World Wide Web 3 (2000) 95–109

SERFing the web: The Re-Web approach for web re-structuring *

Li Chen, Kajal T. Claypool and Elke A. Rundensteiner

Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609, USA
E-mail: {lichen,kajal,rundenst}@cs.wpi.edu

In our emerging digital paper-less society, massive amount of information is being maintained in on-line repositories and diverse web site representations of this information must be served over the Internet to different user groups. E-commerce and digital libraries are two representative sample applications with such needs. In this paper we present a database-centric approach called Re-Web that addresses this need for flexible web site generation, re-structuring, and maintenance. Re-Web is based on two key ideas. First, we exploit the web site structure by associating web semantics (XML equivalents) with the modeling constructs of the ODMG object model to aid the web site generation process. By capturing the logical structure of web views within the OODB system, we can efficiently maintain the web views using standard database techniques. Secondly, to ease the process of specification and construction of multiple customized web view sites, we also propose the notion of generic web view transformations that are encapsulated into re-usable templates. Thus desired new web view sites can be generated simply by applying the corresponding transformations on the underlying database to produce web view classes and then by applying the web semantics on the newly built view classes. The Re-Web system has been implemented using PSE by Object Design Inc. as object repository, ODMG as object model, OQL as transformation language, SERF as OODB evolution facility and IBM XML parser and LotusXSL processor to aid the web site generation. A case study using Re-Web is also presented to illustrate the working of the system. To the best of our knowledge, Re-Web is the first web site management system focusing on the issue of re-usable view generation templates at the content and not at the presentation style level of abstraction.

1. Introduction

In recent years, there has been a large proliferation of information on the web. Most of this information currently exists in the form of HTML (hyper-text mark-up language) pages. These HTML pages publish information that perhaps previously was represented as simple files, as a database or even as a paper trail. The advent of the web with its ease of authoring information and its wide accessibility has raised the expectations of users. Today users change the look and feel of a web page frequently, add more information to their web pages to make them richer, or re-structure the way the information is organized. For individual users and small organizations, this task while tedious may perhaps be manageable. For a large organization, however, the web administrators who have to deal with maintaining possibly huge numbers of pages and re-structuring them as per their users' needs are faced with a mammoth task.

While some emphasis may be placed on how the documents look, a substantial amount of work is devoted to the structural organization of the information in the web pages. That is, there is an inherent implicit structure in HTML documents. We exploit this structure with the help of database technology to bring to the users, especially web administrators, a tool, Re-Web [Claypool *et al.* 1998c], that allows for easy web-site generation and re-structuring. Previous approaches for web site management have invented either specialized web query languages [Atzeni *et al.* 1997], web page scheme languages [Fernandez *et al.* 1997b], or web hyper-graph data models [Arocena and Mendelzon 1998]. In Re-Web, instead we embrace existing object-oriented database (OODB) technology [Cattell *et al.* 1997], in particular, the ODMG standard, and show it to be a sufficient basis for effective web site re-structuring.

Web page generation. In Re-Web we define the notion of web semantics, i.e., a mapping of database constructs to the web, which allows us to design the organizational structure of the information for the web pages at the OODB level. We propose using the XML [Bray *et al.* 1998] format as a middle layer presentation, i.e., between the object model and the final web constructs in HTML. XML is an emerging standard for web data exchange providing an explicit structure for documents. Hence it facilitates the data mapping to and from a database easier and provides easier maintainability. In addition, XML distinguishes between the content and the actual presentation, a model that fits in with using a database as the back-end for storing the actual content (the data). Thus, as commonly done in other approaches [Cruz and Lucas 1997], the addition of visual presentation styles to be applied to the web site structure specifying, for example, font sizes or list indent bullets can be done in a stage separate from the web site generation process.

Web-Site re-structuring. Given our capability to generate web pages from an object model, we now can offer the ease

* This work was supported in part by several grants from NSF, namely, the NSF NYI grant #IRI 94-57609, the NSF CISE Instrumentation grant #IRIS 97-29878, and the NSF grant #IIS 97-32897. Dr. Rundensteiner would like to thank our industrial sponsors, in particular, IBM for the IBM partnership award. Li Chen would also like to thank IBM for the IBM corporate fellowship. Special thanks also goes to the PSE Team specifically, Gordon Landis, Sam Haradhvala, Pat O'Brien and Breman Thuraising at Object Design Inc. for not only software contributions but also for providing us with a customized patch of the PSE Pro2.0 system that exposed schema-related APIs needed to develop our tool.

of web re-structuring simply by applying SERF [Claypool *et al.* 1998a]. SERF is a transformation framework built on top of ODMG compliant OODB systems that enables re-structuring of existing classes or the building of new view classes in OODB systems. New re-structured web pages can then be generated from such re-structured data schemas or views simply by the application of the web semantics as indicated above.

SERF transformations combine together schema evolution primitives along with object transformations and OQL queries to express arbitrarily complex re-structurings. We have found that there is a set of re-structuring transformations that are applied frequently to web sites such as combining of two types of pages into one or flattening a complex linked page. In general, it would be very useful to capture such transformations in a generic fashion such that web designers do not have to re-write them over and over again. We thus utilize the notion of *transformation templates* first introduced in SERF [Claypool *et al.* 1998a] to bring to the user a library of common re-usable re-structurings. Using such a library of transformation templates, the job of re-structuring the web is reduced to simply applying a re-structuring template on the appropriate class(es) and then generating the web pages from the re-structured class(es) with the click of one button. In this paper, we demonstrate with the help of several examples that a transformation library of these generic web view transformations can be a valuable resource for simplifying the web generation and restructuring process. To the best of our knowledge, Re-Web is the first web site management project focusing on the issue of re-usable view generation templates [Claypool *et al.* 1998a, b].

In this paper, we also demonstrate via a case study that OODB systems are indeed suitable and in fact sufficient for supporting a wide variety of diverse web site views over the same underlying data simply by capturing web semantics with the object model constructs.

Implementation. We have implemented a fully functioning Re-Web [Claypool *et al.* 1998c; Rundensteiner *et al.* 2000] prototype that is compliant with the ODMG standard for the DBMS back-end and with XML for the web semantics front-end. It is build using the ODMG object model, applies the OQL query language as the database transformation language, works with an ODMG-compliant system repository, and assumes Java's binding of ODL. Our Re-Web approach thus is general and could easily be ported from our platform (which is the PSE system by Object Design Inc. [O'Brien 1997]) to other ODMG-compliant platforms. Mapping from ODMG into XML allows the web application front-end be aware of the content as well as the structure of the information and thus allows us to exploit existing technologies such as presentation formatting by just plugging in powerful style construction tools, for example, LotusXSL [IBM Alphaworks 1998]. Our Re-Web prototype was demonstrated at SIGMOD 2000 [Rundensteiner *et al.* 2000]. Experiences we gain from our on-going development effort of building the Re-Web prototype thus may directly benefit others that want to incorporate the Re-Web approach into their system [Chen and Rundensteiner 2000].

Contributions. To summarize, the contributions of the Re-Web project include:

(1) the identification of a novel approach for powerful web-site generation and restructuring based solely on standard database technology,

(2) design of web semantics with the ODMG database constructs both at the schema and at the data level,

(3) the notion of re-usable transformation templates for powerful web site re-structuring, offering a wide array of advantages to both end-users and developers as detailed above,

(4) the development of a library of web schema transformations that represents a potentially valuable resource for both novice and expert web-site designers (parallelling the concept of HTML style files),

(5) the design and implementation of a fully functional Re-Web system using an ODMG-compliant database as well as XML/XSL technology [Clark 1998; Microsoft Inc. 1998; Thompson 1998] as a proof of concept of the approach.

Overview. The rest of the paper is organized as follows. Section 2 describes our overall approach while sections 3 and 4 present the web semantics model for generating web pages from the ODMG model constructs and the re-structuring at the database level, respectively. The system architecture and module design are described in section 5 and we discuss the case studies in section 6. Section 7 covers work related to ours, while section 8 concludes this paper.

2. The Re-Web approach

The Re-Web approach is a database centric approach. In our approach we map the ODMG object model to the XML model to automatically generate web pages based on the actual data that is present in the database. Within the database itself, we use SERF [Claypool *et al.* 1998a] to allow us to do complex re-structuring of the object schema as well as to create new view schemas which in turn can be mapped to the new re-structured web pages. While ours is not the only database centric approach, it is the first to offer such a high degree of flexibility and power in terms of the re-structuring.

For example, the web site management project Araneus [Atzeni *et al.* 1998], allows the web administrator to describe the web site schema via defining views based on the underlying relational database. However, by utilizing

relational database technology as its back-end data repository, the hierarchical structure of the web site needs to be molded into the flat structure supported by the relational model. In addition, they had to develop a special purpose language, Penelope, to translate from the relational tables to the desired web structure. Being based on the relational model, the process of generating web pages may generally involve many join operations on the flat data. The relational model chosen for the intermediate repository is thus not a natural model for capturing the hierarchical web site schemas requiring an additional step to overcome the model impedance mismatch.

Similarly, another web site management tool Strudel [Fernandez *et al.* 1997a, b], provides flexible construction of multiple views over the data residing in a graph-based repository. However, the graph repository in the Strudel system serves only as a storage medium. The database does not include any direct re-structuring capabilities; and while users are able to define view schemas, there is no notion of being able to define views over views. That is, re-structuring an existing view, a feature supported by SERF where we simply treat view classes as base classes, is not possible. Moreover, there is no notion of re-usability, in terms of re-using the queries that define the desired view of the web page, as offered by SERF transformation templates.

Our Re-Web system, on the other hand, is a layer defined on top of an OODB system, which serves not just as a back-end data storage, but is also capable of providing complex transformations for re-structuring schemas.

2.1. Web page generation and re-structuring

The goal of the Re-Web system is to provide an automatic web site management tool suite. We provide three tiers of web semantics representations (see figure 1). The web site tier is the HTML representation, i.e., the web pages themselves, which include some visual metaphors and styles. Ideally the users navigate through the whole web site via its HTML pages in a unique and consistent fashion. The ODMG data model is at the OODB tier, the equivalent loss-less representation of the HTML web pages. At this level the database services such as re-structuring can be utilized and exploited to provide the same for the web pages themselves. A re-structuring desired for the web-site is translated to an equivalent transformation template at the database level which then is responsible for the re-structuring of the schema and the associated objects. The new view schema, the result of the re-structuring, represents the basis for automatically producing the desired re-structured web site. In an effort to make this possible, we use XML in the middle tier to capture the web context (semantics) and allow the inter-changeability between the very structured object model and the not so structured HTML document model.

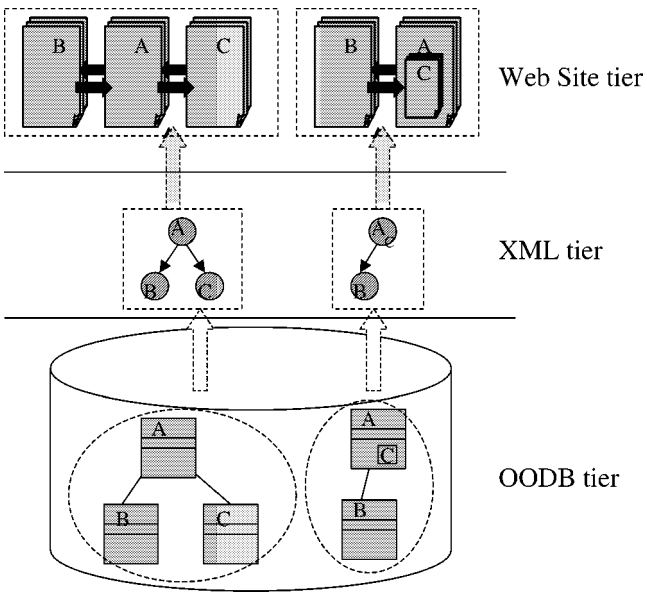

Figure 1. Three tiers of web page generation architecture.

The goal of a view transformation in the context of Re-Web is to re-structure a given set of types in order to create a view schema modeling the desired schema of a web site. A SERF tool-set within the Re-Web system provides the means of doing view transformations by combining a query language with a standard set of view and schema evolution primitives to produce other more complex view transformations. We will detail the procedure of how SERF accomplishes the view schema transformations in section 4 and then walk through a case study in section 6.

Advantages of the Re-Web approach. The Re-Web system offers significant advantages to its users both in terms of ease of use as well as in terms of enriched capabilities. Re-Web brings to its users:

- OODB-based web generation: Web semantics allow a user to interchange between the XML representation and the ODMG object model. Data in the OODB can be automatically translated to web pages using Re-Web.
- Advanced restructuring facilities: Using SERF [Claypool *et al.* 1998a], web administrators can manipulate and re-structure base classes stored in the OODB and thus easily generate the desired structure of web pages.
- Support for multiple web views: Given that web pages are generated from the data in the OODB, we can also utilize the power of the underlying database to define multiple views over the same data. Thus, a web administrator can easily provide diversely different web page structures over the same data.
- Ease of update propagation: While not explicitly addressed in this paper, one of the significant advantages of the Re-Web approach is the exploitation of DBMS technology, such as query optimization or view maintenance under updates. The views defined for obtaining the desired web pages are defined in the database. The OODB system is aware of these and hence the propagation of updates to the different web pages is reduced

to the problem of updating a view [Gluche *et al.* 1997; Keller 1982; Scholl *et al.* 1991] and then re-generating the web pages.

- Re-usable transformation templates: SERF allows the storage of transformations in their generalized form (templates) in a template library. For a common set of re-structuring transformations, the web administrators can re-use previously written transformation templates and apply them to the appropriate classes.
- Flexible specification of transformation templates: If a desired transformation is not found in an existing template library, Re-Web via SERF allows the users to specify their own transformation templates using OQL and the schema evolution primitives. These newly defined transformation templates can be added to the template library for later re-use.

3. Mappings between web semantics representations

In section 2 we have presented the three-tiered architecture, i.e., the ODMG object model [Cattell *et al.* 1997], the XML model [Bray *et al.* 1998] and the HTML representation of the web pages. Here we provide in table 1 the details for a bottom-up vertical mapping from the ODMG object model to the HTML representation of each element. For the purpose of the Re-Web framework we limit our discussion of the ODMG Object Model to Java's binding of the object model and define the web semantics for the same as our tool is built on top of the Java-based OODB, PSE [O'Brien 1997]. The web semantics presented here can be easily extended to accommodate other ODMG modeling constructs as needed.

- **Types.** The basic category for an ODMG compliant database is *types* (also referred to as classes). A type definition gives the structure and the behavior specification for its instances, it may be rather complex to have hierarchically nested subtypes and attributes. It is correspondingly mapped to a **Data Type Definition** (DTD) in the XML context. DTD provides a standarized means for declaratively specifying the types of an XML document, such as what elements an element can contain, the way they can be composed and the typings and default values of their attributes, etc. The XML DTD in turn provides the **web-page-structure** for a set of HTML pages, underneath which their implicit structures share the organizational similarity reflected by the web-page-structure.
- **Object.** The two basic modeling primitives for an ODMG compliant database are *objects* and *literals* (*or immutable objects*), both of which are categorized by their *types*. An object in the OODB system is mapped to an **XML document** compliant with its DTD. In the Re-Web prototype system, we assign a system identifier (i.e., Uniform Resource Identifier **URI**) for each generated XML file, thus it functions correspondingly like a unique **object identifier** (OID) of an object in the database. By combining an XML document with a stylesheet template and passing them through a transformation engine, we can obtain a browsable HTML **web page** with a **URL** as its identifier.
- **Literal.** Generally, a *literal* in ODMG is classified by its structural complexity into one of these three types: **atomic**, **struct** or **collection**. In XML, it corresponds to the text information that represented as **AttValue, CDATA** or **PCDATA**. An atomic literal in the database can be mapped either to an AttValue in XML if it paired with an attribute, or to a CDATA if it is enclosed by an element. PCDATA can be used in the XML-tier as the correspondences of struct or collection literals at the database level. These text constructs in XML are represented as the fixed **web items** that a HTML web page is composed of.
- **Extent of Type.** The *extent* of a type in the ODMG object model reflects the collection of objects that share the same structure, which correspondingly map to a set of **XML documents** that conform to the same DTD. In the third-tier, i.e., HTML, this corresponds to a collection of **web pages** that have the same web-page-structure. Thus the number of objects in the extent of a type is equal to the number of web pages that are generated.
- **Relationship.** There commonly exists a certain *relationship* between types in an ODMG compliant database. This relationship corresponds in the XML-tier to the element **IDREF** definition. Though an IDREF in a DTD does not explicitly declare its referring element type, whether the reference is meaningful can be checked for a particular XML document when its IDREF is instantiated. In a HTML web page, the IDREF is mapped to a **HTML link** via which the navigation to another web page is achieved.

Table 1
Mapping between ODMG, XML and web semantics.

ODMG primitives	XML constructs	Web semantics
Type	DTD	Web-page-structure
Object	XML document	Web page
OID	URI	URL
Literal	AttValue/CDATA/PCDATA	Web-item
Extent of a type	XML documents of a DTD	Web pages of a web-page-structure
Relationship	IDREF	HTML link
Schema	Set of DTDs	Web-site-structure

- **Schema.** An **ODMG schema** is composed of a set of object and literal type definitions and a class hierarchy. A database schema hence can be represented in the XML-tier as a **DTD hierarchy**, i.e., a set of DTDs with element references. It then can be used to model a web site by defining its structure, called its **web-site-structure**. Given an ODMG schema corresponds to a set of type definitions, a DTD hierarchy corresponds to a set of XML document type definitions and a web-site-structure for a given web site corresponds to a set of web-page-structures.

Using the above web semantics mapping, a set of hierarchically inter-linked web pages can be finally generated from a database associated with an ODMG schema. However, not all concepts of ODMG have a one to one mapping. For example, the *inheritance* defined in ODMG has no parallel in the XML and HTML tiers. Also, there is no obvious correspondence in DTD with respect to the *behavior* specification of a type definition in the ODMG data model.

We now show this mapping from a data model via the XML constructs to the web site and the HTML pages by the example shown in figure 2.

Assume we have an original schema modeling a department, which consists of two types of classes: *Course* and *Professor*. We thus first map each of the classes (for example, the Course class) in the database schema into one XML DTD (Course_DTD) and further an implicit web-page-structure for web pages of this type. We observe from figure 2 that there are object instances *obj4*, *obj5* of type

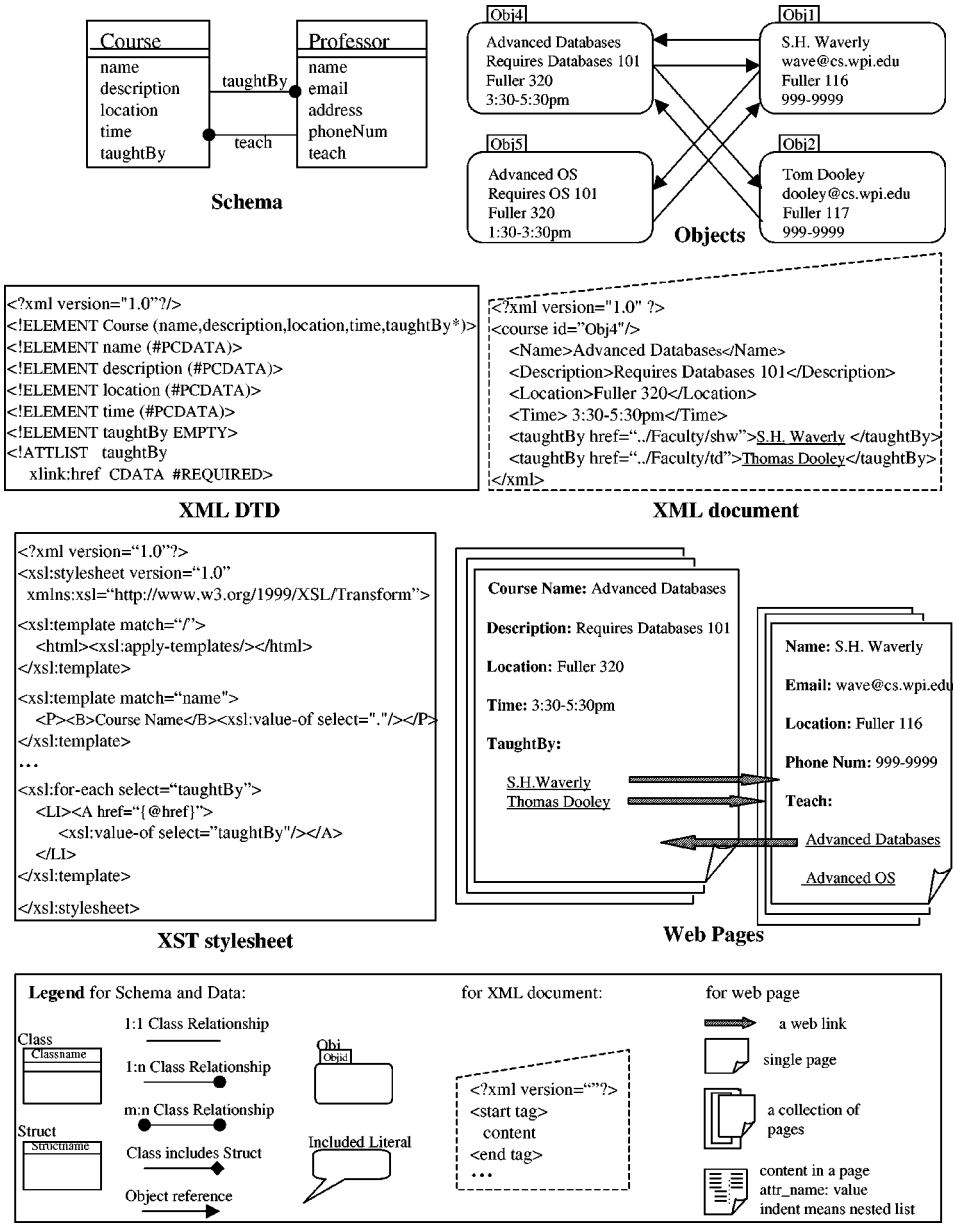

Figure 2. The web object representations in three tiers.

Course and *obj1, obj2* of type *Professor*. For each database object (for example, the obj4), we represent its content as well as the associated type information in the middle-tier using XML data and the enclosing tag, respectively. It is noteworthy that there exists a bi-directional relationship between the *Course* class and the *Professor*. *Course* class has a collection property *taughtBy* that refers to the *Professor* class, while the latter offers courses that must be instances of the *Course* class. This referential constraints are reflected by the inter-linked data objects of two types in the database, the id-referred XML documents in the XML-tier and in the HTML-tier the traversable html links between web pages. For example, a Course object *obj4* is *taughtBy* both of the professors, *obj1* and *obj2*. The XML document representative of *obj4* thus has an element *taughtBy* with an IDREF referring to two other XML documents of Professor type. Correspondingly, the *Advanced Databases* Course page has a list of two URLs pointing to the respective Professor pages.

4. Re-structuring the web: The Re-Web transformation

The mapping from the ODMG object model to web objects as described in section 3 allows the automatic generation of web pages from the database by translating the database semantics of types and objects to the web semantics of web site schemas and web-pages, respectively. However, this allows us to only produce exactly one view schema and defines one structure for the web pages. In order to support multiple web site schemas as well as web pages, we define view schemas over the base classes in the database. This set of diverse view schemas once generated can then be mapped to XML and ultimately down to HTML to produce diverse web sites over the same data. In this section, we now show how we can flexibly re-structure the underlying database to support and generate the desired set of web site schemas and web pages.

A web site may simply need to be adjusted requiring the corresponding schema to be slightly modified perhaps by the addition of an attribute or the deletion of one. We propose the use of schema evolution primitives for in-place manipulation of the schema. Table 2 presents the set of schema evolution primitives that we have defined for the ODMG object model as part of our work.

In summary, a view transformation in our framework lets the user combine an object transformation language with a standard set of schema evolution primitives and view primitives to produce arbitrarily complex transformations. Moreover, these transformations are generalized and stored in a standard library for later re-use. Transformations in this general form are called *templates* in the framework and the library, the *template library*. The Re-Web framework through this powerful re-structuring mechanism succeeds in giving the user the *flexibility* to define web-site structures of their choice, the *extensibility* of defining new complex structures through new view transformations, and the *re-usability* of these structures through the notion of view transformation templates.

4.1. Re-Web view transformation

The goal of a view transformation in the context of Re-Web is to re-structure a given set of types in order to create a view schema modeling the desired schema of a web site.

Figure 3 is an alternative view schema and web site view for the example schema given in figure 2. In figure 2, the structure of the web site reflects the structure of the given database schema. However, this web site schema may not meet the user's requirements and the user may instead desire a web view of the same data as depicted in figure 3. The generation of the web view depicted in figure 3 from the database schema given in figure 2 first necessitates a re-structuring of the underlying database to correctly reflect the schema of the web site. We use a Re-Web view transformation to obtain the required structure of classes in a re-usable manner.

We illustrate the steps involved in a view transformation using this example. The example transformation we work with is *convert-to-literal* given in figure 4 that is defined as the replacement of a collection of referenced types with a collection of structures with the same definition. For example, to get from the schema in figure 2 to the schema in figure 3, the convert-to-literal transformation needs to be applied to the attribute `taughtBy` defined for the class `Course`. This converts the collection of objects of type `Professor` to a collection of literals of `Struct-Professor` structure type. Figure 4 shows

Table 2
Taxonomy of schema evolution primitives.

Term	Description
add-class$(c, \mathcal{C})$	Adds new class c to $\mathcal{C}$ in the schema **S**
delete-class(c)	Deletes class c from $\mathcal{C}$ in the schema **S** if $C(t) = \emptyset$
change-class-name(c, x)	Changes the class name from c to x
add-ISA-edge(c_x, c_y)	Adds an inheritance edge from c_x to c_y
delete-ISA-edge(c_x, c_y)	Deletes the inheritance edge from c_x to c_y
add-attribute(c_x, a_x, t, d)	Add attribute a_x of type t and default value d to class c_x
delete-attribute(c_x, a_x)	Deletes the attribute a_x from the class c_x
change-attribute-name(c_x, a_x, a_y)	Changes the name of the attribute a_x in the class c_x to a_y
change-attribute-domain(c_x, a_x, d_x, d_y)	Changes the domain of the attribute a_x from d_x to d_y in the class c_x

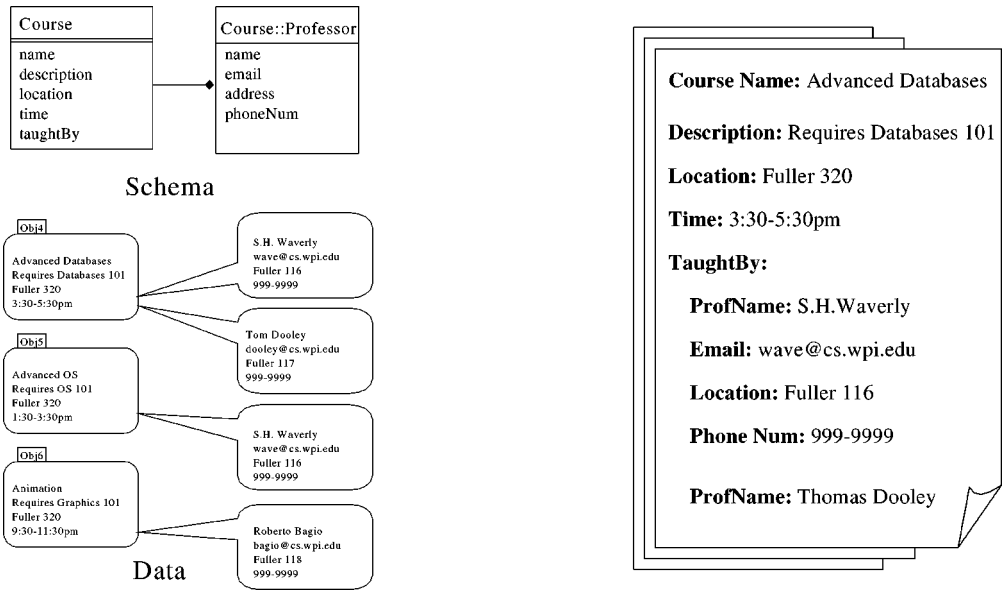

Figure 3. The database schema and the matching web page generated from them.

```
// define a named query for the view                                  } Step B
define ViewDef (viewClass)
   select c
   from viewClass c
   where c.department = "Computer Science";

// create view CourseView from the Course class                       } Step C
// and create struct Struct-Professor  from the Professor class
create_view_class (Course, CourseView, ViewDef (Course ));
create_view_struct (Professor, Struct-Professor);

// add an attribute Struct-Professor to the view                      } Step D
add_attribute (CourseView, Struct-Professor,
               collection<Struct-Professor>, null);

// get all the objects of class                                       } Step E
define Extents (cName)
   select c
   from cName c;

// for each of the CourseView object, find its referred Professor     } Step F
// objects via theTaughtBy  attribute and convert them into a
// a collection of Struct-Professor .
define flattenedCollection (object)
   select (Struct-Professor)p.*
   from Professor p
   where exists (p in object.taughtBy);

for all obj in Extents (CourseView )
   obj.set(obj.Struct-Professor, flattenedCollection(obj));

delete_attribute(CourseView, taughtBy);                               } Step D
```

Figure 4. Convert-to-literal transformation.

```
begin template convert-to-literal (Class mainclassName,
                                   String mainViewName,
                                   Attribute attributeToFlatten,
                                   String structName)
{
   // find the class that needs to be flattened given the attribute name
   refClass = element (                                               } Step A
                select a.attrType
                from MetaAttribute a
                where a.attrName = $attributeToFlatten
                and a.classDefinedIn = $mainclassName );

   // Create the view class
   create_view_class ($mainclassName, $mainViewName, View ($mainclassName));

   // flatten refClass to a struct
   create_view_struct ($refClass, $structName);

   // add a new attribute to hold the struct
   add_attribute ($mainViewName, $structName, collection<$structName>, null);

   // get all the objects of class
   define Extents (cName)
      select c
      from cName c;

   // convert a collection of objects to a collection of structures.
   define flattenedCollection (object)
      select ($structName)p.*
      from $refClass p
      where exists (p in object.$attributetoFlatten)

   for all obj in Extents ($mainViewName)
      obj.set(obj.$structName, flattenedCollection(obj));

   // remove the attributetoFlatten attribute
   delete_attribute ($mainViewName, $attributeToFlatten);
}
end template
```

Figure 5. Convert-to-literal template.

the *convert-to-literal* view transformation expressed in our framework using view creation and definition, OQL statements, schema modification primitives, and system-defined update methods. In this example, the object.set() methods are the system-provided methods.

- **Step B: define the views.** In OQL, named queries are treated as a view mechanism. Thus, OQL can be used to define a view based on classes that exist in the database. This view definition gives not only the definition of a view but also allows the selection of the extent of the view from the database. Step B of figure 4 shows the definition of a view over the `Course` base class that structurally maintains all the properties of the base class `Course` but defines the extent of the view to only have the `Computer Science` courses.
- **Step C: create the views.** View support by OODB systems is often offered in terms of some view primitives to create view classes and named view structures with a query language providing the definition. Step C in figure 4 shows the primitive for creating the view class `CourseView` from the base class `Course` using the definition `ViewDef`.
- **Step D: change the schema.** We require that all structural changes, i.e., changes to the base schema as well as the view schema, are exclusively made through the schema evolution primitives. This helps us guarantee the schema consistency after a transformation [Claypool *et al.* 1998b].

- **Step E: query the objects.** As a preliminary to performing object transformations, we need to obtain the handle for objects involved in the transformation process. This may be objects from which we copy object values (e.g., `Professor` objects in Step E) or objects that are modified (e.g., `CourseView` objects in Step F).
- **Step F: change the objects.** Although not an essential for all view transformation (for example, delete-attribute does not require this), the next step to any view transformation logically is the transformation of the objects to conform to the new view schema. Through Step E, we already have a handle to the affected object set. Step F in figure 4 shows how a query language like OQL and system-defined update methods, like *obj.set*(...), can be used to perform object transformations.

In general, a view transformation in our Re-Web framework uses a query language to query over the schema repository, i.e., the metadata and the application objects, as in Steps A and E. The transformation also uses the query language for views, i.e., to define new views, and due to lack of full view support in OQL to invoke operations to store these views in the database as shown in Step B and Step C. The schema evolution primitives for in-place structural changes and the system-defined functions for updating the objects can also be invoked from OQL, as in Steps D and F.

4.2. Re-Web view transformation templates

A Re-Web transformation as given in figure 4 allows a user to flexibly define view transformations. However, they are not re-usable across different schemas, for example the given transformation works for only the `Professor` and the `Course` class. For this reason, Re-Web adopts from SERF [Claypool *et al.* 1998a] the notion of templates. A template is an arbitrarily complex transformation that has been *generalized* via the use of meta-data and has a name and a set of parameters.

Thus, the key step for converting a transformation to a template is to generalize it such that it no longer pertains to a specific set of classes. We do this in two steps. First we introduce the notion of *parameters*. Using these parameters as variables, we re-write the transformation to obtain necessary information from the Schema Repository as well as to apply schema changes based on the input parameters. For example, in figure 4 we are aware that the class `Professor` is being converted to a structure (see Step C). However, to make this transformation general, we now simply pass in as a parameter the attribute `taughtBy` and use the Schema Repository to discover the actual class that needs to be converted to a structure. This is shown as Step A in the template given in figure 5.

By parameterizing the variables involved in a transformation such as the input and the output classes, e.g., the `Course` and `Professor` classes in our example, and their properties, e.g., the `taughtBy` attribute in our example, and assigning a name to the transformation e.g., *convert-to-literal* in our example, a transformation becomes a *generalized re-usable* transformation. This *template* is applicable to any application schema that meets its requirements. For example, the *convert-to-literal* template requires a relationship to exist between two classes. Thus, this template is applicable to any two classes in any given schema as long as they have a relationship between them. That is, it is not tied to one specific application schema, figure 5 shows the generalized *convert-to-literal* transformation of figure 4 as a template. Vice versa, the *convert-to-literal* template shown in figure 5 can be instantiated with the variables `Course` and `taughtBy` and results in the Re-Web transformation depicted in figure 4. A Re-Web template is thus a named sequence of OQL statements extended with parameterization that can be translated down to pure OQL statements during the process of instantiation.

Templates themselves can be nested and embedded within each other and can be used to write more powerful transformations. For example, in figure 7 we show an example of a recursive template that applies an in-place variation of the basic *convert-to-literal* template, *inline* template given in figure 6, iteratively to an attribute that needs to be flattened. Here we assume that a class `inliningClassName` needs to be flattened `nestLevel` levels down to encapsulate all the attributes for all the classes that initiate at the *attrToFlatten* attribute in the *inliningClassName*. Attributes that are complex, i.e, whose domain is a class and thus belongs to metaclass are flattenable. The nested-convert-to-literal template (see figure 7) first invokes the *inline* template to immediately flatten the class that is referred to by the `attrToFlatten`. It then finds all the complex attributes that are now present

```
begin template inline (className, refAttrName)
(
  refClass = element (
                select a.attrType
                from MetaAttribute a
                where a.attrName = $refAttrName
                and   a.classDefinedIn = $className; )

   define localAttrs(cName) as
      select c.localAttrList
      from MetaClass c
      where c.metaClassName = cName;

   // get all attributes in refAttrName and add to className
   for all attrs in localAttrs(refClass)
      add_atomic_attribute ($className, attrs.attrName,
                            attrs.attrType, attrs.attrValue);

   // get all the extent
   define extents(cName) as
      select c
      from cName c;

   // set: className.Attr = className.refAttrName.Attr
   for all obj in extents($className):
      for all Attr in localAttrs(refClass)
         obj.set (obj.Attr, valueOf(obj.refAttrName.Attr));

   delete_attribute ($className, $refAttrName);
)

end template
```

Figure 6. The inline template.

```
begin template nested-convert-to-literal ( Class inliningClassName,
                                           Attribute flattenAttrName,
                                           Integer nestLevel)
{

    // inline the next level complex attribute into the specified inlineClass.
    inline ($inliningClassName, $flattenAttrName)

    // decide for a class which of its attibutes is complex and to be flattened.
    define findAttrToFlatten (className) as:
       select a
       from MetaAttribute a, MetaClass c
       where a.classDefinedIn = className
       and a.attrType = c.name
       and nestLevel != 0

    // starting at the inliningClass all complex attributes are flattened.
    for attr in findAttrToFlatten ($inliningClassName):
        nested-convert-to-literal($inliningClassName, attr, $nestLevel-1);
}
end template
```

Figure 7. Nested inline template by re-using basic inline template.

in the new `inliningClassName`. The process is repeated by a recursive call to the *nested-convert-to-literal* template until either there are no more complex attributes or the desired `nestedLevel` is reached.

In summary, the templates provide users not only with the advantages achieved by our transformations, i.e, a user can specify their own semantics for transformations, but also allows *re-usability* of these transformations by parameterizing them. As demonstrated by the example in figure 7, templates can be re-used and embedded within templates and can be applied to any application schema for the generation of a wide variety of web structures.

5. Re-Web framework

The Re-Web tool suite targets users such as web site administrators who are faced with the heavy workload of web site generation, restructuring and maintenance. This tool greatly facilitates the automatic web site generation in the sense that a variety of home pages with preferred styles can be generated all at once to reflect the corresponding object instances in the underlying database. It also empowers users with the flexibility of restructuring their web site schema by applying SERF transformation templates directly within the database system.

5.1. System architecture

Figure 8 gives the general architecture of the Re-Web framework. As further detailed below, the Re-Web framework mainly includes four subsystems: the underlying OODB system, the SERF schema restructuring system, the WebGen web view generation system and the GUI system. The bottom part is the OODB system consisting of the components that we expect any underlying OODB system to provide. This includes technology to allow uniform manipulating of both data as well as metadata. The middle layer contains the SERF system to provide the schema restructuring service and the WebGen system for automatic web site generation. The top part of the figure represents the Graphical User Interface (GUI) allowing users to access other three subsystems, i.e., to edit the schema templates and the stylesheets, and to view the schemas that are stored

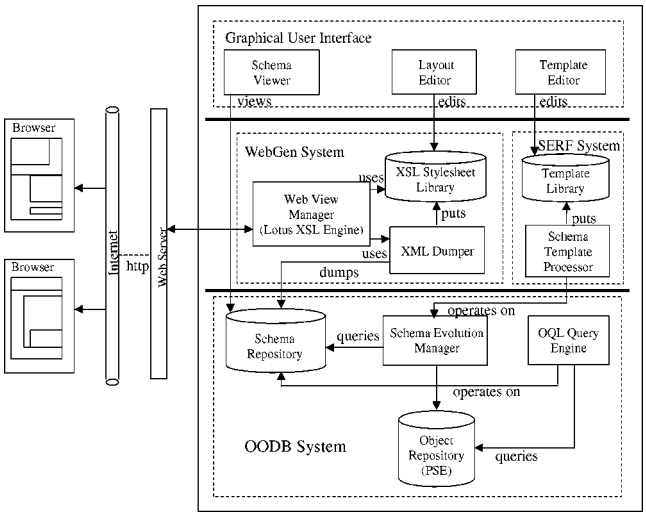

Figure 8. Architecture of the Re-Web framework.

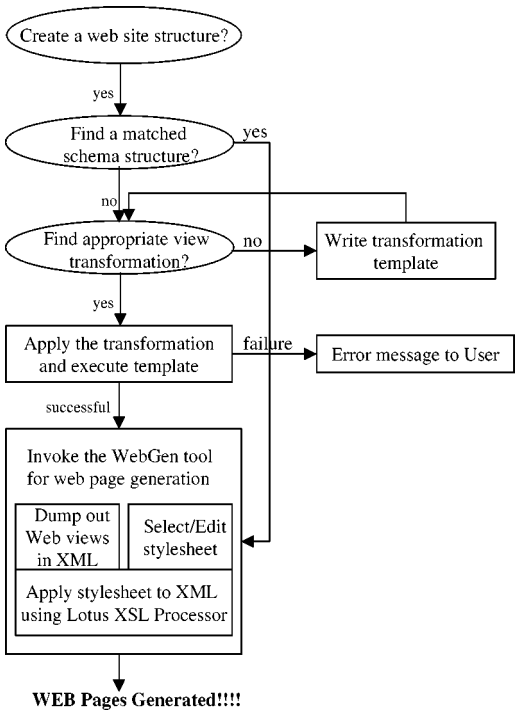

Figure 9. Interaction of the different modules in the Re-Web framework.

in the Schema Repository. The left side of the architecture picture shows that front-end users can use browsers to navigate the web views in HTML pages generated by our system.

Figure 9 shows how the system modules interact with each other. To construct or restructure a web site schema, the first step is to check whether it already exists in the Schema Repository. If such a schema structure is available, there is no need to utilize SERF for schema transformation, otherwise a desired one can be obtained by applying SERF transformation templates. The next step is to invoke the WebGen tool for automatic web page generation. An XML representation of the desired web schema is dumped out

and an appropriate stylesheet is picked to generate the corresponding web pages.

5.1.1. OODB system modules assumed by the Re-Web framework

The system components of an OODB that the Re-Web framework depends upon include:

- **Schema Repository and Object Repository.** The *Schema Repository* module stores and manages metadata which defines the schema of a database. It is also used by the OODB at runtime to guide its access to the database [Cattell *et al.* 1997]. OQL can query over the *Schema Repository* to gather system information for use in the transformations. The object instances of an application schema are stored in the *Object Repository*, which in our prototype is PSE – the persistent storage engine available from Object Design Inc.
- **Schema Evolution Manager.** The *Schema Evolution Manager* provides an interface for the execution of a set of schema evolution primitives. It interacts with the schema repository which contains information on each class and its placement in the class hierarchy. It also updates the PSE data dictionary so as to keep it in sync with our *Schema Repository*. For our implementation, we have assumed that all additions, deletions and modifications to the schema happen through the interface that we have provided. The *Schema Evolution Manager* is also responsible for the migration of objects from the existing (old) class definition to the changed class definition, thus keeping them updated and consistent with the schema change.
- **OQL Query Engine.** Re-Web requires the OODB system to provide a query language capable of expressing a large set of view definitions and of transforming objects of one type to another type. For our system, we choose OQL. Beyond selection, OQL provides some support for creating, deleting and modifying objects. OQL is also capable of invoking view primitive and schema evolution primitives. OQL does all of the above through the invocation of system-defined update methods.
  Although PSE provides a mechanism for querying collection objects, it is very basic (does not have joins) and does not meet all of our criteria for SERF transformation support. Thus as part of the implementation effort, we have implemented an OQL interface for PSE using Fegaras's object algebra approach [Fegaras 1997]. The key functionality of the *OQL Query Engine* is to parse and map a given OQL statement to the right PSE system functions in order to get the required result.

5.2. SERF system

Build upon an OODB system having the assumed components, the SERF system provides all of the functionality for storing, retrieving and executing templates.

- **Schema Template Processor.** The *Schema Template Processor* provides the functionality for the execution of a template. The template processing begins with the user supplying the input parameters, then it will do a *type-check* to ensure the template interface signature. If it matches, then a *bind-check* follows to check the existence of these actual parameters in the schema on which they are being applied by accessing the *Schema Repository*. The SERF template is instantiated by replacing each variable with its bound parameter after all the checks are completed successfully. The instantiated SERF template now corresponds to pure OQL statements, i.e., we call it an OQL transformation. The *OQL Query Engine* provides an interface for the syntax-checking, the *parsing* and the *execution* of the OQL transformation.
- **Schema Template Library.** The SERF templates for a particular domain or specific purpose can be collected in a *SERF Template Library*. The SERF system can install multiple template libraries. A SERF template can be stored and retrieved via a complete path of `libraryName.templateName`. A customized template can be dynamically generated and put into the library. A template object contains the *name, description*, a set of *input parameters*, a list of *keywords* and the *oql source* for its SERF template.

5.3. WebGen system

The WebGen system, built over the SERF system, is responsible for generating the customized web site composed of pages that are reflecting correspondingly the data objects stored in the backend OODB system. In detail, the WebGen system includes the following modules:

- **XML Dumper.** For the mapping of web semantics through three tiers: OODB, XML and Web Site (see table 1), the *XML Dumper* completes the first stage of the mapping from database data model to its XML representation. By translating a schema and its associated object instances in an OODB system into a set of DTDs with their valid XML files, the web site structure is captured and ready for the generation of actual web pages by applying a visual metaphor for presentation purpose. The *XML Dumper* then conducts the mapping procedure while walking through all the classes of the schema hierarchy and their object collections in the Schema and Object Repository, respectively. For each class, the *XML Dumper* translates the classname into the RootElement of an XML DTD, and maps the flat properties into leaf elements, nesting properties into hierarchical elements, and the references out to other classes (relationships) into XLinks. This way an XML DTD is produced. The same mapping rule can be applied on the object instances of the schema to generate a set of XML data files that conform to the DTD. The set of XML DTDs explicitly captures the web schema.

- **Web View Manager.** After a database schema and its associated data objects have been represented in XML files by the XML Dumper, we can then choose from a diverse stylesheet library an appropriate stylesheet, parse it and apply the resultant stylesheet DOM tree to those XML DOM trees to achieve the desired effect for the web pages. In our WebGen system, the *Web View Manager* is an LotusXSL [IBM Alphaworks 1998] processor, which basically is a query/match engine. It transforms a source tree of nodes, according to instructions and templates specified by a stylesheet tree, into a result tree, which in our case is the browsable HTML page.
- **XSL Stylesheet Library.** XML, as the middle layer representation for a web view, facilitates the interchangeability between the data within the databases and the data presented on the Internet. More importantly, it separates the content from its presentation so to enable different authoring, editing, browsing and viewing tools and allows for the packaging of "document types" that can be shared and reused. In the WebGen system, we select the XML Stylesheet Language (XSL) to specify the association of a presentation style with XML information. XSL consists of a location mechanism (context, selector, pattern, query) capable of addressing portions of the XML structure and an action specification performed on the located content. The XSL stylesheets can be classified into libraries according to the presentation requirements as well as the knowledge about the DTD of the XML data on which it is applied.

5.4. Re-Web graphical user interface

The GUI generally provides a web administrator with access to the other subsystems, such as editing the schema templates and the stylesheets. It also allows users to browse the results and to view the schemas that are stored in the Schema Repository.

- **Template Editor.** To construct a schema modeling a desired web site structure, the web administrator can either choose a suitable view template from the template library, use the *Template Editor* (see figure 10) to write a re-usable and generalized template, or simply write an OQL query transformation. The *Schema Template Processor* then instantiates and executes a given view template using the parameters supplied by the user.
- **Schema Viewer.** The Schema Viewer (see figure 11) provides the capability to review the composition of classes and their relationships within a schema, to evaluate its appropriateness for the desired web view and to adjust or re-structure it accordingly.
- **Layout Editor.** While web site structures represent the content of web pages and the links between them, we also want to have a *Layout Editor* to allow the augmentation of the stylesheet library. The goal here is to make stylesheets reused and applicable to data for multiple output formats.

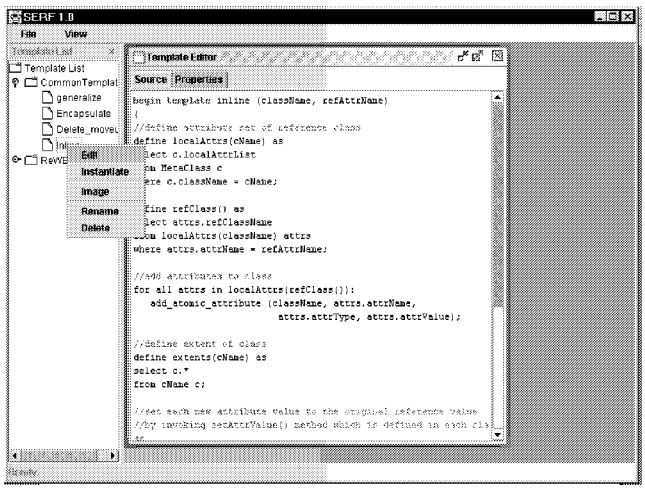

Figure 10. The Template Editor.

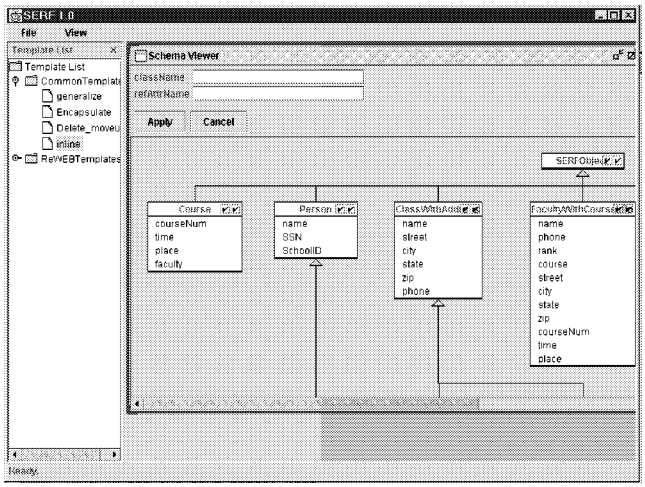

Figure 11. The Schema Viewer.

6. Case study

In this section, we will walk through an example to illustrate the working of our Re-Web system. All displayed web pages have been generated automatically by our Re-Web system.

First, assume the web administrator designs the web site structure of a university's computer science department. The index *Home* page provides three categories of information: Faculties, Courses and ClassRooms, each of which has a link referring to its respective list. The *FacultyList* index page, under the *Faculty* sub-directory relative to the *Home* page, lists all the faculties of this department. Each faculty has a link to a separate page named by the name of the faculty (for example, *Elke* could be the name for her homepage), the personal information, as well as a link to each course she teaches. Vice versa, each *Course* page

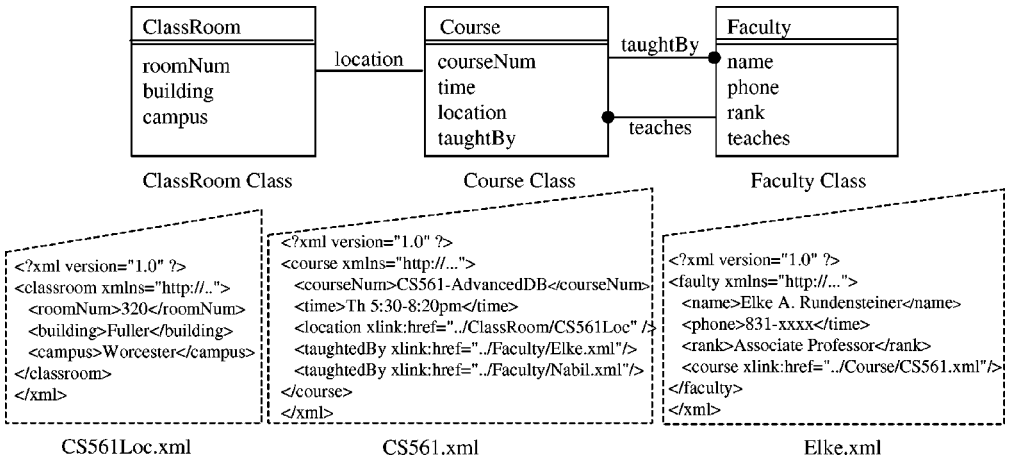

Figure 12. The ODMG data model and corresponding XML files for the original web site.

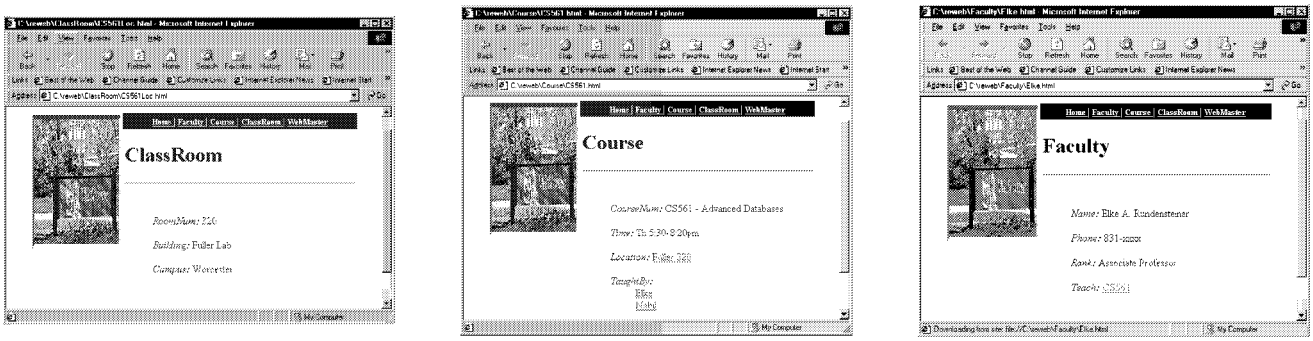

Figure 13. The generated example home pages of original web site.

(that the department provides) has *Faculty* links to all the instructors who are offering the course. With this design of homepages within the department's web site, the web administrator basically captures this web semantics by designing the original OODB schema as depicted in figure 12. The desired web pages can be generated accordingly (see the lower part of figure 12).

In figure 12, we assume there are objects that populate the database already. Assume there is a *Course* object with the courseNum "*CS561*", two faculties are offering this course, one of them is *Elke A. Rundensteiner*, whose personal information is included in a *Faculty* object. The course offered by her is held in *Fuller Lab 320* every *Thursday* night from *5:30 to 8:20 pm*. The classroom location information is encapsulated in a *ClassRoom* object. Thus there exists a bi-directional link between the *Faculty* object and the *Course* object, modeling a (zero to multiple) relationship. Also there is a one-way relationship from the *Course* object to the *ClassRoom* object. Each of these three objects has an associated type in the OODB and the relationships among them follow the OODB schema design.

In the underlying database, all objects of each class are maintained in the class extent and all classes are registered by the schema dictionary. Thus by scanning the schema dictionary, a set of XML files are generated by the Re-Web system. Each of them corresponds to one object from the OODB. We show some of the generated XML files in figure 12. They are *CS561.xml* under the new created subdirectory of *Course*, *Elke.xml* under the subdirectory of *Faculty* and *CS561Loc.xml* under the subdirectory of *ClassRoom*. The XLinks within the XML files capture the relationships between the objects and simulate the navigation mechanism.

Lastly, based on these XML files, the corresponding home pages are then generated using the LotusXSL processor [IBM Alphaworks 1998] provided by IBM (see figure 13). On the top of each page, there is a navigation bar indicating the categories of home pages. Each category has an index page that contains the list of pages that falls into this category. For example, by clicking on the *Faculty* bar, a *Faculty_List* index page is requested containing all links to the faculty members.

However, this web site schema may not meet the user's requirements and the user may instead desire a web view of the same data but with information in a more compact format. For example, by checking a faculty's home page, the user would like to be able to know about not only the personal information of this faculty, but also all the courses she is teaching as well as the location of that course within the same page. Thus the web site needs to be re-structured. The supporting database schema should

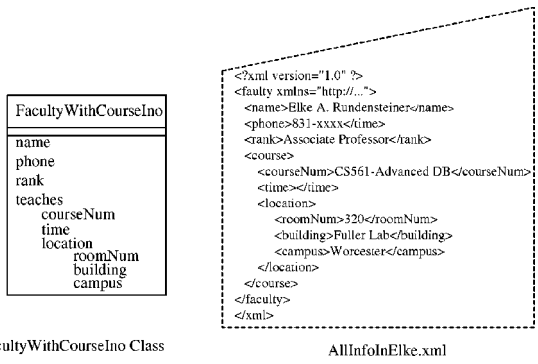

Figure 14. The ODMG data model and corresponding XML files for desired web site.

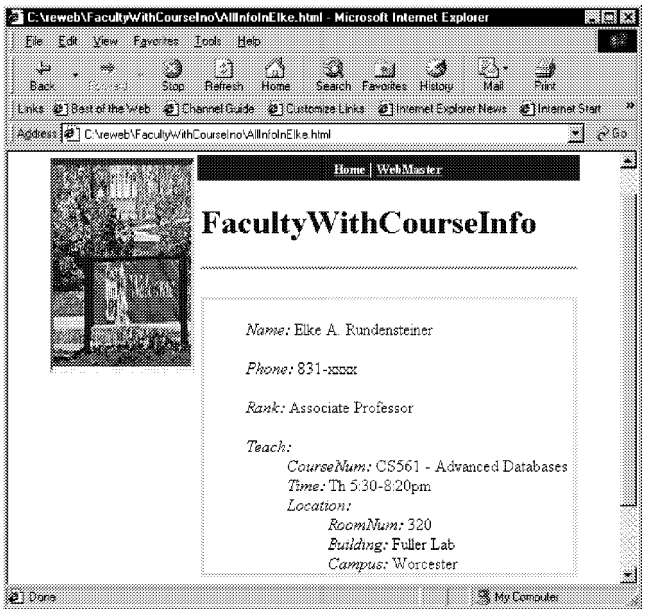

Figure 15. The generated example home pages of desired web site.

be as depicted in the left part of figure 14. After performing the desired view transformation using the chain nesting transformation depicted in figure 7 with the help of the SERF subsystem, the underlying database is restructured to correctly reflect the desired structure of the web site.

Again, using the WebGen subsystem, the restructured database schema and its transformed objects are dumped out and represented in XML format (as shown at the right part of figure 14). The screendump of a resulting example homepage is shown in figure 15. The previous home page for the faculty named "Elke" has been transformed and now directly embeds all the information collected along the chain from itself to its *Course* links and from there to their respective *Classroom* links.

The power of Re-Web lies in its capabilities to do complex transformations, i.e., those defined in the template library or user-defined transformations. The example presented above is one example that shows the ease with which web pages can be re-structured using Re-Web. The above example however, requires that a link or a relationship between the two web pages (or classes) must exist for its successful execution. However, a user may wish to combine information from two disjoint web pages (no link) and present it in one web page.

```
begin template merge_union(Class className1,
                           Class className2,
                           String newCName,
                           Attribute attr1,
                           Attribute attr2,
                           String join_op)
{
    // Retrieve all attributes of class1 and class2
    // from the Schema Repository
    define unionAttributes(cName1, cName2) as
        select distinct c.localAttrList
        from MetaClass c
        where c.className = cName1 or
            c.className = cName2;

    // Create the new merge class
    create_class ($newCName);

    // Add all the attributes to it
    for all attr in unionAttributes($className1,
                                    $className2):
        add_atomic_attribute($newCName,
                             attr.attrName,
                             attr.attrType,
                             attr.attrDefault);

    // Get the extent of a class
    define extent(cName) as
        select c.*
        from c in cName;

    // Do the object transformation
    define new_extent () as
        select obj1.*, obj2.*
        from extent($className1) obj1,
            extent($className2) obj2
        where obj1.$attr1 $join_op obj2.$attr2;

    for all obj in new_extent ():
        $newCName(obj);
}
end template
```

Figure 16. The merge-union template: the structure of the new class is given by a union of the properties of the two source classes.

Figure 16 shows a template which can combine two disjoint classes and produces a view that represents a union of the two classes. Figure 17 shows a template which can combine two disjoint classes and produces a view that represents a difference of the properties of the two classes. This view can now be generated by WebGen to produce a composite web page between two otherwise disjoint web pages. In [Claypool *et al.* 1998b] we provide a detailed case study of several templates to to show the variety of possibilities for re-structuring web information using Re-Web.

```
begin template merge_diff(Class className1,
                          Class className2,
                          String newCName,
                          Attribute attr1,
                          Attribute attr2,
                          String join_op)
{
    // Basic statement to get attributes of a class
    define attributes(cName) as
        select c.localAttrList
        from MetaClass c
        where c.className = cName;

    // Retrieve the difference attributes of class1
    // and class 2 from the Schema Repository
    define diffAttributes(cName1, cName2) as
        select attrs.*
        from attributes(cName1) attrs
        where not
        (attrs exists in attributes(cName2));

    // Create the new merge class
    create_class ($newCName);

    // Add all the attributes to it
    for all attr in diffAttributes($className1,
                                   $className2):
        add_atomic_attribute($newCName,
                             attr.attrName,
                             attr.attrType,
                             attr.attrDefault);

    // Get the extent of a class
    define extent(cName) as
        select c.*
        from c in cName;

    // Do the object transformation
    define new_extent () as
        select obj1.*, obj2.*
        from extent($className1) obj1,
             extent($className2) obj2
        where obj1.$attr1 $join_op obj2.$attr2;

    for all obj in new_extent ():
        $newCName(obj);
}
end template
```

Figure 17. The merge-difference template: the structure of the new class is given by the difference of the properties of the two source classes.

7. Related work

Numerous approaches have been proposed in the literature that try to model semi-structured data such as web pages. They typically invent either some modeling language from scratch or they design some extensions specific to web constructs such as URL, links, ordered lists, etc., to existing languages to enable querying of the web. WebOQL [Arocena and Mendelzon 1998] is one such example with its data model being based on an extended OEM model, namely, hypertrees, with abstractions for references, collections, nesting and ordering to model web structures. WebOQL focuses on *schema-free* data modeling, thus supporting to capture web site structures and the restructuring of sites while querying the web. Our work instead focuses on flexible and re-usable mechanisms for web site construction and management by exploiting a *database-centric* approach. In particular, we generalize standard *schema transformations* and map the underlying restructured schema and data to the alternative web site structure and home pages, instead of generating web pages on the fly using queries over the web.

Our Re-Web approach has been inspired by web site restructuring systems like Araneus [Atzeni *et al.* 1997] and Strudel [Fernandez *et al.* 1997b]. Similar to them, we also exploit the knowledge of a web site's structure for defining alternative views over its content. Araneus's approach is highly-typed: pages in the web site must be classified and formally described before they can be manipulated. Although the Araneus Data Model (ADM) describes page schemes in a manner similar to ours, they utilize relational database technology as back-end data repository. The relational model chosen for the intermediate repository is not the most natural model for capturing the complexity of the web page structures being modeled and thus requires a step of indirection for data flows in both directions, i.e., from and to the web. In Re-Web, we thus advocate the use of object data models and particular the ODMG object model as intermediate modeling paradigm. Hence, the hyper-graph structure of a web site can be modeled by simply associating web semantics with the standard ODMG object model.

To summarize, we now give a comparison between our Re-Web approach with other systems from the perspectives as in table 3.

Table 3
A comparison between related systems.

	Strudel	WebOQL	Araneus	Re-Web
Data source	Semi-structured data	Semi-structured data	Data in RDB	Data in OODB
Data model	OEM	Extended OEM	Relational model	ODMG Model
Query	Graph traversal	Graph traversal	Projection & selection & join	OQL
Restructure	Query & restructure	Query & restructure & render page	Page schema define language	SERF template
Render page	Apply HTML template	No separate render phase	Apply style sheet	Dump DB to XML & apply XSL

8. Conclusions

In this paper we have presented a database-centric approach, called Re-Web, for flexible web site generation, restructuring and maintenance. A large class of web views can be supported using this Re-Web approach. The DBMS in Re-Web, having full knowledge of the layout structure of web views defined over the database, can thus bring standard database techniques to bear for efficiently maintaining web views. The specification of complex, possibly deeply nested OQL queries needed to specify some transformations to map from one view schema to another one is however not trivial. To address this issue and thus ease the process of web site specification and construction, we now propose the notion of generic web view transformations that can be encapsulated into re-usable templates. Generality and re-usability of templates is achieved due to the use of named, typed transformations and the query-based access to the system dictionary, allowing the transformation to both inquire as well as manipulate classes at the schema level at run time.

Our approach is firmly grounded on standard OO modeling and particular the ODMG object model, allowing us on the one hand to both take full advantage of mature database techniques that are likely to be cost-efficient as well as to share new techniques we design to be immediately transferable to other systems that are based on ODMG. We demonstrate in this paper that these generic web view transformations, if collected in a template transformation library, have the potential to represent valuable resource for simplifying the web generation and restructuring process. To the best of our knowledge, Re-Web is the first web site management project focusing on the issue of re-usable view generation templates.

References

Arocena, G. and A. Mendelzon (1998), "WebOQL: Restructuring Documents, Databases, and Webs," In *IEEE Int. Conf. on Data Eng.*, pp. 24–33.

Atzeni, P., G. Mecca, and P. Merialdo (1997), "To Weave the Web," In *Int. Conference on Very Large Data Bases*, pp. 206–215.

Atzeni, P., G. Mecca, and P. Merialdo (1998), "Design and Maintenance of Data Intensive Web Sites," In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98)*, Valencia, Spain, pp. 436–450.

Bray, T., J. Paoli, and C.M. Sperberg-McQueen (1998), "Extensible Markup Language (XML) 1.0," http://www.w3.org/TR/REC-xml.

Cattell, R.G.G. *et al.* (1997), *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann, San Mateo, CA.

Chen, L. and E.A. Rundensteiner (2000), "Aggregate Path Index for Incremetnal Web View Maintenance," In *The 2nd Int. Workshop on Advanced Issues of E-Commerce and Web-based Information Systems*, San Jose, to appear.

Clark, J. (1998), "Jade from James Clark," http://www.jclark.com/jade/.

Claypool, K.T., J. Jin, and E.A. Rundensteiner (1998a), "SERF: Schema Evolution through an Extensible, Re-usable and Flexible Framework," In *Int. Conf. on Information and Knowledge Management*, pp. 314–321.

Claypool, K.T., J. Jin, and E.A. Rundensteiner (1998b), "SERF:Schema Evolution through an Extensible, Re-usable and Flexible Framework," Technical Report WPI-CS-TR-98-9, Worcester Polytechnic Institute.

Claypool, K.T., E.A. Rundensteiner, L. Chen, and B. Kothari (1998c), "Re-usable ODMG-based Templates for Web View Generation and Restructuring," In *WIDM'98*.

Cruz, I.F. and W.T. Lucas (1997), "Delaunay: a Visual Framework for Multimedia Presentation," In *IEEE Symposium on Visual Languages (VL '97)*.

Fegaras, L. (1997), "Optimizing Large OODB Queries," In *Int. Conference on Deductive and Object-Oriented Databases*, pp. 421–422.

Fernandez, M., D. Florescu, J. Kang, A. Levy, and D. Suciu (1997a), "System Demonstration – Strudel: A Web-site Management System," In *ACM SIGMOD Conference on Management of Data*, pp. 549–552.

Fernandez, M., D. Florescu, A. Levy, and D. Suciu (1997b), "A Query Language for a Web-Site Management System," *SIGMOD 26*, 3, 4–11.

Gluche, D., T. Grust, C. Mainberger, and M. Scholl (1997), "Incremental Updates for Materialized OQL Views," In *Proceedings of the 5th DOOD Conference*, pp. 52–66.

IBM Alphaworks (1998), "An Experimental Implementation of the Construction Rules section of the XSL," `http://www.alphaworks.ibm.com/tech/LotusXSL`.

Keller, A. (1982), "Updates to Relational Database Through Views Involving Joins," In *Scheuermann*.

Microsoft Inc. (1998), "XSL support in IE4," `http://www.microsoft.com/xml/xsl/`.

O'Brien, P. (1997), "Making Java Objects Persistent," *Java Report 1*, 1, 49–60.

Rundensteiner, E., K. Claypool, and L. Chen (2000), "SERFing the Web: A Comprehensive Approach for Web Site Management," In *Demo Session Proceedings of SIGMOD'00*.

Scholl, M.H., C. Laasch, and M. Tresch (1991), "Updatable Views in Object-oriented Databases," In *Proceedings of the 2nd DOOD Conference*, pp. 189–207.

Thompson, H. (1998), "XSLJ from Henry Thompson," `http://www.ltg.ed.ac.uk/~ht/xslj.html`.