or a constant. The type indicates the label of the corresponding schema node. The label of a cluster box specifies the schema of all graph elements contained in this box.

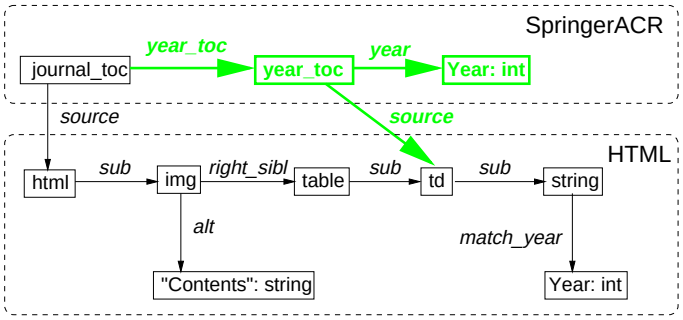

Figure 2.6: Rule get_year_toc. The graphical notation is explained in Table 2.1.

Building references to the year's issues. The second rule, rule get_issue_ref, computes an *issue* edge from the year_toc node (created by rule 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.

2.3.3 Defining a View over the ACR Graphs

After defining the ACR graph of each Web site, we establish the database graph as a view over the set of ACR graphs of the sites to be integrated. To build this view, the administrator of the integrated virtual site specifies a rule set per ACR graph. This rule set maps the ACR graph to the database graph of the application.

The rules defined to build a view over the ACR graphs are of the same nature as those building an ACR graph as a view over the HTML graphs and adhere to the same concepts, as described

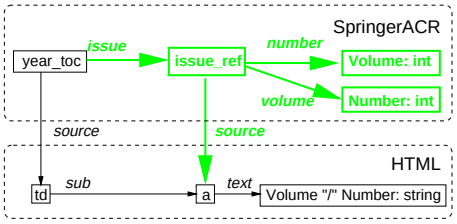

Figure 2.7: Rule get_issue_ref

in 2.3.2.

In DARWIN, the database graph is a view defined by a set of publisher-specific rules. From the rule set for the ACR graph of the Springer-Verlag, we select the rule get_issue as an example. This rule is shown in Figure 2.8. Note that this rule contains reuse subgraphs, denoted by dotted loops.

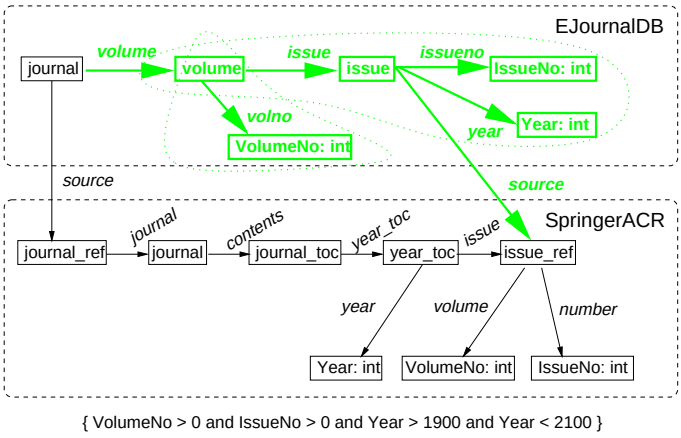

Figure 2.8: ACR rule get_issue. Notation: *Reuse Subgraphs* are indicated by dotted loops.

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 restructuring 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*.

In order to prevent the creation of multiple volume nodes for the same value of *VolumeNo*, we apply the notion of *reuse subgraph*, i.e., a group of graph items that is materialized by a rule only if it is not already part of the result graph. In the case of rule get_issue, this implies that a volume node is identified by journal to which it belongs and its *volno* attribute, and a issue node is identified by the volume node to which it belongs, together with its *issueno* and *year* attributes.

2.3.4 Querying the HyperView system

The user can now query the database graph that hides the heterogeneity of the underlying Web sites. The HyperView system provides a simple but general graph-based language, HVQL (cf. Chahvql), in which transformation rules and queries are expressed using the graph rewriting paradigm.

A query is a rule that does not produce new graph elements, but just returns for each match a binding for the variables occurring 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, the variable bindings are returned as query match.

In Figure 2.9 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 whose title contains the string “Digital Libraries”.

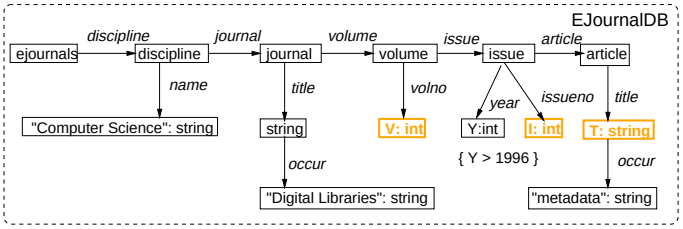

Figure 2.9: Example Query: select all articles on “metadata” having appeared in a computer science journal on “Digital Libraries” after 1996. The vertices corresponding to the retrieved attributes are shown in boldface.

HVQL is not intended for the end-user. It is conceived for the interface designer, who designs the HTML forms to be filled and translated into HVQL queries. It is also intended for the application programmer, who develops software for data imports into a back-end database system. In DARWIN, HVQL is used in the latter way, to specify the information to be imported into the DARWIN catalogue.

In DARWIN, we use information from the digital library database to formulate optimized queries for bibliographical data import 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 the publisher’s home page to the journal page but can access it directly in order to check for a new issue.

Figure 2.10 shows such an import query for the next issue of the “International Journal on Digital Libraries”. From the scheduling data in the catalogue we know that the current issue is number 4 of the first volume. The result of this query is inserted by the HyperNavigator into the DARWIN catalogue.

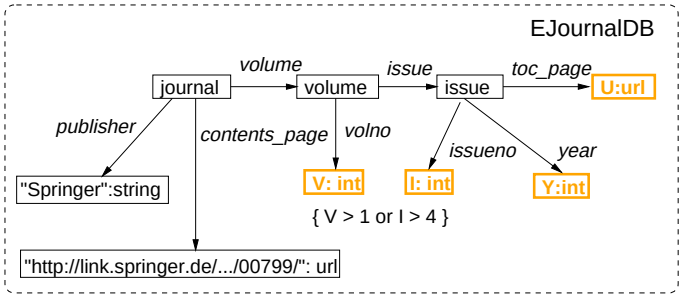

Figure 2.10: Import Query for the next issue of the IJODL

2.4 The Architecture of DARWIN

The architecture of DARWIN can be seen as a configuration of the generic "Warehouse for Internet Data" (WIND) architecture presented in [Faulstich *et al.*, 1997]. In this architecture, data extracted from information sources in the Internet is replicated in a number of central repositories where it can be accessed by the user via a Web interface.

In the framework of DARWIN, only one repository in form of a relational database system (RDBMS) is needed. If we were to decide to create a full text index on article abstracts, we could add a text retrieval engine as a second repository.

Triggered by the so-called Import Scheduling Service described in [Faulstich and Spiliopoulou, 1999], bibliographic data is imported into the repository by the HyperNavigator module. This module is based on the HyperView system. 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 a common schema of electronic journals without knowledge of their actual representation at the publisher Web site. The activation of the wrapper responsible for each publisher is performed internally by the mediator.

2.5 Summary

In this chapter, we have demonstrated the key concepts of the HyperView approach by presenting the application of the HyperView methodology for integrating publisher Web sites of electronic journals. The presentation in this chapter is based on the work described in [Faulstich and Spiliopoulou, 1998] and [Faulstich and Spiliopoulou, 1999] that has been carried out in the framework of the DARWIN project. In this project, an online catalogue of electronic journals is being established. The HyperView system is used to extract meta data from these journals to keep the catalogue up-to-date.

Possible extensions of this catalogue are to:

- wrap existing search interfaces of publisher WWW servers to support distributed full-text retrieval
- integrate notification services offered by some publishers. E.g., CDF channels could be easily wrapped using the HyperView system
- add a Web interface to the HyperView System to create a version of the DARWIN catalogue which is based completely on the HyperView methodology. The issue of how to establish such a HyperView Web interface is addressed in Chapter 6.