Chapter 8

The HyperView Methodology

This chapter describes the HyperView methodology and gives a guideline for building HyperView-based virtual Web sites over a collection of information sources in the Web. This process is related to methodologies for building federated DBMS (cf. [Sheth and Larson, 1990]) and schema integration (cf. [Batini *et al.*, 1986]). The underlying sources are autonomous and a global schema is needed that defines the representation into which data from the sources has to be integrated.

However, building a virtual Web site goes well beyond building a federated database since the underlying sources are not databases and lack schemata and query processing facilities. Moreover, the resulting virtual Web site again is more than a federated database since the contents of the database layer must be presented in form of a coherent set of dynamically generated HTML pages.

The main steps to be carried out are:

Content Specification: selection of sources and formulation of a presentation concept

Schema development: for each source, an ACR schema has to be developed. A DB schema defining a global conceptual model and a schema modeling the virtual Web site have to be specified as well.

View development: for each source, a HTML view into the ACR schema of this source and a ACR view into the DB schema have to be defined. The user interface has to be defined as a view over the global schema.

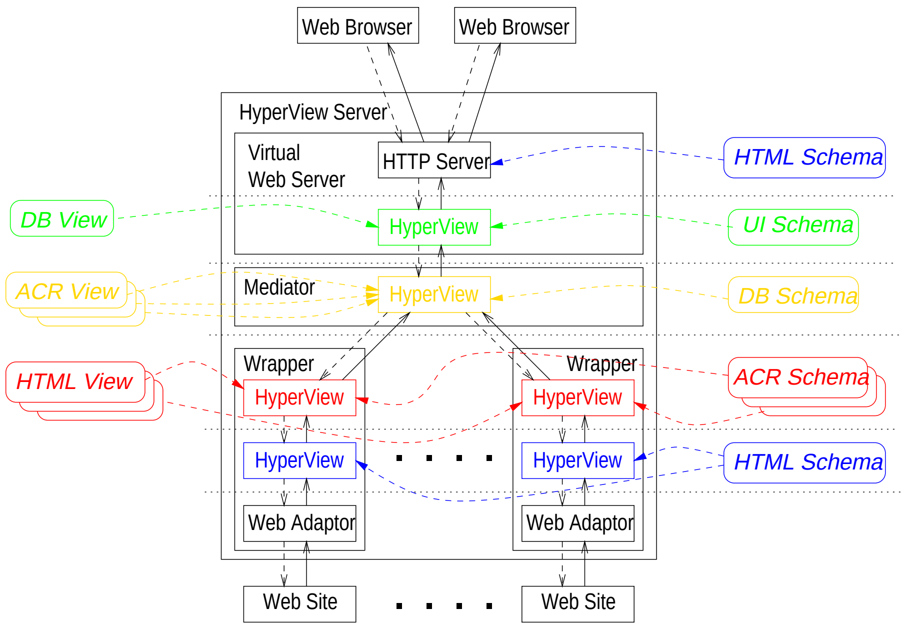

Figure 8.1: Schema and view definitions in the HyperView architecture

In Figure 8.1, the mentioned schema and view definitions are depicted together with the components of the HyperView architecture where these definitions are used. We can see that the HTML views are used in the wrappers to produce the ACR layer for each source. The ACR views for each source are used together at the DB layer to materialize this layer. Finally the UI view maps the DB layer to the UI layer and produces HTML clusters that are formatted and served to the user by the HTTP server component.

8.1 User roles

Before we discuss the details of the phases for setting up and operating a virtual Web site based on HyperView, we have to define more precisely the four user roles that are sketched in Section 1.2.3.

the *domain expert* selects the underlying Web sites sources and information assets therein. He decides which information is to be offered to the user as a result of integrating the underlying sources, and on the overall form (i.e., the presentation concept) in which this information is to be offered to the user.

the *designer* cooperates with the domain expert to refine his ideas and to implement the HyperViews needed to achieve the integration proposed by the domain expert. Due to the modular architecture of the HyperView system, several *designers* can work on wrappers for different sources, on the mediator, and on the Web interface concurrently.

The *designer* must be familiar with the HyperView methodology and must understand the semantics of graph schemata and the graph-transformation rules used to define HyperViews. In the current prototype, the *designer* must be able to code schemata and rules in the HVQL language described in Chapter 5. The forthcoming graphical user view definition interface [Öksüz, 1999a, Öksüz, 1999b] will replace this manual coding with interactive graphical editing.

the *administrator* operates the HyperView server. This involves mostly system administration like configuring the HTTP server, administration of the underlying machine, backup of local data etc. Since maintenance of schemata and HyperView definitions is carried out by the *designer*, this role requires a different and less specialized qualification profile than the designer role.

the *end-user* retrieves information offered by the HyperView server. The *end-user* can configure the presentation of the retrieved data by changing settings in his / her user profile. There may be several groups of end-users having different information demand, expertise, or access privileges. HyperView can provide different virtual Web sites to each of these groups.

These role descriptions are reflected in the use case diagrams in Figure 8.2 and Figure 8.3.

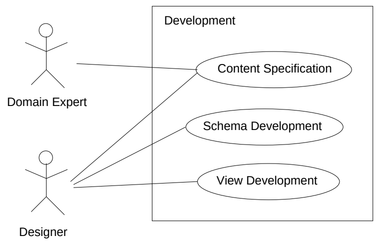

Figure 8.2: Use case diagram for the development of a virtual Web site.

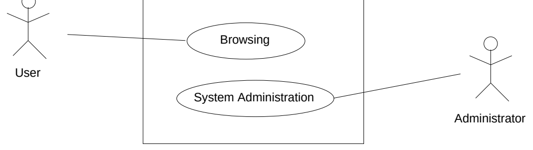

Figure 8.3: Use case diagram for the operation of a virtual Web site.

Depending on the tool support and expertise available, the domain expert and / or the administrator can act as designer. Moreover, it is conceivable that the user designs and maintains a HyperView server for his / her personal use. Normally however, a HyperView server will be operated by some organization as a service to some community. This community can be for instance a group of intranet users in a company or a certain scientific community within the Internet.

8.2 Content Specification

The content specification of a virtual Web site involves selection of sources and the decision which part of the source content is considered for the integration. On the other hand, it requires the formulation of an informal presentation concept that describes the content and the form of the virtual Web site. This step is similar to the *preintegration phase* of [Batini *et al.*, 1986]. The whole content specification process is summarized in Figure 8.4.

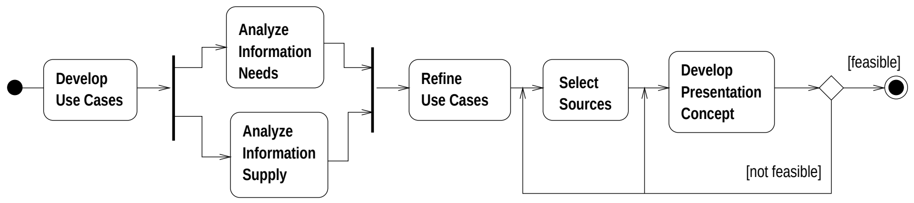
Develop Use Cases
Analyze Information Needs
Analyze Information Supply
Refine Use Cases
Select Sources
Develop Presentation Concept
[feasible]
[not feasible]

Figure 8.4: UML activity diagram describing the content specification process.

We assume that an application area for the virtual Web site has been chosen and a *domain expert* for this application area is available. The expert collaborates with the HyperView designer in configuring the virtual Web site.

First, the expert develops *Use Cases* (cf. Section 2.1.2, Section 7.2.1) that describe how users would like to use an ideal virtual Web site in the chosen application domain.

Then the expert analyses both the information supply and the information needs in the application area. The information supply is determined by inspecting existing relevant information sources. The information needs may be constrained in advance by preferences of a customer or may be determined from the experience of the expert or the potential users of the virtual Web site.

After the analysis of information supply and information needs both are compared by the expert in order to find those information needs that are not or only to a limited extent satisfied by the existing information sources. The outcome of this step is a refinement of the use cases from the first step that takes into account this comparative analysis. The refined use cases define goals for the outcome of the whole design process.

It is now the task of the expert to actually select those sources whose combination can yield a value-added information service that overcomes the identified deficiencies in the information supply and satisfies the requirements of the use cases. Before deciding for a source, copyright issues have to be resolved. In general this requires negotiations with the owners of a Web site. The result of this phase is a selected set of sources and a choice of information assets provided by these sources that will be considered for integration within the virtual Web site.

After selecting the sources, the expert has to collaborate with the designer to develop a *presentation concept* that specifies which content is to be presented in the virtual Web site and how its overall structure will look like. The content of the virtual Web site is decided by determining the main concepts that are to be modeled in the global schema and their connection to the information assets provided by the underlying sources. The form of the virtual Web site is specified by analyzing use cases and deciding on the access methods and criteria for information selection by the user that will be supported. The presentation concept may also define several phases of extension that allow to start with a subset of the selected sources and add new functionalities to the virtual Web site step by step.

During or after formulating the presentation concept, the designer has to assess whether the presentation concept is feasible on the basis of the HyperView System. For instance, Web sites relying too much on applets or Java Script may have to be ruled out. The designer must also take care to identify semantic incompatibilities that might prevent a successful integration. If such incompatibilities are recognized, the designer has to investigate the existence of algorithmic or external translation facilities. In the case of overlaps between sources, it must be decided whether one source is given precedence over the other or whether and how the contents of the overlapping sources are merged[1].

If the reached content specification is not satisfying, additional or different sources have to be considered or the presentation concept has to be modified.

8.3 The Design Space of HyperView

Before we discuss the phases of schema and view development in detail, we describe the design space, i.e., the conceptual space the designer has to deal with. This space is described by the HyperView design model depicted in Figure 8.5.

First of all, there is a *generic domain model* of the domain of Web sites. In the HyperView methodology Web sites are perceived as sets of pages and parts of pages which are generalized into the concept of WebNode. WebNodes can be linked in different ways: by hyper-links, by HTML-forms, and by the hierarchical part-of relationship between page parts. These associations are generalized into the concept of WebAssoc. The instances of the generic domain model are real-world Web sites, HTML pages, hyper-links etc.

Web sites are modeled by concepts from the *design object model*. The designer builds for each Web site an ACRSchema which is a Schema specialized to the ACR level. Schemata are graph clusters that describe the structure of the runtime graph clusters within the HyperView System. Schema clusters (like graph clusters in general) consist of vertices and edges. ACRSChemaVertex and ACRSchemaEdge are specializations of the concepts of Vertex and Edge. They model WebNodes and WebAssocs, respectively.

Design objects are defined by documents as specified by the *design document model*. Schemata are defined graphically by SchemaDiagrams and then translated into SchemaDefinition documents in HVQL format. Each schema definition references ViewDefinition documents that are created by combining a number of rules. Rules are specified graphically in form of RuleDiagrams.

8.4 Schema development

The source selection and the presentation concept derived in the content specification phase guide the development of all necessary schemas for the virtual Web site. The following schemata have to be developed: one ACR schema for each source Web site, the conceptual database (DB) schema, and the schema of the user interface (UI).

As already pointed out in Chapter 1, there are correspondences to the schemata in the reference architecture for federated database systems of [Sheth and Larson, 1990]: ACR schemata are similar to the *export schemata* of component databases, the DB schema corresponds to the *federated schema*, and the UI schema is similar to the *external schemata* offered to applications of the FDBMS.

However, there are major differences as well: while export schemata are defined by the administrators of the component databases, the ACR schemata have to be developed by analyzing the structure of a Web site without support by its administrator. An ACR schema does not model concepts of the application domain, but the *structure* of the underlying Web site. The UI schema does not describe the restricted view of an application on the global schema, but the structure of

[1]Work on linking of databases [Fed, 1997] may serve as a basis here.

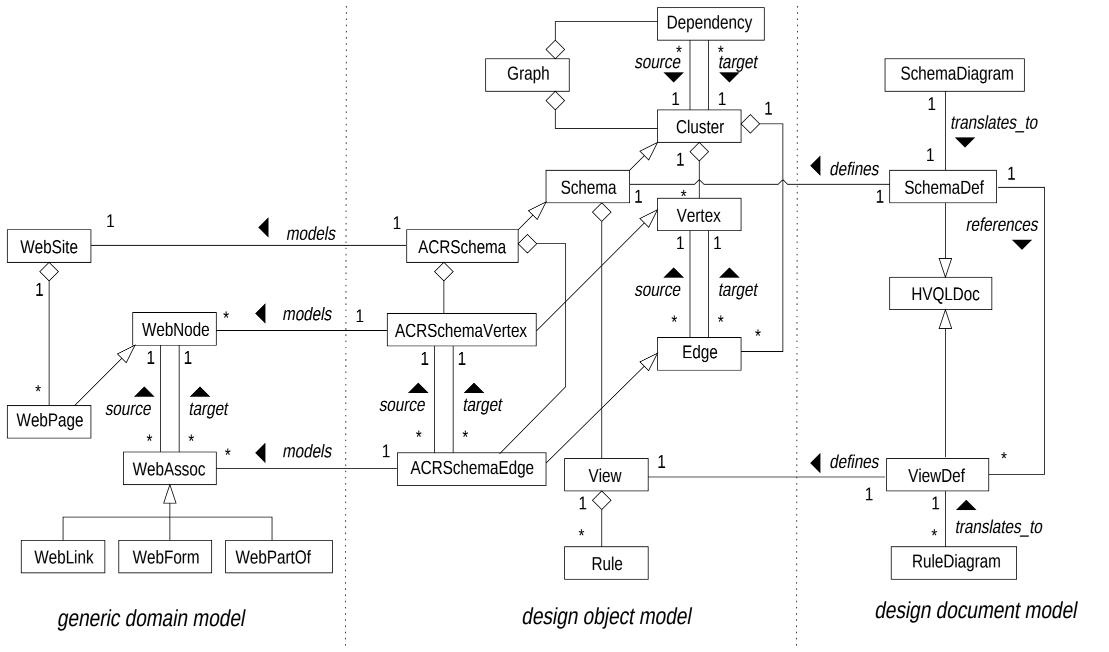

Figure 8.5: UML class diagram describing the HyperView design model: on the left, a generic model of the domain of Web sites is given. Web sites are modeled in HyperView by design objects described by the *design object model* (middle). These design objects are defined in documents as specified by the *design document model* (right).

a hyper-text that covers the whole global schema in a highly redundant way in order to facilitate navigation.

Since each layer of a HyperView-based virtual Web site is a view of the layer beneath it, the information capabilities of each layer must be sufficient to build the layer above it. Hence a bottom-up approach similar to the one presented in [Sheth and Larson, 1990] to develop the schemata at the different levels makes most sense. This development process must be guided by the presentation concept derived in the content specification phase in order not to waste efforts in modeling concepts that will not be included in contents of the virtual Web site. On the other hand, care has to be taken not to model concepts in a layer that are not supported by the layer below it. If it turns out that the presentation concept cannot be sufficiently supported by the resulting UI schema, it becomes necessary to go back and extend one or more schemata. Figure 8.6 illustrates the schema development process.

Whether the information capability of the schemata in a layer is indeed sufficient to support the whole schema of the next layer will be discovered in the view development phase when source and target schema are closely inspected in order to specify transformation rules between them. Hence it may become necessary to extend or revise schemata in order to remove obstacles for the definition of HyperViews.

We now discuss the particularities of the schema development in each of the different layers of the HyperView architecture.

8.4.1 HTML layer

HTML pages are described by a fixed generic schema. Hence source-specific HTML schemata do not have to be developed. The generic HTML schema includes several computed edges which provide logic relationships between page elements such as between a list and its list items, or

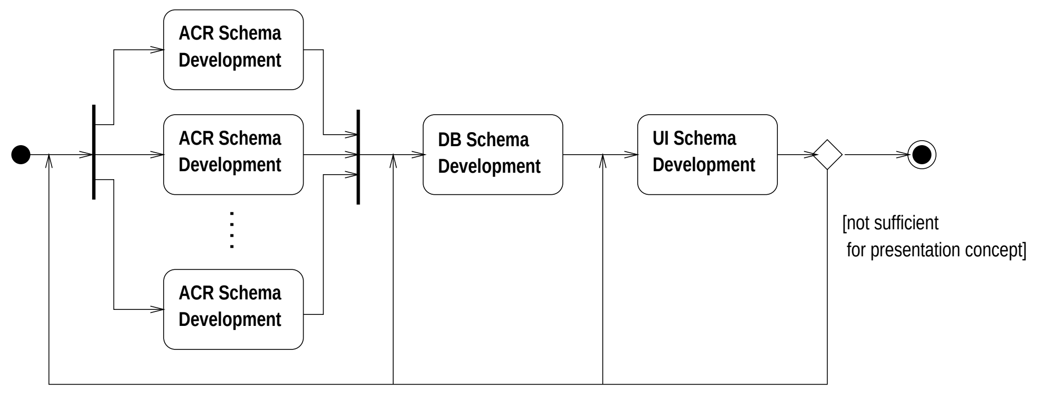

Figure 8.6: UML activity diagram describing the schema development.

between a headline and the following section. In Section 5.8 the available computed edges are described.

8.4.2 ACR layer

For each of the sources selected in the content specification phase an ACR schema has to be developed. This requires the designer to analyze the structure of the source Web site. This analysis proceeds according to the following steps:

1. Identification of pages and page fragments that that contain the information assets selected in the content specification process.
2. Discovery of navigation paths that lead from the server's home page or other fixed entry pages to the relevant information assets.
3. Abstraction of syntactically similar pages or page fragments into ACR vertex types and of the respective links or part-of relationships into ACR edge types. Only pages that contain relevant information assets or that are part of navigation paths have to be considered.
4. Construction of the ACR schema graph. Its vertices represent ACR vertex types and its edges represent ACR edge types connecting the vertex types.

8.4.3 Database layer

The conceptual (DB) schema models the application domain or more precisely the part of it that is to be covered by the virtual Web site. Concepts of the application domain are represented by vertices and the relations between these concepts by edges. In contrast to an ER-schema, no cardinality constraints are specified (i.e., all relations are $m : n$), and attributes are modeled as relationships. For stylistic reasons, nodes representing atomic data-types may be replicated. Thus each attribute points to a atomic type node of its own.

Similar to object-oriented schemata, edges represent unidirectional navigable relationships. This means in particular that reverse navigation paths have to be explicitly foreseen in the schema.

Although the DB schema development is driven by the presentation concept, this process is restricted by the requirement that the DB schema integrates the underlying ACR schemata. As already pointed out, an ACR schema describes only the structure of a Web site. However, domain concepts are usually reflected by this structure. Hence it is possible to adapt the relevant literature on schema integration (cf. [Batini *et al.*, 1986]) for this task. While the mapping between ACR and DB layer will be specified in detail later in the view development phase, it is important to find already at this stage the ACR concepts that model the information needed to instantiate a certain DB concept.

8.4.4 UI layer

The User Interface layer is specified by a schema similar to an ACR schema. This schema describes the abstract structure of the Web site generated by the HyperView System. This means that nodes of the UI schema model page (fragment) classes and edges model link classes.

Instead of developing a new UI schema, it is possible to reuse the existing generic UI schema of the HyperView browser discussed in Chapter 6 and to use the customization features of this database browser.

8.5 View development

After the schemata for a virtual Web site have been established, HyperViews that map between these schemata have to be defined. This process can start as soon as the source and target schema for a view are specified. Hence view development may as well take place in parallel to the schema definition phase. These dependencies are depicted in Figure 8.7.

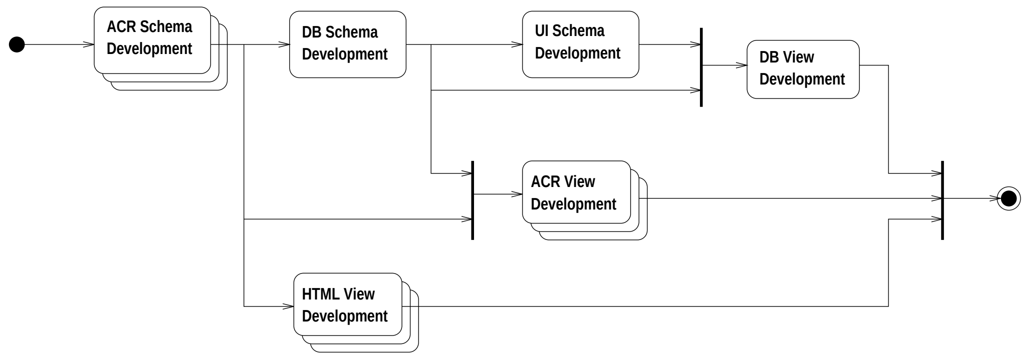

Figure 8.7: UML activity diagram describing the view development process and its dependencies from schema development activities.

Each HyperView consists of a set of rules which match elements in one or more input graph clusters in a layer of the HyperView architecture and materialize new elements in an output cluster at the layer above it. Typically, when a rule is a applied to a start node, it follows an edge to the input cluster, matches graph elements in this cluster, and adds a new outgoing edge from the start node, a new target node, and possibly (as a side effect) further graph elements to the output cluster. When the rule contains reuse specifications, existing elements of the output cluster may be reused instead. The rule usually adds again an edge pointing to the input layer that enables the application of further rules.

Thus, for every schema element at the upper layer, at least one rule has to be developed that can be used to materialize this element directly or as a side effect. Each activated rule may match multiple times and materialize for each match a new set of graph elements. Multiple rules implementing the same schema elements are allowed as well; all of them will be executed and may contribute new graph elements.

The implementation of a HyperView starts at the root node(s) of the output schema and continues along the paths of the schema. A rule for a schema edge can be tested once that all edges along a path from a root node have been implemented.

8.5.1 Implementing HTML views

HTML views define mappings between the graph representation of HTML pages and the ACR graph of a single source. To develop a HTML view, the designer inspects the source code of the pages corresponding to the ACR nodes, starting from the entry node(s) of the ACR schema.

The first goal of this inspection is to identify the parts of the HTML graph that correspond to instances of the ACR schema elements. The second goal is then to define for each edge of the ACR schema a navigation in the HTML graph that corresponds to the traversal of this edge. This navigation is a HVQL query as described in Chapter 5.

The challenge in defining these navigations is to generalize from a few example pages or page fragments to yield navigations that work reliably on *all* instances of a page (fragment) class even in the presence of small variations and structural evolution. The HyperDesigner tool [Öksüz, 1999a] currently under development will help in the task of generalization by discovering the common structure of a set of pages. The robustness of rules is an issue of programming discipline. This discipline tries to avoid order-dependent navigations, navigations relying on font style elements or graphical ornaments, or specifying the full text of labels to be matched. Sometimes however, the designer is forced to use such unreliable features due to the bad design of the underlying HTML code.

Basically, each of the navigations designed in the last step is now turned into a HVQL rule implementing the corresponding ACR schema edge. However, schema edges that serve as singleton attributes of an ACR node may be computed by the same rule that computes the ACR node.

ACR schema edges that correspond to hyper-links may use the computed *href_target* edge that dereferences the href attribute of an anchor (<a>) tag.

Often text nodes occurring in the HTML graph have a regular structure that corresponds to a small subschema of the ACR schema. For instance, a date string consists of components for year, month, and day that may be modeled by corresponding attributes of an ACR node. Hence, the text has to be parsed and the relevant data has to be extracted from it. The required parser can be implemented externally as a Prolog predicate that is called as a rule constraint from within a HVQL rule. Alternatively, there is a set of computed edges that allows to split text nodes into tokens that can be matched and extracted within a rule. A library of parsers for common formats such as dates, times or prices is provided by the HyperView System and can be used for both alternatives.

8.5.2 ACR Views

The DB layer is formed as an integrated view of the ACR layer that consists of an ACR cluster for each of the underlying sources. The purpose of an ACR view is to translate the source-dependent representation of a Web site in its ACR cluster into the source-independent representation of the DB cluster.

In contrast to HTML views where only a generic HTML schema is available, rules of ACR views can be specified against the ACR schema. Often there exist ACR schema nodes which correspond closely to nodes of the database schema. Other ACR schema nodes may have been introduced primarily for the purpose of modeling navigation paths in the underlying Web site and may provide only little information that is to be extracted. In such cases, the respective rules in the ACR views will use relatively simple navigations compared to rules for the HTML views.

In other cases however, there may be a larger mismatch between the concepts modeled in the database schema and the concepts offered by the ACR schemata. For instance, issues of electronic journals may be organized in the source by year and in the database by volume (cf. Figure 2.8 on page 19). ACR rules may even have to collect information for a single database node from several ACR nodes.

The task of an ACR view includes the translation of site specific notations, physical units or local conventions into the uniform representation used in the database graph. Often, this means to complement the data explicitly stored in the ACR with other data in order to provide a context.

This context may be implicitly or explicitly given. For instance, local telephone numbers may have to be completed to full telephone numbers by adding a fixed area code if it is known that the source is dedicated to a certain city. Alternatively, the knowledge for which city a telephone number is valid may be explicitly given in the source. Then this information can be used (together with some translation service) to add the correct area codes.

Further tasks of ACR views deal with resolving the problems of recognizing identical entities and classification of entities that were discussed in Chapter 7.

8.5.3 DB Views

Result pages generated by the HyperView Web Site are views over the database graph which map some part of the database to an output HTML graph. From this HTML graph, HTML code is printed and returned to the user requesting the page. Each DB view corresponds to a class of similar result pages which is represented by a node in the UI schema. Schema edges pointing to other page classes are implemented by hyper-links in the generated pages. Source edges are used to compose different page fragments to a whole page. How such views look like is described in detail in Chapter 6.

In the case that the generic HyperView browser is used as a Web interface, the DB view consists of the rules defining this browser. Instead of writing new rules, the browser is customized by choosing style parameters.

8.6 Maintenance

Virtual Web sites have to be actively maintained due to changed information needs on one hand, and source evolution and fluctuation on the other hand. Changed information needs of users can be countered by extending and or restructuring the virtual Web site. Extensions almost always require additional information assets to be integrated. Source fluctuation means the disappearance of existing sources or the appearance of new potential sources. Source evolution means changes in structure and / or content of a source.

In all cases, the design may have to be revised in order to adapt the virtual Web site to the changes. However, the HyperView approach shows an inherent robustness that minimizes the effort needed for this adaptation.

8.6.1 Robustness

First, HyperView uses rules rather than regular expressions to match syntax graphs of HTML pages. Independence from details of the HTML layout can be easier achieved in rules than in regular expressions. Differently from regular expressions, rules are insensitive to the ordering of elements on a HTML page and are thus not affected by textual reorderings and insertions of additional text elements. Furthermore, since rules can invoke other rules, a library of base, layout independent rules can be built and serve as a basis for the design. If details of the layout, such as ordering, are inevitable, they can be encapsulated into the rule library as well, and become transparent to the rules using the library.

Second, rule sets can be designed in a way that ensures maximal independence of the rules. This can be achieved by using separate rules to compute different edges emanating from the same 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.

Finally, the intermediate ACR layer to some extent decouples the HyperView defining the database graph from the HyperViews defining the ACR graphs. It allows many small structural changes to be handled by modifying the rule set for the ACR layer, without having to change the ACR schema. In this case the HyperView over the ACR layer is not affected.

8.6.2 Error detection

The HyperView system provides tracing of rule failures for debugging purposes. This allows to identify rules that have been broken by structural source evolution. The designer of a HyperView may annotate rules to identify whether a failure is to be considered an error. Rule failure messages can be collected in log files and then aggregated to determine the average behavior of rules. If failure rates increase significantly, the administrator is notified. Data mining techniques can also be applied to distinguish critical from unimportant failure patterns.

Moreover, a suite of test queries that is periodically executed can be employed to detect broken views by comparing their results. To this end, the queries must be designed in a way that they produce the same or monotonically increasing results over time.

For instance, the set of available issues of an electronic journal in a publisher Web site can be expected to increase over time. Similarly, the set of movie theaters in Berlin will be almost constant from week to week, and the number of announced concerts in a city will always be over some reasonable threshold. By comparing the current results with archived earlier results, failures can be detected.

In principle this technique can also be adapted to test virtual Web sites. Since queries cannot be issued directly but must be expressed as navigations in the virtual Web site, this requires to wrap this Web site. The HyperView System can be employed to wrap another HyperView-based virtual Web site and to monitor it by running the test suite against it and evaluating the results.

8.6.3 Adaption

Minor structural changes are thus tolerated by the inherent robustness of HyperView rules. Changes that are not tolerated can be detected as described in Section 8.6.2.

Structural changes in a source that break the existing HTML view require rules of this view to be reimplemented. Larger structural changes, in particular those that reflect new content types, require the ACR schema to be modified. This in turn may require rules of the ACR view to be fixed. If the ACR schema has been extended and the additional content is to be presented in the virtual Web site, then these extensions have to be propagated to the DB schema and the UI schema and appropriate rules implementing these extensions have to be added.

Sources that disappear entirely or new sources that are added to the virtual Web site result in the removal or addition of entire ACR schemata, HTML and ACR views for these sources.

Compared to wrappers written in an ad-hoc way using text matching, HyperView rules are easier to design and to maintain for the following reasons:

HyperView rules are on a much higher conceptual level. There is no need to care about the underlying HTTP protocol for loading pages and about syntactical details of HTML pages. Matching in graphs is more intuitive than the specification of complex regular expressions.

Rule sets are easily extensible. The scope of a HyperView is typically extended by adding new rules rather than modifying existing ones. For instance, it may be discovered that a class of pages is not homogeneous, but has some structural variants. Then alternative rules may be added which cover those variants for which the existing rules fail to work.

8.7 Summary

The HyperView methodology for building virtual Web sites foresees three tasks: content specification, schema development, and view development. The content specification phase precedes the other two and consists of selecting source Web sites and deciding about the content and overall structure of the virtual Web site.

The database schema has to be developed as a conceptual model of the application domain. For each source Web site, an ACR schema, a HTML view, and an ACR view has to be developed. Finally, the user interface layer is modeled by a schema and its content is defined by a HyperView on the database.

The amount of work to set up a virtual Web site can vary depending on the design of the underlying sources and the chosen presentation concept. According to our experience in the case study on electronic journals (see Chapter 2), it took about one person day to develop a HyperView wrapper for a typical publisher Web site. Such a wrapper has about 20–50 rules and 200–500 lines of code.

Compared to an ad-hoc solution for a virtual Web site, the HyperView System encapsules the complete details of the HTTP protocol, provides powerful matching facilities for data extraction from HTML pages, libraries for parsing common notations such as date or time formats, a powerful query language, and a generic Web interface. Together with the guideline of the HyperView methodology this reduces the development effort considerably. Moreover, the robustness and easy maintainability of the HyperView approach reduces the maintenance costs compared to conventional programs.

A bottleneck currently remaining in setting up a virtual Web site is the missing tool support for design and maintenance of views. The graph-based approach taken in the HyperView methodology lends itself easily to visualization and graphical editing of rules. HVQL will serve as a target language for the view design and schema discovery tools [Öksüz, 1999a, Öksüz, 1999b] currently under development. We expect that these graphical development tools for HyperViews will reduce the time for constructing and testing views significantly.