Chapter 1

Introduction

1.1 Integration of semistructured information sources

In order to answer a specific question using the WWW, users often have to collect and combine information from multiple pages at different Web sites. Due to the autonomy of information providers in the Internet, Web sites show a high degree of incoherence and structural heterogeneity. This causes several problems to the Web user:

The user has to learn the organization and layout of each Web site in order to find the relevant pages and interpret them correctly. Even though this *structural heterogeneity* is not a major problem for a human user, it still may cause confusion and unnecessary delays.

Most often, there are *missing links* between closely related information on different Web sites. For instance, institutions or persons are often mentioned without providing links to their respective home pages. Instead of directly following links, the user thus has to search for the desired pages.

Several Web sites may serve similar information on a certain topic, but often no single site reaches a sufficient degree of completeness. Due to this *incomplete coverage*, the user may have to search several Web sites.

Data from one provider often cannot be used to retrieve and/or match data from another provider due to *semantic heterogeneity* which manifests itself in different terminology, notation, or data granularity.

There is a *fluctuation* of new Web sites that appear and existing ones that cease to be maintained or become unavailable. Sources may *change* their organization, layout, access methods, and contents without notice. Such external changes often cause *dangling links* and may also outdate any other information copied from external Web sites.

Virtual Web Sites One promising approach to facilitate the combination of information from multiple Web sites is to build "*Virtual Web Sites*" that integrate existing Web sites. A virtual Web site is a service that *combines* information from a number of Web sites and serves it in a *homogenized* form in order to save the user from tediously searching for and browsing numerous pages from these Web sites.

We call such a service "*virtual*" since it gives the impression of a single, homogeneous Web site by hiding the heterogeneity and possibly even the existence of the underlying Web sites from the user. The *main promises* of virtual Web sites are to resolve the problems of *structural heterogeneity*, *missing links*, *incomplete coverage*, and to some extent also of *semantic heterogeneity* and of *structural changes*. The problem of *source updates* can be solved by including notification services into a virtual Web site. But already the fact that virtual Web sites simplify the search for information makes it easier for the user to repeat searches and thus stay up-to-date.

According to the definition above, a search machine could be viewed as a virtual Web site as well, since it offers homogenized information extracted from other Web sites. However, we focus here on *semantic* integration of Web sites. This means that the virtual Web site reflects concepts of a specific *application domain* rather than serving generic metadata extracted from Web pages.

There is one search machine that indeed is a simple, yet classical example of a virtual Web site: the MetaCrawler[1] extracts and combines data from other search machines and presents all returned references in a uniform format. Thus the application domain is "references for Web documents".

However, MetaCrawler may return multiple references from different search engines that point to the same document. True semantic integration goes a step further requiring that data from different sources has to be combined and presented as a single entity if it represents the same entity.

To achieve such semantic integration of data, first the meaning of this data has to be understood and then a transformation homogenizing the data has to be found that restructures the data according to a conceptual model of the application domain. This clearly requires intelligence and hence cannot be automated successfully.

[1]<www.metacrawler.com>

However, there are many Web sites that show a sufficiently regular structure to allow predictions on the structure and semantic content of their pages based on the analysis of some exemplary pages. We call such Web sites *semistructured sources* since – even though they lack *explicit* schemata and strict typing – they still loosely follow an *implicit* structure up to some variations and irregularities in the data (cf. [Abiteboul, 1997]).

Thus, the semantic integration of semistructured Web sites depends on *a-priori knowledge* about their implicit structure and the meaning of this structure. For documents in layout-oriented formats such as HTML, discovering this structure requires to find syntactic regularities in the documents, to identify variable data elements among static layout elements, and finally to deduce a hypothetical conceptual structure of the data that explains the inspected pages. This task can be seen as a reverse-engineering process. The result of this analysis must be an executable specification that can effectively extract the data at run-time and reorganize it according to the derived conceptual structure. The advent of application-specific XML formats in the Web will alleviate the problem of discovering the conceptual structure of the data and of finding the data in documents. However, facilities for querying of documents and data extraction are necessary as well. Moreover, the problem of homogenizing and combining data from heterogeneous documents will stay. Both tasks require solutions very similar to those for the HTML case.

The draw-back of any integration approach based on a-priori knowledge about the content and structure of Web sites is that this knowledge is site-specific and has to be provided and maintained by domain experts. Therefore it is crucial to reduce the costs for setting up such an integration of Web sites. The HyperView approach aims at reducing these costs by providing a design approach, a declarative view definition language (HVQL), and a software infrastructure (the HyperView System) for integrating web sites. This work serves as a basis for the interactive view design tools currently under development ([Öksüz, 1999a, Öksüz, 1999b]). These tools will further reduce the costs for building and maintaining a virtual HyperView Web site.

1.2 The HyperView approach

To solve the problem of integration of semistructured data sources as outlined above, three subproblems have to solved:

Extraction: relevant data has to be extracted from the sources. For sources being Web sites, this means loading HTML pages, finding the relevant pieces of information on these pages, and identifying the relations between them. The requirement that sources are well-known and sufficiently regular allows to define the extraction methods in advance.

Integration: data from different sources and from different locations within a source has to be combined and restructured into a uniform semantic representation. This step relies on the existence of a global schema of reasonable size due to a narrow application domain.

Presentation: the user interface supports the user in formulating and submitting information requests and presents responses consisting of parts of the integrated data in a certain target format. The primary target format is dynamically generated HTML, because it allows to set up a Web site which the user can browse using a standard HTTP client to access the integrated data.

These steps are reflected in the HyperView methodology and architecture. HyperView offers an unifying formal model that applies to all three of these steps and a generic system based on this formalism that implements virtual Web sites defined by declarative specifications for extraction, integration, and presentation.

1.2.1 Data Model and View Mechanism

The three steps introduced above can be conceived as consecutive mappings. Each of this mappings can be implemented as a view on its input data that provides the input data for the next step.

To reduce the complexity of the system, a uniform data model and view mechanism is to be used throughout all steps. A graph-based data model has been chosen since graphs have been found suitable for representing semistructured data [Abiteboul, 1997],[Quass *et al.*, 1996], [Buneman *et al.*, 1996],[Fernandez *et al.*, 1997]. Thus directed labeled graphs are used uniformly to represent the input data (HTML pages), the data extracted from these pages, the integrated data, and finally the data returned to the user. Graphs are described by graph schemata which are again graphs. Since we have to handle several graphs at a time, we have developed the *Clustered Graph Data Model* (CGDM). In this model, each graph in the system is viewed as a so-called *cluster* of a global modularized graph. Using this data model, the WWW is represented conceptually as a clustered graph where each HTML page forms a cluster of its own. Hyperlinks between different are represented by inter-cluster edges.

The view mechanism needed in HyperView has to support mappings between graphs. To be more precise, it must transform of one or several input graph clusters into an output graph cluster. Since space and time limitations as well as source updates make it difficult to extract and integrate all available information in advance, the view mechanism must support incremental materialization in response to user requests. Graph transformation techniques have been intensively studied in the field of graph grammars (cf. [G. Rozenberg Montanari *et al.*, 1997]). However, the requirements of modularization and of demand driven execution are not met by the existing approaches. Therefore we have developed a special kind of graph-transformation rules by which views on graphs can be materialized using demand-driven activation strategy for rules.

To support the composition of views, rule activation propagates through several layers of views in a backward-chaining fashion. By this, queries expressed as requests for graph elements are propagated to the bottom layer. In the HyperView System where the bottom layer models the WWW, this layer is materialized on demand by loading the requested pages and converting them into new graph clusters. The data added to each layer allows the rules of the succeeding view to be applied and to extend the next layer. Thus, a cascade of rule activations finally propagates the requested information back to the top layer.

This novel view mechanism of HyperView forms a key contribution of this work. The formal treatment of the *Clustered Graph Data Model* (CGDM) and of the view mechanism is presented in Chapter 3.

1.2.2 Architecture

In order to separate the steps of information extraction, integration, and presentation, we use a hierarchy of views which map information from a lower to a higher level of abstraction. These levels of abstraction are reflected by the layers of the conceptual HyperView architecture depicted in Figure 1.1.

The HTML layer models single HTML pages loaded from the Web sites at a purely syntactical level. The *Abstract Context Representation* (ACR) layer represents the data extracted from each Web site. This data is described by a schema, the *ACR schema* of the site. It models the relevant content of the Web site at a semantic level, but organized according to the Web site's structure. The ACR schemata are analogous to the component *export* schemata in the well-known reference architecture of [Sheth and Larson, 1990] for federated database management systems. Note, that ACR schemata do not correspond to the *internal* component schemata since they capture only the data relevant for the integration.

Information extracted from different Web sites is combined at the database (DB) layer which provides an integrated domain-oriented view on top of the ACR layer. The DB layer is described by a schema called the *DB schema*. This schema corresponds to the *federated schema* in [Sheth and Larson, 1990].

Finally, the user interface (UI) layer consists of a set of views on the database layer which are then formatted as HTML pages and delivered to the user's HTTP client. Each of these views is described by a corresponding UI schema. In [Sheth and Larson, 1990], these schemata are analogous to *external schemata*.

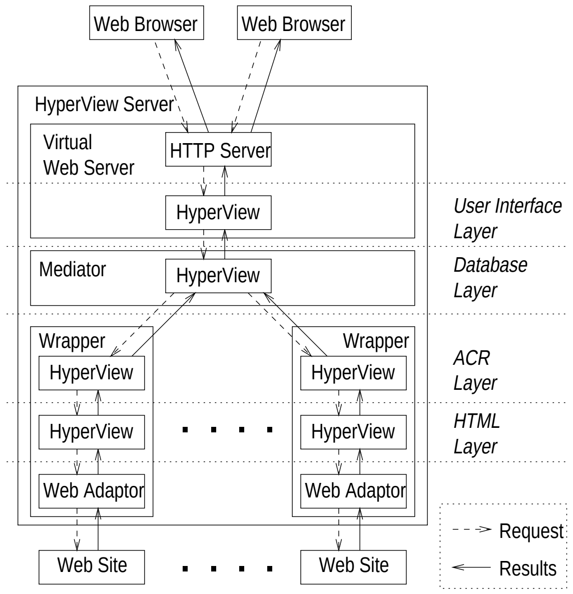

Figure 1.1: The HyperView architecture.

Besides virtual Web sites, all other conceivable kinds of applications that rely on integrated information from Web sites can be built on top of the database layer. One example of such an application in the field of digital libraries is presented in Chapter 2.

From the viewpoint of systems design, the layered conceptual architecture of HyperView fits well in a wrapper-mediator architecture [Wiederhold, 1992]. Each Web site can be encapsulated into a site-specific wrapper which implements the HTML and the ACR layer. Information from different wrappers is integrated in a mediator which implements the DB layer. The UI layer is implemented by the user interface which can be seen as a specialized kind of mediator.

The basic building block of the HyperView architecture is the module depicted in Figure 1.2. Each of these modules stores a virtual graph that is computed as a view of the virtual graphs in the underlying modules. Since this view mechanism is the core functionality within the HyperView architecture, these modules are called HyperView modules. Each HyperView module contains a processor that receives HVQL queries against the graph base of the module. This graph stores the already materialized part of the view. If the query cannot be answered from the current contents of the graph base, the processor uses view definitions stored in a rule base to query other HyperView modules and to materialize the needed graph elements on the fly.

The bottom most HyperView modules are called *Web Adaptors* since they do not issue HVQL queries, but load pages from the underlying Web sites in order to answer the queries they have received.

The modular HyperView architecture lends itself for a distributed implementation even though the current prototype runs on a single machine.

1.2.3 Application of the HyperView Technology

There are several application scenarios for the HyperView technology: the HyperView System can be used to integrate competing sources and / or to join complementary sources. Moreover it can also employed to wrap and transform single legacy sources.

As mentioned, the main application of HyperView is to build virtual Web sites. To the human user, a virtual Web site does not look different from a conventional Web site. Hence, the usage of virtual Web sites will be quite similar as well.

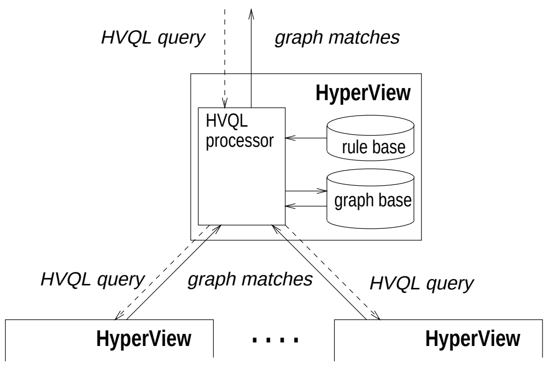

Figure 1.2: The HyperView module: basic building block of the HyperView architecture.

Instead for building virtual Web sites that serve pages to *human users*, the HyperView System can be used also to feed other existing information systems, such as conventional database systems. This is for instance the case in the DARWIN project where a bibliographic database is filled with data gathered by the HyperView System. Alternatively, the HyperView System can also provide input to other Web-based applications by serving XML documents in application-specific formats.

The view specifications necessary for setting up a HyperView System can be developed by single parties as proprietary code or on an open source basis by (groups of) interested individuals within the Internet. The view development requires both expertise in the domain of the information to be integrated, and software engineering skills to write the view specifications. Hence we envision that domain experts will collaborate with software engineers in this task.

The installation and operation of the HyperView System does not require any particular system administration skills since it is similar to the installation and operation of a HTTP server. Hence the deployment of the HyperView technology is not limited by this factor. It is even conceivable that single users operate their personal HyperView server.

Typically however, a virtual Web site will be operated by some organization as a service to a larger user group. This user group could for instance be the intranet users within a company or a certain scientific community within the Internet.

1.3 Related Work (Overview)

This section gives a brief overview of the related work which is reviewed in detail in Section 9.1. For each of the steps of data extraction, integration, and presentation there exist various approaches:

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 and tend to be not very robust. Therefore we have adopted a graph representation based on HTML parse trees. Recently, this approach has been adopted by other projects like WebOQL [Arocena and Mendelzon, 1998] and W4F [Arnaud Sahuguet and Fabien Azavant, 1999].

Data integration. Integration of *semistructured* data is often tackled with **query-based approaches**. Web query languages like WebSQL [Mendelzon *et al.*, 1997] and W3QS [Konopnicki and Shmueli, 1997] aim at ad-hoc queries, not at a permanent integration of Web sites. 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 relational approaches to data integration 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.

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).

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

Completeness. 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. 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 complete, even though pragmatic solution. This has the disadvantage that different languages and data models are involved. In contrast to this, HyperView offers a homogeneous approach using a graph-based data model and graph-transformation techniques that are applied uniformly throughout all stages. This uniform approach is reflected in the HyperView 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.

1.4 Overview

The rest of this thesis is organized as follows: in Chapter 2, HyperView is introduced informally in a case study from the field of digital libraries.

Chapter 3 treats the formal framework that constitutes one main contribution of this work. It includes the graph data model and the view mechanism based on graph transformation techniques.

The next three chapters deal with the other main contribution, namely the implementation of this formal framework: In Chapter 4 the HyperView System prototype is discussed. This prototype implements the view mechanism of HyperView and provides infrastructure for accessing the underlying Web sites. The view definition language HVQL is presented in Chapter 5. HVQL defines a notation for the graph-transformation rules introduced in Chapter 3 and a translation into the internal representations used in the prototype. The generation of virtual Web sites on top of the database layer of the HyperView architecture is discussed in Chapter 6.

An extended case study in the field of town information is presented in Chapter 7. The methodology derived from both case studies is then introduced in Chapter 8.

Chapter 9 concludes the thesis. In Section 9.1 related work is reviewed. Future XML-based applications of the HyperView approach in the fields of metadata management and alerting are described in Section 9.2.