Chapter 9

Discussion and Outlook

This thesis describes the HyperView approach for the integration of semistructured data sources into virtual Web sites.

Today, the Web offers information on the same topic in many different Web sites. Since these sites tend to be incomplete or oriented towards certain aspects, the user often has to collect and combine information from various Web sites in order to answer a specific question or justify a decision. This task can be quite tedious since different Web sites are structurally heterogeneous and related information in different Web sites is most often not connected by hyper-links.

This situation motivates the creation of *virtual Web sites* for fixed application domains that integrate on the fly data from different source Web sites and present it in a uniformly in a way that fits best for the chosen application domain. The HyperView approach provides a methodology, a formal framework, and a software environment for building such virtual Web sites.

The structure of the WWW lends itself very well to be modeled with graphs. The HyperView approach has gone a step further: not only does it represent semistructured Web documents as graphs, but also the abstract content of Web sites and the data integrated from different Web sites. Graph-based data model CGDM of HyperView allows all these graphs to be seen as components (*clusters*) of a global data graph. Since it is not feasible to mirror the complete contents of the underlying Web sites, the HyperView approach implements the steps of data extraction, integration, and presentation as views that define mappings between graphs at different levels of abstraction. These levels are reflected by the layered architecture of the HyperView System that comprises the HTML layer for representing HTML pages, the ACR layer for data extracted from each Web site, the DB layer where data from different sources is integrated, and the UI layer that presents the integrated data in form of the pages served by the virtual Web site (see Figure 9.1).

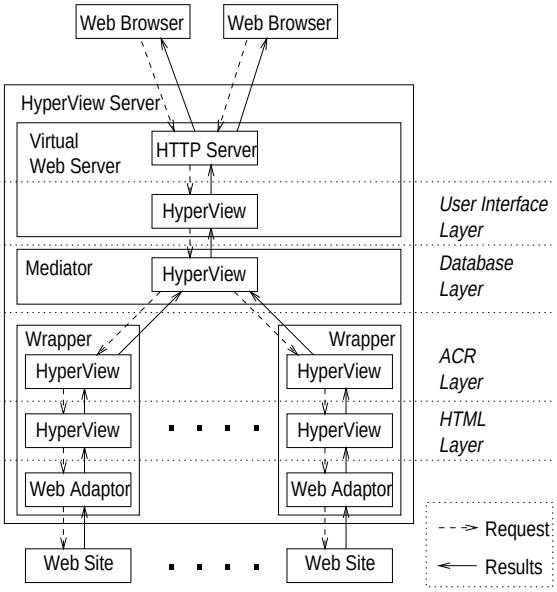

Figure 9.1: The HyperView architecture. (Copy of Figure 1.1, repeated for convenience.)

Views are defined by sets of graph transformation rules. We call these views HyperViews. The rules of a HyperView are activated by a demand-driven mechanism that results in the propagation of page requests for the virtual Web site to page requests sent to the underlying Web sites. The activated rules extract data from the loaded pages into the source-specific ACR layer, integrate this data in the DB layer, and format it as Web documents in the UI layer. In order to avoid pages from the source Web sites to be loaded more than once, views are incrementally materialized at each layer. This yields a cache mechanism that considerably speeds up query execution.

The HyperView System provides a platform on top of which views can be implemented and virtual Web sites can be built. It is implemented in Prolog. The formal concepts like graphs,

queries, rules, and HyperViews are denoted in the language HVQL and translated into corresponding Prolog constructs. The HyperView System exploits heavily the unification and backtracking mechanism of Prolog in order to implement efficient graph matching. HyperViews are encoded as logic programs. External software tools are used to load and serve pages in order to couple the HyperView System with the underlying Web sites and the user's Web browsers.

This concluding chapter is organized as follows: In Section 9.1 the HyperView approach is compared with related work. Section 9.2 discusses future applications of the HyperView approach in the context of XML. Open issues are presented in Section 9.3. Section 9.4 summarizes the contributions of this thesis and gives an outlook. The acknowledgments in Section 9.5 conclude the chapter.

9.1 Related Work

This overview on related work deals with projects on semistructured data. We first present data models and attempts to introduce a notion of schemata for semistructured data. Then we deal with methods for extracting information from semistructured documents such as HTML pages and how for querying the WWW. Then we present projects that focus more on integrating the extraction and query results.

We conclude with a comparison of projects related to this work and a discussion of the contributions of the HyperView project compared to these projects.

9.1.1 Data models and schemata for semistructured data

The two most early and prominent efforts on data models for semistructured data are (i) the works [Quass *et al.*, 1996, Abiteboul *et al.*, 1997, Goldman and Widom, 1997] in the TSIMMIS [Garcia-Molina *et al.*, 1995] project at the University of Stanford and (ii) the works of Peter Buneman [Buneman *et al.*, 1996], [Buneman *et al.*, 1997],[Buneman, 1997] at the University of Pennsylvania.

Many other groups and projects (e.g. Strudel or YAT) use similar data models.

OEM, LORE, LOREL, Data Guides. The Stanford database group has developed the OEM data model and the LOREL query language [Abiteboul *et al.*, 1997] for the Lightweight Object Repository (LORE) [Quass *et al.*, 1996]. In OEM, data are represented as edge labeled graphs whose leaves contain data atoms. OEM also defines a syntax for textual representation of such graphs. LOREL supports regular path expressions and path variables. Other important features are the use of automatic type coercions and the robustness against missing data.

In [Goldman and Widom, 1997], a concept of *a posteriori* schema called "Data Guide" is introduced. A Data Guide is a graph which summarizes an OEM data source in the sense that its set of edge label sequences is identical to that of the data source. One of the main disadvantages of this concept is that data sources may use edge labels to store data (instead of attribute names). In this case, the size of the data guide will be comparable to that of the database. A data guide based on *classes* of edge labels is not foreseen. Another problem are the high costs for creating and maintaining data guides.

Recently, support for querying XML documents has been added to LORE [Goldman *et al.*, 1999]. Roughly speaking, XML documents are represented in OEM by their parse trees. The children of a document element are reachable via edges labeled by the tag names of the children. Element attributes are (similar to the HyperView System) reachable via edges carrying the attribute name. The XML representation goes beyond pure parse trees in that cross-links within documents are resolved by creating cyclic structures.

UnQL. Peter Buneman uses a data model based on unordered trees which stores data only in edge labels. In order to support general graphs, a reference mechanism for identifying tree

leaves with roots of (sub-)trees is provided. The query language for this data model is named UnQL[Buneman *et al.*, 1996]. It supports regular path expressions, recursive restructuring of trees, and nested queries. The semantics of UnQL are defined in terms of the UnCAL calculus.

In [Buneman *et al.*, 1997], a schema concept for semistructured data is introduced. Schemata are modeled as graphs which are labeled with unary predicates. According to this approach, a data graph conforms to a schema graph if there exists a simulation between the data graph and the schema graph that associates their respective root nodes and relates every data graph edge to one or more schema edges such that its label fulfills the predicates of these schema edges.

One problem with this approach is that the empty data graph conforms to every schema graph, whereas adding a single edge that can not properly be mapped causes a conforming graph to loose this property. Also, a concept for schema modularization and inheritance is missing.

9.1.2 Data Extraction from Semistructured Documents

O2Yacc [Abiteboul *et al.*, 1993, Abiteboul *et al.*, 1998] is a tool for querying, and updating data in structured files. It uses a YACC-like grammar annotated with database insert statements. Using O2Yacc, data can be extracted from files by simply querying them from within an O_2 database. However, since this tool relies on grammars that *fully* specify the whole document structure, it seems too rigid to deal with semistructured documents in the Web. Moreover, it does not provide any particular support for the treatment of HTML pages.

InfoExtractor [Smith and Lopez, 1997] is a tool developed by INRIA / Bull. This tool as well as the **web extractor [Hammer *et al.*, 1997]** built by the TSIMMIS group both use a hierarchy of regions defined by regular expressions to analyze the structure of a document and extract parts of it as result. Hammer et al. argue that more complex analysis methods than regular expressions are too costly. In typical applications however, processing time can be neglected in comparison to network delay and WWW server response time. Even though a pattern matching approach is in general robust against adding additional information, the use of regular expressions at the HTML level makes the system vulnerable against even minor representational and layout changes.

The wrapper generation toolkit [Ashish and Knoblock, 1997] of Ashish and Knoblock works in a semi-automatic way by guessing the structure and interesting regions of example pages. Before a parser for each page type can be generated these guesses have to be reworked manually, since they are based on rather simple heuristics. The user has to specify a function which maps query arguments to URLs. The resulting URLs are used to retrieve the relevant pages which are to be parsed by the wrapper. The SIMS architecture [Arens *et al.*, 1993] is used to integrate the wrapped information sources. This approach does not seem to be very robust against source evolution since the hierarchical structure of HTML is ignored. Also, the issue of complex traversal paths – which are often necessary to select relevant pages – is not discussed.

EDITOR [Atzeni and Mecca, 1997] describes the procedural EDITOR language for extraction and restructuring arbitrary documents. It is used in the context of the ARANEUS project discussed below. EDITOR uses the operators regexp-search, cut, copy, paste, replacement. These operators work on a text region of the input document or on temporary clipboards, and create a result document by editing the input document. Java control structures make EDITOR programs computationally powerful. In fact, the EDITOR language is implemented as a small subset of Java.

However, the concept of a procedural language which matches and manipulates character sequences seems to be on a very low level of abstraction. This raises the concern that EDITOR programs are difficult to specify and maintain, and provides only low robustness against deviations and layout changes. Those concerns may have stimulated the development of the MINERVA project discussed next.

MINERVA [Crescenzi and Mecca, 1998] combines parsing of regular expressions[1] with an exception handling mechanism based on the EDITOR language. It is used in ARANEUS as an alternative to purely EDITOR-based wrappers.

If a production can be applied successfully it assigns strings to all non-terminals that occur in the production. If a production having an exception clause fails, the EDITOR code contained in this exception clause is executed instead. This error recovery code may directly assign strings to the nonterminals or change the document text itself and then resume the parsing process. As a result of parsing a document, new tuples are inserted into a single database table. This approach seems to be inappropriate in the case that the contents of a document are too complex to be represented as tuples of a single relation.

Writing MINERVA grammars can might be quite tedious since regular expressions for matching HTML tags have to be specified explicitly and productions for text regions enclosed by a pair of tags have to specify the closing tag as look-ahead in order not to leave the text region accidentally. Moreover, the exception code tends to be more voluminous than the grammar itself. These drawbacks imply that MINERVA is appropriate for rather simple wrappers only.

NoDoSE [Adelberg, 1998] aims at extracting structured data from semistructured text documents in general.Application examples are listings produced by simulation packages, e-Mail messages etc. It has an extensible architecture which can be supplied with various analysis modules.

First, the designer creates an object-oriented schema. Then he uses the system to examine a single document and decompose it hierarchically into smaller fragments. For each text fragment, the designer has to indicate the type of the data to be extracted from it.

The system comprises a mining component which tries to infer the grammar underlying each document part by generating and evaluating parsing rules. Currently, the miner is limited to homogeneous lists and records having a fixed ordering of attributes. Errors of the miner can be corrected manually. A grammar developed for a single document will then be validated (and possibly adapted) for other documents of the same type. Finally, the user has to specify an output format for the extracted data.

For HTML documents, parse trees are constructed which have to be edited and annotated by the user before feeding them to the text miner.

Jedi [Huck *et al.*, 1998] is an information-extraction tool using an error-tolerant parsing technique based on ambiguous context-free grammars. For each production of a grammar, a fall-back alternative accepting arbitrary text is added. The Jedi parser uses a non-deterministic stack automaton which is executed breadth-first. In each step, only the so far most specific derivation leading to a state of the automaton is kept. This pruning helps avoiding the combinatorial explosion caused by the ambiguities of the grammar. The parsed text is processed by executing code fragments attached to the productions.

In contrast to HyperView, Jedi does not take into account the hierarchical structure of HTML text and its rules are not robust with respect to textual reordering.

World Wide Web Wrapper Factory (W4F) [Arnaud Sahuguet and Fabien Azavant, 1999] is a recent project for the generation of Java wrappers in the group of Peter Buneman. It uses a three-step approach consisting of phases for retrieval, extraction, and mapping. The simple retrieval language specifies where and how to load a HTML document. The extraction language HEL operates on parse trees but supports both hierarchical and text-order navigation. Regular pattern matching for text fields is also available. A special split operator can be used to specify choice-points for backtracking in order to support the extraction of multiple values into a nested list structure (NLS). NLS are Java data structures. The mapping of these generic data structures

[1]The authors present MINERVA grammars as an extension of LL(1)-grammars. However, the formal semantics presented in the paper clearly show that these grammars are limited to regular expressions.

into application data has to be coded in Java. Alternatively, a template language described in [Sahuguet and Azavant, 1999] can be used to transform NLS data into XML documents which can be made available on the WWW.

The W4F toolkit includes CGI-scripts for page annotation that assist in specifying the retrieval and extraction rules. A similar program for assisting the mapping specification is planned, but not implemented yet. Furthermore, the so-called "extraction wizard" is missing a schema discovery component. This leads to over-specific rules which are not robust or even general enough for the intended class of documents and hence have to be redesigned manually.

The extraction phase of W4F compares to the HTML views of HyperView and the mapping phase to ACR views. Other than in HyperView, different mechanisms are used at each stage. On the other hand, W4F offers a tight integration with Java. In principle, tools similar to that of W4F can be implemented for HyperView and HVQL as well.

9.1.3 Querying the Web

WebSQL, W3QS, and WebLog are relational languages for querying the Web. The first two are extensions of SQL. In WebSQL[Mendelzon *et al.*, 1997], the web is modeled by two (virtual) relations "Document" and "Anchor". Regular path expressions allow to specify link traversals between HTML documents subject to link labels and the textual context of anchors in the documents. Entry documents can also be found via external search machines by specifying keywords. No attempt is made to interpret the contents of pages and to extract data except of the links.

W3QS [Konopnicki and Shmueli, 1997] is similar to WebSQL, but does not support regular path expressions. The W3QS system is simply extensible by external calls to Unix commands and the automatic submission of HTML forms from within queries. External filter programs can be used to extract attributes from pages, but each page is still modeled as a relational tuple.

WebLog [Lakshmanan *et al.*, 1996] is a concept for a deductive database on top of some computed base relations modeling the WWW (similar to WebSQL and W3QS). Deductive rules are used to formulate queries and to specify new pages containing information extracted from other pages. However, only the link structure but not the syntactical structure of pages is considered.

All these query languages have the disadvantage that only the link structure and the text content of web pages can be queried, but not their HTML structure, which reflects much of their semantics.

WebOQL [Arocena and Mendelzon, 1998] is a query language for web documents which supports to queries against the hierarchical structure of web pages and allows the restructuring of query results into new HTML documents. It uses an ordered edge-labeled tree data model called *HyperTrees* for representing the structure of HTML pages as abstract syntax trees. Inter-tree edges ("external arcs") are used to model hyper-links. A set of trees associated with URLs is called a *Web*. As an intuitive specification method for complex restructurings, so-called "Document Patterns" can be used. These are HTML templates that can automatically be translated into WebOQL queries. WebOQL queries can be used to generate HTML code. Although it is possible to combine WebOQL queries to define more complex views, there is no notion of schema and a methodology of how to integrate multiple Web sites is missing.

FLORID [Ludäscher *et al.*, 1998a] is a deductive object-oriented database system based on F-Logic [Kifer *et al.*, 1995]. In [Ludäscher *et al.*, 1998a] and [Ludäscher *et al.*, 1998b], extensions of FLORID for querying the Web are presented.

The web pages to be accessed are described by a program consisting of FLORID rules. When this program is evaluated using the bottom-up evaluation strategy of FLORID, the database is populated by loading all pages described by the program. Pages are represented in the database by certain metadata attributes (like title, author etc.) and the set of hyper-links contained in each page. During program evaluation, rules may become applicable depending on the contents of the pages loaded so far. Hence we can speak of *data-driven* page materialization.

However, the database can be queried only *after* it has been populated, which can be characterized as a Data Warehouse approach. Hence FLORID does *not* support *demand-driven* materialization as offered by HyperView.

General path expressions for querying semistructured data are implemented using FLORID rules, similar to the meta edges defined in HVQL. In [Ludäscher *et al.*, 1998b] additional rules for parsing XML-documents are proposed. These rules apply in principle to HTML pages as well, yielding an expressiveness for queries similar to that of HVQL rules. However, FLORID does not abstract from the syntactical structure of the underlying HTML pages since a layered view concept similar to that of HyperView is missing.

9.1.4 Integration of Heterogeneous Data Sources

ARANEUS [Paolo Atzeni, 1997] is a tool-box for the management of data from the Web. It uses the EDITOR language [Atzeni and Mecca, 1997] and recently MINERVA [Crescenzi and Mecca, 1998] grammars (cf. Section 9.1.2) to extract data from semistructured HTML pages modeled in the hierarchical ("page-oriented") ARANEUS Data Model (ADM). The language ULIXES is used to specify and navigate the link structure of the pages and to build relational views over the extracted data. The generation of Web pages from these views is specified in the language PENELOPE. Essentially PENELOPE produces one or more nested relations from a single relational table and creates for each resulting nested tuple a page using a HTML template.

TSIMMIS [Garcia-Molina *et al.*, 1995] , "The Stanford-IBM Manager of Multiple Information Sources" is an integration architecture for heterogeneous information servers wrapped in *translators* that transform them into OEM data sources. *Mediators* are used to integrate information provided by wrappers or other mediators.

In [Papakonstantinou *et al.*, 1996b], MedMaker, a rule based approach to integration of OEM data sources is described. The "Mediator Specification Language" MSL is used to build mediators in the TSIMMIS project. MSL rules can be seen as tree rewriting rules. Queries are represented as rules as well. To process a query, its body is unified with matching rule heads. The instantiated rules result in queries that are executed against wrappers after applying some algebraic query optimization techniques.

In [Papakonstantinou *et al.*, 1996a], this approach is extended by introducing the notion of "Object Fusion" which means the merging of objects that represent the same real world entity in different sources. This is achieved by introducing logic object identifiers similar to the Skolem functions of YAT and Strudel (see below).

Strudel [Fernandez *et al.*, 1997, Fernandez *et al.*, 1998] is a system supporting the management and restructuring of Web Sites. The idea is to provide a collection of HTML documents as a view over a semistructured database which itself may be an integrated view of multiple external data sources. External data sources, the integrated view, and the Web Site itself are modeled as graphs, using a data model similar to OEM. The existence of appropriate wrappers which provide this graph representation of external sources is assumed. In [Fernandez *et al.*, 1997], the query language and query processor of Strudel are described. The language especially supports the construction of new graphs from the query results using Skolem functions. It does not have a schema concept to guide query and view formulation. Also, an eager query evaluation strategy is used which may result in materializing unnecessarily large view graphs. In [Fernandez *et al.*, 1998], the methodology for managing Web Sites with Strudel is discussed in more detail.

YAT [Cluet *et al.*, 1998, Jerome Simeon, 1998]. The YAT system is a tool for specifying data conversions. The YAT data model uses ordered labeled trees which can be linked by references pointing from leaves to root nodes. 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. Skolem-functions

are used to avoid the creation of redundant tree structures. Rules are priorized according to how specific they are w.r.t. the instantiation hierarchy. Currently YAT does not use HTML pages, but O_2 objects and semantic-oriented XML documents with known DTD as input. Data extraction for layout-oriented input formats such as HTML requires support for generalized path expressions (GPEs) as proposed for instance in [Abiteboul *et al.*, 1997]. This is not available in YAT due to the fact that – differing from the HyperView System– YAT does *not* support the recursive rules necessary to implement GPEs.

YAT does not offer a layered view concept like that of HyperView. Although YAT modules can in principle be combined to form a wrapper-mediator architecture, YAT is not intended as a tool for Web site integration.

ARIADNE [Knoblock *et al.*, 1998]. ARIADNE is a successor of the SIMS project [Arens *et al.*, 1993]. It uses wrappers generated by the "*Wrapper Generation Toolkit*" [Ashish and Knoblock, 1997] described in Section 9.1.2 for single (sub-)pages. These wrappers are attached to classes in a domain schema. The knowledge representation language Loom KR is used to express schema and queries against it. The emphasis of this approach lies on the creation and optimization of query execution plans against this domain schema.

Each wrapper has a number of input and output attributes which are taken into account by the query planner. Mapping tables / functions and utility functions are also treated in a uniform way as wrappers. However, navigations of arbitrary depth (like collecting all answers of a search machine from a chain of linked pages) cannot be expressed in the schema and must be encapsulated within a single wrapper. In this approach, domain schema and site structure are not decoupled completely, and it is not clear how alternative sources with similar information, but different site structure can be modeled.

9.1.5 Related applications of Graph-Transformation techniques

There are only few works that present applications of graph-transformation techniques in the field of databases.

In [Claßen and Löwe, 1995] a formal framework for the schema evolution in object-oriented databases is presented. This work is relevant from a technical point of view since the presented typing concept is similar to our notion of conformance to a schema. There, a many-sorted attribute algebra is turned into an untyped base algebra, and the typing information is internalized into the attribute algebra of the schema graph. An instance of a schema is an untyped graph together with a typing morphism (*interpretation*) into the schema graph. The contribution of HyperView regarding the typing concept lies in the modularization of graphs and schemata into so-called *clusters*.

A design environment for migrating relational to object-oriented database schemata is described in [Jahnke *et al.*, 1996]. Graph grammar rules selected by the database administrator express generic correspondences between relational and object-oriented constructs. Together, they constitute a schema translation.

A HyperView system solves a task similar to the ones discussed in [Claßen and Löwe, 1995] and [Jahnke *et al.*, 1996], the transformation of a data collection. The main differences to the mentioned works are that HyperView (i) is able to deal with semistructured data lacking an explicit schema, (ii) integrates data from *several* heterogeneous sources and (iii) offers a novel demand-driven incremental view mechanism.

In [Bergholz and Freytag, 1999] a system for querying semistructured data is presented. This system shows similarities to the HyperView approach: It uses a graph-based data model and expresses queries as graphs labeled with variables that are to be matched in the data graph. Restructuring of query results is supported by *restructuring queries* that can be viewed as algebraic Single Pushout Rules. However, while this work primarily aims at efficiently querying semistructured data, the HyperView approach goes much further by providing its demand-driven view mechanism and using graph-transformation techniques for building virtual Web sites.

9.1.6 Comparison with HyperView

The task of integrating semistructured data sources into a virtual Web site can be decomposed into the phases of data extraction, integration, and presentation. For each of these phases there exist various approaches. In Table 9.1 the most relevant projects are listed. In the following paragraphs, these projects are compared with respect to the mentioned phases.

Data extraction. Most approaches for data extraction represent Web documents as strings and use regular expression matching to find substrings relevant for extraction. We argue that these approaches neglect important structural information inherent in HTML documents. Therefore we have adopted a graph representation based on HTML parse trees. This approach is shared only by more recent projects like WebOQL [Arocena and Mendelzon, 1998], YAT [Cluet *et al.*, 1998] and W4F [Arnaud Sahuguet and Fabien Azavant, 1999].

Furthermore, we argue that regular expression matching tends to be less robust than the graph matching used in the HyperView approach. This is due to the fact that parse trees abstract from textual details such as white space and ordering of attributes. HyperView goes beyond WebOQL, YAT, and W4F in providing a set of virtual edges defined over HTML parse trees (see Table 5.3) which further abstract from the details of parse trees and implement logical relationships between document elements that are not explicit in the hierarchical document structure. This way, the HyperView System provides an effective abstraction from layout details and thus increases the robustness of HyperView-based wrappers.

Data integration. Integration of *semistructured* data is often tackled with query-based approaches. In the case of Web query languages like WebSQL [Mendelzon *et al.*, 1997] and W3QS [Konopnicki and Shmueli, 1997], the integration is ad-hoc only. WebOQL is primarily a *query language,* not a data integration tool. It can be used to a limited extent to achieve data integration, but the lacking schema concept seems to make the specification and maintenance of the necessary queries difficult. The same holds for MedMaker [Papakonstantinou *et al.*, 1996b] and for Strudel [Fernandez *et al.*, 1997] which do not use any schemata to model input and output data.

In ARANEUS [Paolo Atzeni, 1997], the task of integration is left to relational views. However, the large body of approaches to integration of relational data does not apply well to semistructured data due to the inherent variability and irregularity of such data.

Rule-based approaches like HyperView, YAT [Cluet *et al.*, 1998], or FLORID [Ludäscher *et al.*, 1998a] support the definition of views consisting of rule sets. Such rule-based views have the advantage that they are easier maintainable and extensible than views defined by single monolithic queries. In contrast to approaches like Strudel, FLORID, HyperView supports the demand-driven integration of data.

Compared to YAT, rule sets in HyperView can be designed in a way that ensures maximal independence of the rules. This can be achieved by using alternative rules to compute the same outgoing edges of a schema node. The failure of one of these rules might be due to missing data or structural evolution; this does not affect the computation of the other rules. This means that the HyperView does not break as a whole, but just yields less complete information.

Data presentation. Data presentation nowadays primarily means publishing the integrated data in the Web. This amounts to generating HTML or XML documents and serving them to the user's browser on request. A static approach is taken by Strudel which generates all pages of a virtual web site in advance. In general this is not feasible for external information sources. Moreover, HTML forms cannot be supported this way.

WebOQL uses document patterns to facilitate the formulation of queries that return HTML pages as results. However, this approach aims not so much at defining complex virtual web sites but at providing certain focused information services.

The PENELOPE language used in the ARANEUS project uses relational nesting operators to format sets of tuples from a relational table into HTML pages.

In YAT, incoming HTTP requests trigger rules that generate HTML pages by matching information in the input data (which may be loaded from external wrappers).

Project	Extraction	Integration	Presentation	Data Model	Materialization	Remarks
HyperView	**HVQL (navigation on HTML graphs)**	**HVQL**	**HVQL**	**clustered graphs**	**demand-driven**	**formal framework**
InfoExtractor	hierarchical REs	—	—	frames	—	—
web extractor (TSIMMIS)	hierarchical REs	—	—	OEM	—	—
MedMaker (TSIMMIS)	—	MSL	—	OEM	—	—
EDITOR (ARANEUS)	EDITOR (procedures + REs)	—	—	Text	—	formal computational model
MINERVA (ARANEUS)	regular LL(1)-grammars + EDITOR (exception handling)	—	—	Relational	yes	—
ULIXES (ARANEUS)	navigation in ADM trees	—	—	ADM → relational	—	—
PENELOPE (ARANEUS)	—	—	nesting ops. + HTML templates	relational → ADM	—	—
NoDoSE	Generated parser	—	—	object-oriented	—	—
Jedi	error tolerant parser	—	—	Java	—	—
W4F	HEL (navigation on parse trees)	—	—	Nested list structures	—	—
WebOQL	WebOQL (navigation on parse trees)	WebOQL	WebOQL (document patterns)	HyperTrees (cross-linked trees)	none	—
FLORID	Links, REs	F-Logic	Command line	F-Logic	in advance, data driven	formal framework: F-Logic
Strudel	(external wrappers)	StruQL	HTML templates	graphs	in advance	—
YAT	(external wrappers + YATL)	YATL	YATL	cross-linked trees	—	formal semantics; no GPEs
Wrapper Generation Toolkit (ARIADNE)	generated parser (Lex+Yacc)	—	—	hierarchical	—	—
ARIADNE	(external wrappers)	Loom KR	—	Loom KR (object-oriented)	—	—

Table 9.1: Overview on related projects

Similarly to YAT, HTTP requests reaching a HyperView virtual web site triggers rules which produce HTML graphs that are formatted as HTML documents. Differing from other approaches such as WebOQL and YAT, in HyperView rules for the presentation layer are specified against the database schema, rather than against the structure of the (HTML) input data.

Closely related approaches. Most of the approaches presented in this chapter do *not* cover all three subtasks necessary for the integration of semistructured information sources, i.e., extraction, integration, and presentation. For instance, TSIMMIS supports data extraction and integration, but not the creation of virtual web sites. YAT and Strudel do not support data extraction from HTML, even though adding this functionality seems possible.

MedMaker, WebOQL, and Strudel are missing a schema concept which makes it difficult to build and maintain complex virtual web sites. In contrast, the schemata at the different layers of the HyperView architecture increase the maintainability of views since they effectively decouple view definitions at different levels. Moreover these approaches are lacking a formal semantics similar to the formal framework of HyperView which forms one of the main contributions of this thesis.

In ARANEUS, heterogeneous approaches are combined into a pragmatic, but complete solution. This has the disadvantage that different languages and data models are needed. In contrast to this, HyperView offers a homogeneous approach using a graph-based data model and graph-transformation techniques that is applied uniformly through all stages. The use of a single view definition language also results in a better maintainability. The uniformity of the HyperView approach is reflected in the architecture which consistently uses the same view mechanism to map between different levels of abstraction. This architecture and the HyperView methodology constitute unique contributions of this thesis.

9.2 Future Applications: XML & RDF

In this section possible applications of the HyperView System in the context of XML are discussed. We present several XML-related standards and show how HyperView can be used to deal with the proposed document formats. We further discuss how application specific XML formats may alleviate the data extraction problem for building virtual Web sites, without resolving it completely. Similarly, standardized XML formats may reduce the effort for data integration, although the need for integrating heterogeneous documents will remain.

Additionally, the HyperView System may be used as a platform for implementing standards for the processing of XML documents (XPointer, XQL, XSL) and metadata on Web resources (CDF, RDF).

Since most of the standardization efforts discussed here are preliminary and most of the corresponding specifications are drafts, this chapter has the character of an exploratory study.

9.2.1 XML

SGML is a framework for the definition of structured document formats and for encoding structured documents. One particular format defined using SGML is HTML. XML is a simplified variant of SGML that aims at defining document formats for data exchange, in particular via the WWW. Such formats are also called "XML *applications*". Examples of XML applications are the "Mathematical Markup Language" (MathML [MathML1.01, 1999]), the "Scalable Vector Graphics" (SVG [SVG1.0, 1999]) and the "Resource Description Framework" (RDF, [W3C, 1999a]).

XML has the potential of introducing standardized *semantic* formats to the WWW to supersede the layout-oriented HTML format and proprietary formats that require specific software to be viewed and manipulated.

XML formats can be called "semantic" formats since document elements in this format can represent concepts of some application domain rather than formatting instructions. However, it is also possible to "abuse" XML to devise layout-oriented formats. For instance, the XHTML

effort [XHTML1.0, 1999] is currently redefining HTML as an XML application, and the SVG project aims at defining a XML format for vector graphics. Indeed, there is a continuum between pure semantic formats and pure presentation formats. For instance, MathML offers both presentational *and* semantic document elements.

In theory, when wrapping a single XML document in a pure semantic format, there is no need to *extract* its semantic content, e.g. into a HyperView ACR graph. Rather, the document parse-tree itself may serve as ACR graph. However, this is an idealized assumption. We should expect that Web sites of the future will serve a mixture of HTML pages and XML documents. Moreover, many XML documents will contain layout-oriented parts that have to be mapped into a semantic representation. Another issue is the need for normalization that arises from alternative representations allowed by a format. For instance, RDF allows different encodings for the same content (see Section 9.2.7). Therefore the problem of information extraction from documents into the ACR layer may become simpler, but will not cease to exist. Hence systems like the HyperView System will be needed for wrapping XML documents in the future.

XML formats are "open" in the sense that XML documents are not binary data and document type definitions (DTDs) are usually available. Many XML formats are defined as a standardization effort in some community of practice. However, there is no organizational framework that prevents software vendors from the definition of proprietary or competing XML formats. For instance, it is possible that publishers will define a common XML format for electronic journals, but it is just as likely that many publishers will develop proprietary formats of their own. Even if common XML formats appear in certain application areas, there will be intersections between these areas where facts expressed in one format must be combined with facts expressed in another format. This means that the need for data integration will persist, but the data providers will provide a better basis for it.

9.2.2 XML Parsing

XML shows some syntactic differences to SGML, namely:

no markup minimization: opening and closing tags must always be balanced.

empty elements: a special syntax is used for elements with empty content (`<empty/>` instead of `<empty></empty>`).

name spaces: element types defined in other DTDs can be referenced in a DTD. Documents elements conforming to these "external" element types use the `<abbr:tagname>` notation where `abbr` has to be defined as a shorthand for a reference to the external DTD.

The existing HTML parser in the HyperView System has been extended in a way to support basic XML parsing. This was straight-forward since the parser is generic in the sense that it does not depend on the fixed tag set of HTML. Just like HTML documents, XML documents are represented as trees whose nodes represent document elements (i.e., tag pairs or singleton tags). These nodes are connected via numbered *#(i)* edges to their children nodes. Plain text (PCDATA) is represented by nodes labeled with the text strings. Attributes are represented by edges labeled with the attribute name and pointing to a node labeled with the attribute value. Computable edges defined for HTML (e.g., for dereferencing hyper-links) can be used for XML as well.

To support full XML parsing, support for handling name spaces has to be added. Instead of extending the existing parser, it can be investigated whether a DOM implementation can be coupled with the HyperView System. DOM, the *Document Object Model* [W3C, 1998] defines an API for accessing XML and HTML documents. A DOM implementation is a library that includes a parser and provides several access methods that support navigation in the parse tree. In the HyperView System, the DOM API could be wrapped in such a way that a parsed document appears as a graph cluster.

One might speculate that DOM provides a canonical data model for XML. However, a tree representation seems too low-level for this purpose. For instance, cyclic structures have to be

resolved manually by following intra-document links. Moreover, although XML documents are strictly typed by their DTDs, DOM does not support any form of type checking or lookup of DTD declarations.

9.2.3 XML DTDs and schemata

XML formats are defined by *Document Type Descriptions* (DTDs). A DTD defines *elements types* that correspond to the tag names that may occur in a document. For each element type, the admissible attribute names and the allowed element content is specified. The content of an element is specified by a so-called *content model* which essentially is a regular expression over tag names. In Figure 9.2 a small example DTD for book references is given. In this format, a book entry consists of a list of authors, the title, and the name of the publisher. A book has attributes that indicate its publication year and its ISBN number.

```
<!ELEMENT book (author+, title, publisher)>
<!ATTLIST book year CDATA
               isbn CDATA>
<!ELEMENT author PCDATA>
<!ELEMENT title PCDATA>
<!ELEMENT publisher PCDATA>
```

Figure 9.2: Example: the book DTD

DTDs have two main disadvantages, namely that attribute value types cannot be specified and that the DTDs themselves do not form valid XML documents. The former poses a problem when using XML to exchange structured data rather than documents, while the latter prevents tools for XML documents (such as for instance a XML query processor) to be applied to XML DTDs.

The XML Schema initiative [W3C, 1999d] at the W3C tries to remove the mentioned shortcomings of DTDs by introducing a XML format for defining other XML formats. This proposal extends the functionality of DTDs by a mechanism for the specification of the types of attribute values and string ("PCDATA") contents of document elements. It offers a number of predefined atomic data types such as date, time, as well as a language for the definition of new data types and their string notation. Moreover it allows the definition of abstract element types (so-called *archetypes*) that can be used and refined to define concrete element types.

Compared to a DTD, the XML schema format is more verbose as one can see for the book example in Figure 9.3. Note, that in this schema the types of the attributes `year` and `ISBN` are restricted to appropriate predefined data-types.

Both DTDs and XML schemata can be understood as schemata for semistructured data rather than as grammars defining documents. Similarly to the straight-forward representation of HTML and XML documents as directed graphs in the HyperView System, a DTD or an XML schema can be represented as a schema graph. The nodes of this schema graph correspond to the element types and the atomic data-types. Attributes are modeled by edges labeled with attribute names. The content model of an element type is relaxed to a set of children types that can be reached via special *#(_)* edges. While the extension of the parser in the HyperView System to XML is sufficient to load XML schemata, a dedicated parser needs to be added in order to input DTDs.

The conversion from a XML schema document to a HyperView schema graph can be achieved easily in the HyperView System using a small set of transformation rules. We present two such rules to give a flavor of how this task can be specified quite naturally using HyperView. The rule in Figure 9.4 matches an attribute declaration in an element type declaration of a XML schema represented as graph and adds a new attribute to the corresponding schema graph. In Figure 9.5, an element type reference in the content model of an element declaration is matched and added as a new child node to the corresponding schema node. In both cases, the reuse specifications

```
<schema>
  <elementType name="book">
    <sequence>
      <sequence minOccur="1" maxOccur="*">
        <elementTypeRef name="author"/>
      </sequence>
      <elementTypeRef name="title"/>
      <elementTypeRef name="publisher"/>
    </sequence>
    <attrDecl name="year">
      <datatypeRef name="integer"/>
    </attrDecl>
    <attrDecl name="isbn">
      <datatypeRef name="ISBN"/>
    </attrDecl>
  </elementType>
  <elementType name="author">
    <datatypeRef name="PCDATA"/>
  </elementType>
  <elementType name="title">
    <datatypeRef name="PCDATA"/>
  </elementType>
  <elementType name="publisher">
    <datatypeRef name="PCDATA"/>
  </elementType>
</schema>
```

Figure 9.3: Example: the book XML schema

expressed by the dotted loops in the rule head ensure that element and data types are represented by unique schema nodes.

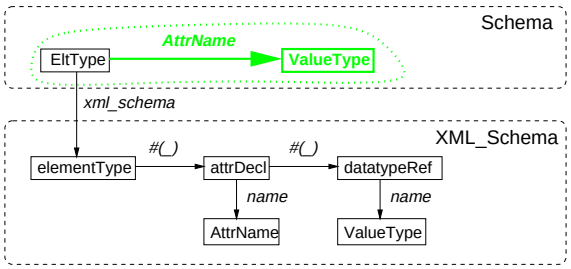

Figure 9.4: Rule for extracting attribute definitions from XML schemata. The graphical notation is explained in Table 2.1 on page 18.

9.2.4 XPointer and XQL

The XPointer standardization effort [W3C, 1999c] of the W3C aims to define a language that addresses points and regions of a XML document. The goal of this language is to go far beyond the expressivity of links in HTML. Whereas HTML links can point only to a whole document or to a named anchor inside of a document, XPointer allows links to specify document parts identified by navigations on the document structure and content. A document part addressed by XPointer references do not have to be intended as link target by the author of the document.

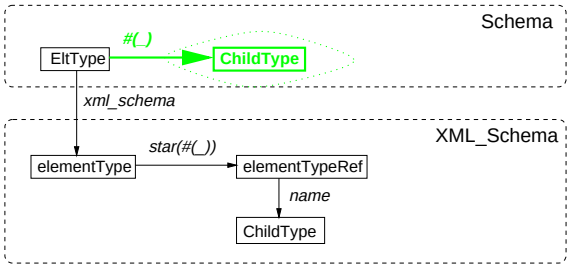

Figure 9.5: Rule for extracting children elements from content models in XML schemata.

The HVQL query language of HyperView is sufficient to implement the navigations specified by XPointer links.

While the XPointer effort navigation aims at identifying document parts, the XQL proposal [Robie *et al.*, 1998] for a XML query language has the goal of supporting data extraction from XML documents. This language provides only a subset of the navigation functionality of HVQL: it does not support restructuring, joins, and aggregation. In Table 9.2 some XQL queries and their HVQL equivalents are presented. As we can see from these examples, XQL is more concise than HVQL, but at the price of lesser expressiveness.

XQL query	**HVQL query**	**Description**
`book/title`	`_:book->` `#(_)=Title:title`	All `title` elements being children of `book` elements
`book/author[0]`	`_:book->` `#(1)=Author:author`	The first `author` element of each book

`book[@year=1999]/title`	`_:book->`
	`[year=1999]->`
	`#(_)=Title:title`
		All titles of books that have appeared in 1999

`book[author!text()=`
`'M. Smith'`
	`_:book->`
	`[author->text='M. Smith']->`
	`#(_)=Title:title`
		All titles of books by 'M. Smith'

Table 9.2: Example XQL queries and their HVQL translation

9.2.5 Extensible Stylesheet Language

The *Extensible Stylesheet Language* (XSL) effort aims at defining a language for the transformation and formatting of XML documents. While the XSL specification [W3C, 1999e] mainly deals with formatting, the accompanying specification [W3C, 1999f] defines the template-based transformation mechanism of XSL.

A XSL stylesheet defines a transformation of an input document into an output document. It essentially consists of a set of templates. Each template specifies a pattern that has to be matched in the input document and an XML fragment that is to be instantiated by the match, recursively expanded using other templates, and then inserted into the output document.

Patterns are specified using a language that extends XQL with further navigation capabilities, such as ancestors in the document hierarchy, preceding or succeeding nodes in text order etc. These navigation modes are available as computed edges in the HyperView System as well.

The values inserted into a template upon its instantiation are determined by embedded queries that identify sub-documents relative to the current match. It can be specified that the inserted sub-documents should further be transformed recursively.

A document is processed by a traversal of the document tree where each node is replaced by the instantiation of the most specific template that matches the node.

Formally, XSL templates can be seen as tree rewrite rules that could as well be implemented in HVQL. Compared to HyperView, the XSL transformation language is lacking a formal model, an incremental activation mechanism and the ability to deal with multiple input documents. It is neither appropriate nor intended for information integration tasks, as sometimes speculated.

9.2.6 Channel Definition Format

The CDF "*Channel Definition Format*" [Mic, 1997] supports the alerting of users about documents available at a Web site. A CDF file (also called "channel") essentially lists a set of documents that are new or have been changed recently. The CDF client polls the CDF channel at the Web site on a regular basis and notifies the user or automatically displays the new documents if changes in the channel have occurred. This technology makes most sense when personalized CDF files are generated dynamically according to user profiles.

The HyperView System can take advantage from CDF channels since CDF is a XML format with a rather simple structure. This makes it possible to load and interpret CDF files easily. One possible application of this functionality would be in writing wrappers for EJournal Web sites (cf. Chapter 2) that offer a CDF channel for announcing new journal issues. Instead of traversing a number of journal home pages, the HyperView wrapper would load the channel file and check for those journals that have new issues. This would reduce the communication load considerably, in particular for large publishers.

Moreover, HyperView can also be used as a generator for personalized CDF files since XML documents can be produced using the mechanism for dynamic HTML pages described in Chapter 6.

9.2.7 Resource Description Framework (RDF)

RDF [W3C, 1999a] aims at providing a generic framework for expressing metadata on resources in the WWW. Roughly speaking, RDF amounts to propositions over binary predicates. This means that a RDF model can be represented by a labeled, directed graph whose nodes are Web resources (e.g. document URLs) or literals, and whose edges are labeled with predicate names. For instance, an edge labeled "author" from an URL to a string "M. Smith" encodes the statement that a person called "M. Smith" is the author of the document at that particular URL.

RDF is an XML application. Different ways of encoding RDF models into XML are supported. Therefore, it does not make sense to work directly on RDF parse trees, but rather on a graph representation independent of the chosen XML encoding. This transformation can be implemented in HyperView in an intuitive way. For instance, Figure 9.6 sketches a rule that matches a RDF-description element in the tree-representation of an RDF document, creates a unique node representing the described resource and an edge labeled with the property name that points to the property value.

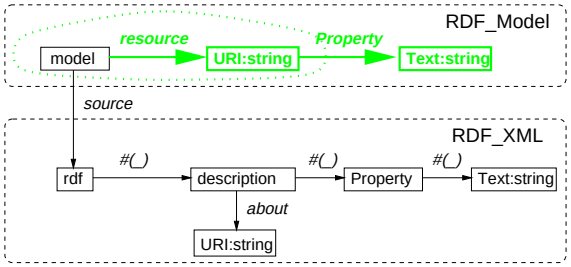

Figure 9.6: Rule for converting a RDF description into a labeled edge in a RDF model.

Besides statements on external documents, RDF supports also statements about other RDF statements. Such a higher-order statement could for instance specify the author of a bibliographic record on a certain document in the Web. To be the object of another statement, a statement must be "*reified*", i.e., it must be a resource of its own. A reified statement is modeled differently from a regular RDF statement. This means that regular and reified statements have to be queried differently. The HyperView System can be used to select and convert reified statements on demand into regular statements. Figure 9.7 shows a rule for this task.

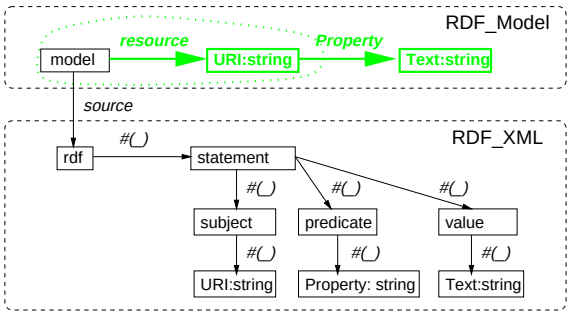

Figure 9.7: Rule for converting a reified RDF statement into a labeled edge in a RDF model.

Note, that for a realistic implementation, several rules will be necessary, since RDF allows several alternative XML encodings of the same RDF model.

The generic database browser of the HyperView System can be used to view RDF models and access the described resources via hyper-links.

9.2.8 RDF Schemata

RDF is a generic framework for defining metadata standards. Vocabularies for concrete metadata standards can be defined using RDF schemata [W3C, 1999b]. Examples of such standards are the Dublin Core, PICS, MARC etc. RDF schemata are similar to object-oriented or conceptual schemata. They support multiple inheritance and specialization of attributes. RDF schemata describe classes of conforming RDF models and are themselves denoted by RDF models. Such schema models conform to the RDF meta-schema. The exact *semantics* of RDF schemata and the conformance of RDF models to schemata are left unspecified in [W3C, 1999b]. However, a RDF schema does not enforce the existence of certain data elements, but rather specifies data elements that may occur in a RDF model. In this sense, the RDF notion of conformance follows the philosophy of semistructured data.

RDF schemata can be transformed into HyperView schemata in a way similar to XML schemata. Since HyperView does not support generalization hierarchies, the translation must add inherited attributes explicitly to the inheriting classes, though. Generalization of attributes can be implemented by adding rules for virtual edges to the HyperView schema. For instance, the *father* attribute of a person can be seen as a specialization of its *parent* attribute. Hence, a rule can be inserted that computes a missing *parent* value from the *father* value. To capture the object-oriented features of RDF schemata more *directly*, an object-oriented extension of the HyperView schema concept has to be considered. Another interesting application would be to use the RDF schema format as an exchange format for HyperView schemata.

9.2.9 Summary

The switch to XML documents on the Web will alleviate, but not remove the data extraction problem. Although emerging standard formats will reduce the costs for data integration within narrow application domains, there will always be need for the integration of data from structurally

heterogeneous documents. The HyperView approach is highly appropriate for this task.

At the same time, XML extensions such as XPointer, query languages like XQL and emerging XML applications such as RDF create ample need for query and transformation facilities for semistructured documents. HyperView can be used in a straight-forward way to serve these needs.

A third application of HyperView in the context of XML will be in creating XML documents from "legacy" HTML pages. The HyperView System provides facilities for dynamic generation of HTML pages that can be used without changes for creating XML documents as well.

9.3 Open Issues

9.3.1 Theoretical Issues

The theoretic framework presented in Chapter 3 formalizes the view mechanism of HyperView. The extensions of this view mechanism that are presented in this section can be undertaken as future work.

Regular path expressions are a powerful mechanism for supporting robust data extraction from semistructured documents that are represented as graphs. In the HyperView System regular path expressions are implemented as builtin meta-edges in HVQL, but without theoretic foundation. Hence it is an interesting question how can they be supported in the theoretic framework. One way to implement regular path expressions is to use recursive rules. However, this requires to formally define the semantics of recursive rules in the formal framework. Alternatively one can consider instead of a graph its closure under regular path expressions. This means that all graphs are viewed as being equipped with all edges that correspond to arbitrary regular path expressions. For instance, the transitive closure of edges labeled by some label a is represented by edges labeled by $a*$.

In order to identify data items from different sources or from different locations within the same source with each other, it is important to support a mechanism similar to keys in relational databases. For this purpose, HyperView uses rule annotations called **reuse specifications**. This mechanism is very flexible, but on the other hand it requires a certain coding discipline. Hence it seems worthwhile to investigate as an alternative mechanism the automatic generation of rule annotations from schema annotations. Such a mechanism would be analogous to classical databases where keys are specified as part of the schema definition rather than within update statements.

Often it is helpful to define rules that apply to a group of vertex types, with the exception of a few vertex types where special rules apply. For instance, the descendents of a document element can be defined generically as its children and the descendents of its children. However, for headlines in HTML such as the <h1>,.., <h6> tags it makes more sense to define as descendents also all document elements that are textually right of the headline, but left of the next headline of the same level. A pragmatic approach taken in this work is to copy rules for all possible target types. Since this is needed only for a finite number of vertex types, this approach is feasible. However, cases like this motivate concepts for **inheritance** and **parameterized types** to be introduced in the formal framework.

9.3.2 Integration Issues

As pointed out in Chapter 7, there are classical problems of data integration where HyperView relies on existing work. We briefly discuss these problems and appropriate approaches here.

In many applications, different records describing the same entity may differ due to different naming conventions, different spelling, incomplete information etc. While HyperView provides the concept of reuse specifications to identify identical records, additional heuristics are needed for matching records. Work on **record linkage** in relational databases [Fed, 1997] can be adapted

also to semistructured data. Approaches for management and querying of incomplete information such as [Imieliński and Lipski, Jr., 1984] are also relevant here.

Entities in a database often have to be classified in some way. In the town information example described in Chapter 7, cultural projects are classified by their genre. The problem is how to achieve this classification. Sometimes all entities served by a certain source can be assigned to the same class due to some a-priori knowledge. Other sources provide a classification of their own. Unfortunately, the classes of different sources often do not match.

A pragmatic solution is to mirror all classes occurring in a source in the database. Changes in the classification scheme of a source will reflect immediately in the available classes in the database. However, there may be many synonymous classes from different sources.

Another approach is to establish a fixed classification hierarchy in the database and mappings from the source-specific classes into this hierarchy. However, changes in the classification scheme of a source will break these mappings.

Moreover, a fixed hierarchy may lead either to an information loss or misclassifications if no exact correspondence between a source-specific class and a class in the database exists. This may require methods for automatic reclassification. Otherwise, misclassified instances have to be accepted.

This is a general data integration problem that goes well beyond the scope of this thesis. Probabilistic methods similar to those proposed for record linkage may help as well in fusing classes with similar names and insert subclass relationships to a number pre-established superclasses. Work on thesauri and ontologies applies as well.

9.3.3 Implementation and Performance Issues

The issue of **query optimization** becomes important also in the semistructured setting. For instance, there are queries for which the HyperView System accesses all underlying servers although a single server would be able to provide all the requested information. Classical query optimization techniques are not sufficient in the context of HyperView, due to its semistructured data model and because of the incremental view materialization which has to be taken into account. If a query is subsumed by an earlier query, the result of the new query can be obtained by restricting the materialized result of the earlier one. Hence methods for query optimization with materialized views [Chaudhuri *et al.*, 1995] and semantic caching [Dar *et al.*, 1996], [Keller and Basu, 1994] in relational databases might be adapted for the semistructured setting.

Since more and more data is materialized with each query against the HyperView System, the memory consumption of the HyperView System increases over time. A possible remedy against memory overflow is to restart the system. Since the HyperView System serves as cache for external data, a restart will not result in information loss. However, if the HyperView System is required to be permanently available, a **cache management strategy** least-recently-used is to be used in the graph database. This is not to be confused with garbage collection: the task is not to remove data items that are unreachable, but those who are less likely to be needed in the future. It has to be ensured that data is removed from the graph base only if it can be restored by applying the rules in the system again. Therefore the rule design must take this into account and the cache management strategy must be adapted appropriately.

The execution of queries which require a lot of external pages to be loaded can exceed the response time tolerated by a typical user. To speed up such queries, the HyperView System could be forced to materialize often used parts of its database by executing some general queries like "find out all available journals". Such a **cache warming strategy** will force the relevant pages to be loaded in advance. After having warmed the cache, the retrieval of a journal can be executed in-memory, without any external page accesses.

Since the Prolog machine on top of which the HyperView System is implemented does not support **multi-threading**, only one page request can be executed at a time. To serve multiple pages in concurrently, the HyperView servlet should be extended to support parallel incarnations of the HyperView server. The Java multi-threading facilities and the session support offered by the Java Servlet API make this feasible. Recently, an experimental multi-threading API for SWI

Prolog has become available (SWI 3.3.0). Hence an investigation of how this API can be used to implement a multi-threaded HyperView server seems worthwhile.

There is a large potential for **static optimizations** in the HVQL compiler. Schema information can be employed to resolve rule dispatch at compile time.

HyperView uses a simple Prolog-based HTML parser which can (with some restrictions) also process XML documents. However, it seems to be more efficient to incorporate an existing **HTML/XML parser** via the foreign library interface of Prolog.

9.3.4 Interface Issues

The style mechanism discussed in Chapter 6 may be used to offer the user at runtime the choice of different layouts and to customize the layout by changing style parameters. Moreover, the user may control the presentation of all nodes of the same type: by changing the schema annotations certain attributes can be ex- or included to / from the display. To implement such **customization features**, the browser has to be extended with a mechanism for editing graphs. This requires a mechanism for specifying access rights for graphs. Moreover, it must be ensured that changes of one user do not affect other users by storing changes for each user separately. Finally, users might want to keep there settings between sessions. This requires methods for user authentication and persistent storage of user profiles.

The URLs of pages generated by the HyperView System do not persist between different runs of the HyperView System. Since users like to bookmark relevant pages, it is important to assign URLs that associatively identify pages. Support for such **bookmarking** amounts to encode an HVQL query into the URL of each page such that the result of the query uniquely identifies the contents of the page.

Parameterized edges are presented by the HyperView database browser as forms with text fields. If the set of admissible parameter values is finite and it is known in advance or can be computed by a HVQL query, alternative input elements such as pop-up menus should be supported as a customization feature.

9.4 Contributions and Outlook

The main contributions of the HyperView approach are the following:

The distinguishing key idea of the HyperView approach is to apply the same view mechanism uniformly to solve the problems of extraction, integration, and presentation. This elegant concept results in a modular **layered architecture** that is composed from multiple instances of the same HyperView component.

The view mechanism employed in the architecture as well as the underlying data model are formally defined (see Chapter 3). The **formal framework** builds on the well-established theory of algebraic graph transformation. The main contributions of this formalism are the novel clustered graph data model CGDM and the formal semantics of the demand-driven view materialization mechanism. This theoretic foundation distinguishes the HyperView approach from other more pragmatic approaches. The formal framework has been published in [Faulstich, 1998] and [Faulstich, 1999b].

The **implementation** of the concepts defined by the formal framework is presented in Chapter 4. The HyperView System serves as a platform for defining views and building virtual Web sites. It consists mainly of approx. 9000 lines of Prolog code. The HyperView distribution includes all external tools and runs on most Unix platforms. The transfer of the formal concepts defined in Chapter 3 into efficiently executable Prolog constructs and the implementation of the prototype constitute another main contribution of this thesis.

The **view definition language HVQL** described in Chapter 5 provides an intuitive syntax not only for defining views, but also for denoting data and schema graphs, and for formulating queries. Furthermore HVQL defines an interface between the development tool HyperDesigner

and the HyperView System. HVQL effectively decouples the view design from the details of the particular Prolog implementation in the prototype.

The **HyperView methodology** for modeling and integrating Web sites is described in Chapter 8. It provides a guideline for establishing virtual Web sites based on the HyperView System. It defines a set of user roles and describes the activities into which the developing process can be decomposed and the dependencies between these activities.

In two **case studies**, on electronic journals (see Chapter 2) and on cultural town information (see Chapter 7), the HyperView approach has been applied and virtual Web sites have been established. The wrappers for publisher Web sites developed in the first case study are used in the DARWIN project for automatic import of bibliographical data. This work is published in [Faulstich and Spiliopoulou, 1998] and in an extended version in [Faulstich and Spiliopoulou, 1999]. The case study on town information is published in [Faulstich, 1999a].

Compared to ad-hoc techniques, HyperView thus offers a straight-forward and efficient design method, a language with a formally defined declarative semantics, and an infrastructure for WWW access, robust information extraction from HTML pages, and flexible generation of HTML pages. All together HyperView provides all necessary means to build and operate virtual Web sites.

Outlook. The HyperView architecture allows the composition of independently developed views, thus fostering the exchange and reuse of view specifications. Communities of interested users or organizations may become important in this context. HyperView servers can be operated in the intranet of an organization, by a third party over the Internet, or locally by individual users or groups.

In the near future, XML formats will supersede HTML pages in the WWW. As pointed out in Section 9.2, "semantic" formats and domain-specific standards in XML will facilitate the integration of information sources using the HyperView approach. The more Web sites offer data in XML format, the more interesting become virtual Web sites since the costs for the integration of XML-based sources is expected to be lower.

However, the need for data integration from different sources will persist. Emerging domain-specific standard formats can reduce the costs of homogenizing data from the same application domain, but not that of relating and fusing data. Moreover, rules for the integration of data from different documents in standard formats can be reused to integrate Web sites that use the same standard formats. Finally there will be an increasing need to convert existing HTML-based sources into XML sources by wrapping them using HyperView or a similar technology. For all these reasons, the advent of XML will even increase the importance of the HyperView approach.

Regardless of XML, it can be expected that HTML will persist in the long run since many Web sites are targeted towards the end-user who does not need XML data, but in the first place Web pages in a presentation-oriented format like HTML. Hence many HTML sources will remain to which the HyperView approach can be applied in its full extent.

In addition to HTML and XML, there are many other semistructured data formats, particularly in legacy applications. The HyperView approach can be applied also to such formats by providing parsers that map these formats into the HyperView data model.

To sum up, the future will bring many opportunities for applying the HyperView approach in different constellations to sources with various document formats. The HyperView approach will help users to cope with the diversity and heterogeneity of the Web by accessing multiple Web sites through virtual Web sites that present information tailored to the users' interests.

9.5 Acknowledgments

Many people have contributed to the success of this thesis.

First of all, I would like to express my thanks to Prof. Dr. Heinz Schweppe who has invited me to join his group and the Graduate School on Distributed Information Systems, who encouraged

me in freely selecting and pursuing my topic and supported me with helpful advice, constructive comments and regular consultations.

Prof. Dr. Herbert Weber (TU Berlin) invited me as well to the CIS group and the Graduate School on Distributed Information Systems. Although my topic brought me closer to the group of Prof. Schweppe, there have been strong bounds to the CIS group all the time and I'm grateful to him for reviewing this thesis.

Prof. Dr. Hartmut Ehrig is leading the group "Theory of Formal Systems" at the TU Berlin. He is a well-known expert in the theory of graph transformation techniques. I am glad that he has accepted to review my thesis and in particular its theoretical part.

Dr. Ralf-Detlef Kutsche from the CIS group at the TU Berlin has accompanied and advised my research with constructive and fruitful comments on the evolving versions of this thesis. In particular, he has stressed the software engineering aspects and stimulated the development of the methodology for deploying and using the HyperView System. Moreover, he introduced me to Dr. Gabriele Taentzer from the group of Professor Ehrig, TU Berlin.

Dr. Gabriele Taentzer supported me in formalizing my ideas of a rule-based view mechanism on graphs with helpful discussions, bibliographic hints, and by closely reading and commenting in detail on an earlier version of this thesis. She also encouraged me to contribute to the graph transformation community which resulted in [Faulstich, 1998, Faulstich, 1999b].

I would also like to mention the two diploma theses in the context of HyperView that have been written in cooperation with CIS: I am grateful to Ahter and Mürüvet Öksüz (TU) for having contributed valuable tools that will foster the application of the HyperView System and to Prof. Weber and Dr. Kutsche for making this cooperation possible.

I would like to express my thanks also to all members of the Berlin-Brandenburg Graduate School on Distributed Information Systems, to its speaker Prof. Oliver Günther, Ph.D. (HU Berlin), all other professors and researchers, and all collegiates for creating such an exciting, stimulating and supportive environment. The regular presentations and constructive discussions were crucial for the progress of my thesis.

My dissertation has resulted in several publications, which have been refereed anonymously. I am indebted to the reviewers, whose comments and suggestions helped not only on the improvement of my papers but also on the consolidation of the framework of my work.

Research does not only need an exciting intellectual environment to be successful, but also financial and material support. In the first place I want to thank the German Research Society (DFG grant no. GRK 316) for the PhD grant that allowed me for three years to concentrate fully on my research without teaching or administrative obligations. Moreover, the German Research Society financed my travels to several conferences giving me the opportunity to present my work. When speaking of travel grants, I have to mention as well the TMR program of the European Union which supported in 1999 my participation in the Dagstuhl Seminar on Systems Integration.

For material support in form of working space and computing environment I am first of all indebted to Professor Schweppe and the Institute of Computer Science, Free University of Berlin, but also to Professor Günther and the Institute of Information Systems, Humboldt University Berlin, and to Professor Theoharis and the Department of Informatics, University of Athens.

Last but not least I want to deeply thank my wife and collaborator Dr. Myra Spiliopoulou for both her emotional and scientific support, for numerous fruitful discussions, for her contributions to various publications (e.g. [Faulstich *et al.*, 1997, Faulstich and Spiliopoulou, 1998, Faulstich and Spiliopoulou, 1999]), and for reading and commenting on various versions and parts of this thesis.